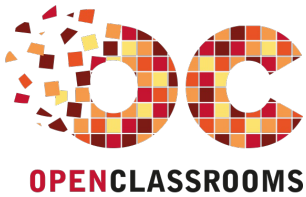
Créez des applications pour BlackBerry

Par Guillaume.



www.openclassrooms.com

Sommaire

Créez des applications pour BlackBerry

Par Guillaume.

Mise à jour : 11/02/2013
Difficulté : Intermédiaire Durée d'étude : 3 mois

Vous avez envie d'apprendre à développer des applications pour BlackBerry ?

Mais vous ne trouvez aucun cours qui vous explique comment procéder ?

Bienvenue dans un cours qui va justement vous apprendre à développer pas à pas des applications pour BlackBerry OS !
Vous remarquerez qu'il est relativement facile de se procurer des ouvrages et tutoriels qui traitent du développement d'applications pour iPhone ou smartphones Android. En revanche sous BlackBerry OS, il est beaucoup plus difficile de trouver des cours à la fois complets et progressifs. Dans ce cours, nous essaierons donc de progresser petit à petit pour vous permettre de réaliser les applications de vos rêves.

À qui ce cours s'adresse-t-il ?

- Ce cours s'adresse aux *initiés en Java*, vous ne pourrez pas suivre ce cours si vous êtes débutant en programmation.
- Aucune connaissance supplémentaire n'est exigée : vous pouvez tout à fait réaliser des applications sans connaître quoi que ce soit sur les requêtes SQL par exemple, tout dépend du type d'applications que vous souhaitez développer.
- Enfin, sachez qu'il ne vous est pas demandé de posséder un smartphone BlackBerry pour suivre ce cours. En effet, il n'est pas indispensable de posséder un tel appareil pour pouvoir développer, tester et publier ses applications.

Tout au long du tutoriel, nous essaierons de mettre en pratique les diverses notions qui auront été mises en évidence. Nous essaierons au cours de ces travaux pratiques de réaliser des applications intéressantes et variées. Je vous laisse découvrir le genre d'applications que nous développerons dans ce cours :

Je vous invite donc sans plus attendre, à vous lancer dans cet apprentissage !

Partie 1 : Les bases du développement BlackBerry

Un petit mot sur BlackBerry

Dans ce chapitre, nous ferons une brève introduction à l'univers BlackBerry. Nous parlerons des smartphones du même nom ainsi que de la compagnie qui en est à l'origine. Nous verrons également les différents moyens de développement d'applications pour leur système d'exploitation **BlackBerry OS**. Dès le prochain chapitre, nous mettrons en place les outils nécessaires au développement de ces applications.
Je rappelle également que la lecture de ce cours suppose que vous avez déjà quelques connaissances en programmation. C'est pourquoi nous ne reviendrons pas sur des notions de base de l'informatique telles que la définition d'un système d'exploitation ou encore sur la signification d'un langage de programmation.

Introduction

BlackBerry est à la base une ligne de smartphones développée par la société canadienne *Research In Motion* (RIM). Cette compagnie a été fondée en 1984 par Mike Lazaridis. Son siège social est à Waterloo en Ontario mais la firme possède des bureaux en Europe, en Asie et en Amérique du Nord. Elle est à l'heure actuelle composée de plus de 13 000 collaborateurs pour un chiffre d'affaires d'environ 16 milliards de $ en 2011. C'est en 1999 qu'elle a commercialisé le premier BlackBerry. Pour plus d'informations, consultez leur site web à l'adresse www.rim.com.

Les smartphones BlackBerry sont réputés pour leur service de messages et courriels en temps réel. Ces téléphones utilisent le système d'exploitation propriétaire **BlackBerry OS**. Il existe de nos jours, près d'une vingtaine de modèles différents de smartphones BlackBerry, et environ 50 millions d'utilisateurs à travers le monde.
Depuis peu, la compagnie canadienne a développé sa première tablette nommée **BlackBerry PlayBook**. Néanmoins, cette tablette ne fonctionne pas sur le même système d'exploitation que les smartphones. Celui-ci est nommé **BlackBerry Tablet OS**.

Les applications

Dans ce cours nous apprendrons pas à pas, à réaliser des applications fonctionnant sous BlackBerry OS. Mais avant de se lancer dans plus de détails sur la manière de les développer, nous verrons un peu ce qu'il est possible de réaliser.

Une application c'est quoi ?

Pour appréhender la notion d'applications, nous allons faire une analogie avec les ordinateurs. Windows 7, par exemple, est un *système d'exploitation*. Celui-ci possède beaucoup de fonctionnalités intégrées, mais il n'est pas rare qu'on ait besoin d'encore plus. Pour cela, nous installons des programmes supplémentaires pour augmenter les fonctionnalités de notre ordinateur.
Voici une liste de programmes ou logiciels :

- Word : logiciel de traitement de texte
- Photoshop : logiciel de retouche d'images
- Blender : logiciel d'infographie 3D

- Eclipse : IDE pour écrire du code

Le système d'exploitation BlackBerry fonctionne sur le même principe. Cependant ici on va parler d'applications. On retrouvera des applications diverses qui servent à un tas de trucs. Voici une liste d'applications :

- Navigateur Web : application pour naviguer sur Internet
- Calculatrice : la calculatrice intégrée à BlackBerry OS est aussi une application
- Monopoly : le célèbre jeu de société disponible sur *BlackBerry App World*

Quelques applications sont présentées dans le tableau ci-dessous :

Navigateur Web	Monopoly	Fruit Ninja
		

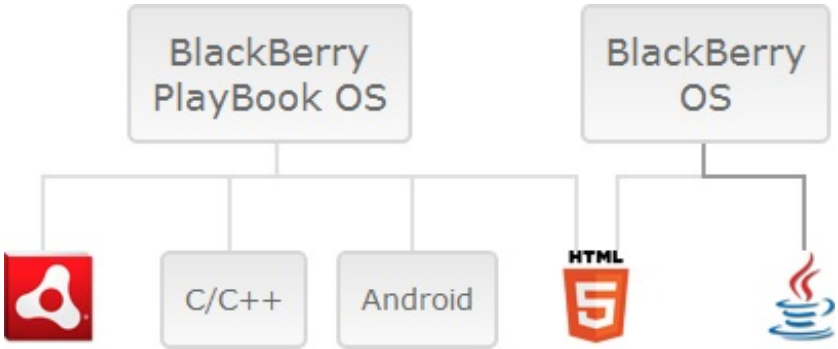
Les outils mis à disposition par RIM nous permettent de créer des applications de toutes sortes. Nous pourrons y intégrer de belles interfaces en utilisant des composants proposés (boutons, menus, etc.), mais également des interfaces graphiques entièrement personnalisées avec des images ou encore utiliser la 3D. Nous aurons aussi tous les outils nécessaires pour par exemple écrire des fichiers dans le terminal ou encore avoir accès aux services de localisation.

Avoir de belles interfaces graphiques est toujours agréable dans une application. La réalisation de telles interfaces demande du travail !
Cependant, beaucoup d'entre vous souhaiteront réaliser des jeux aux graphismes sensationnels. Nous étudierons donc un maximum d'outils permettant la création d'interfaces complexes, qui répondront à vos besoins.

La réalisation d'interfaces graphiques peut nécessiter l'utilisation de logiciels de graphisme tels que Photoshop ou encore Blender, qui ne sont pas l'objet de ce cours. Vous trouverez sur le Site du Zéro des tutoriels correspondant à ces logiciels.

Le développement

La compagnie Research In Motion propose tout un tas d'outils pour réaliser vos applications sur BlackBerry OS et BlackBerry Tablet OS. Le développement d'applications peut être réalisé dans différents langages de programmation. Voici tout ce que la société nous propose :



Voici les possibilités de développement :

- *Adobe AIR* : Il est possible de réaliser des applications à l'aide des outils de développement Adobe Flex, AIR et Flash. Si vous êtes intéressés, Programmez en Actionscript 3 grâce au tutoriel du Site du Zéro.
- *C/C++* : Grâce au développement en C/C++, il est possible de créer des applications de hautes performances et d'intégrer des bibliothèques open source telles que Qt ou OpenAL.
- *Android* : BlackBerry Tablet OS intègre une machine virtuelle qui permet d'exécuter des applications Android.
- *HTML5* : Grâce au BlackBerry WebWorks, vous pouvez développer des applications compatibles sur smartphones et

tablettes.
- *Java* : En utilisant un plugin pour le célèbre IDE nommé Eclipse, vous pourrez réaliser des applications en Java.

En ce qui concerne BlackBerry OS, le développement peut être réalisé uniquement en Java ou en HTML5. Dans notre cas, nous programmerons nos applications en Java tout au long du cours. Dès le prochain chapitre, nous commencerons à nous équiper des outils nécessaires au développement BlackBerry en Java.

Dans ce cours, nous réaliserons des applications pour BlackBerry en *Java*. Pour suivre ce cours, il est donc requis de connaître les bases du Java et de la Programmation Orientée Objet. Pour cela, veuillez suivre au moins les parties I et II du cours sur Java proposé par le Site du Zéro.

- BlackBerry est à l'origine une ligne de smartphones développée par la société canadienne Research In Motion.
- **BlackBerry OS** est le système d'exploitation présent actuellement sur les smartphones.
- Une **application** d'un smartphone est l'équivalent d'un logiciel pour un ordinateur.
- Dans ce cours, nous nous intéresserons au développement d'applications en **Java**.

Installation des outils de développement

Pour réaliser des applications pour BlackBerry, nous aurons besoin de différents outils. L'Environnement de Développement Intégré (ou IDE) conseillé pour le développement d'application BlackBerry en Java est *Eclipse*. Cet environnement de développement est libre, gratuit et multi-plateforme. Nous installerons donc le *Plugin Java BlackBerry* pour Eclipse.
Dans ce chapitre, nous détaillerons toutes les étapes d'installation pas à pas pour ne perdre personne. Une fois terminé, nous aurons alors tout ce qu'il faut pour écrire nos applications, les compiler et les tester à travers un simulateur. Pour ceux qui ne connaîtraient pas l'environnement de développement, nous présenterons rapidement son interface.
Les outils de développement Java BlackBerry fournis par RIM sont malheureusement disponibles *uniquement* sous Windows et Mac.

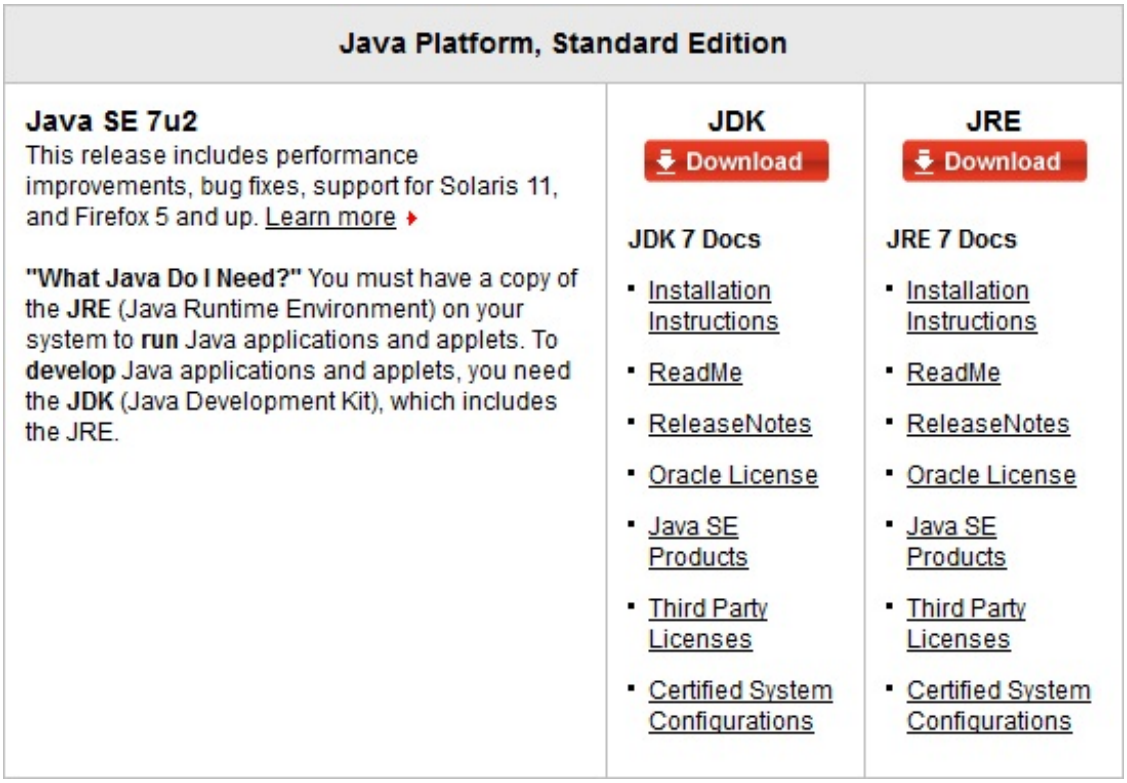
Installer le Java SE JDK

Si vous possédez déjà *Java SE JDK 6 update 14* ou une version ultérieure sur votre machine, vous pouvez passer directement au téléchargement des outils BlackBerry et de l'IDE Eclipse au paragraphe suivant.

Téléchargement du JDK

Les programmes écrits en Java ont la particularité d'être *portables*, c'est-à-dire qu'ils peuvent fonctionner sur différents systèmes d'exploitation. Ceci vient du fait que les programmes Java sont compilés en **Byte Code** et vont s'exécuter sur une machine virtuelle. Cette machine virtuelle est appelée JRE (ou Java Runtime Environment). Nous en aurons besoin car Eclipse est lui-même principalement écrit en Java. Étant donné que les applications BlackBerry sont écrites en Java, nous aurons aussi besoin du JDK (ou Java Development Kit) qui intègre les outils pour développer et compiler du Java. Notez que le JRE est inclus dans le JDK.

Pour télécharger la dernière version du JDK, rendez-vous sur le site d'Oracle à la page suivante. Cliquez alors sur le bouton `Download` du JDK pour passer à l'étape suivante.



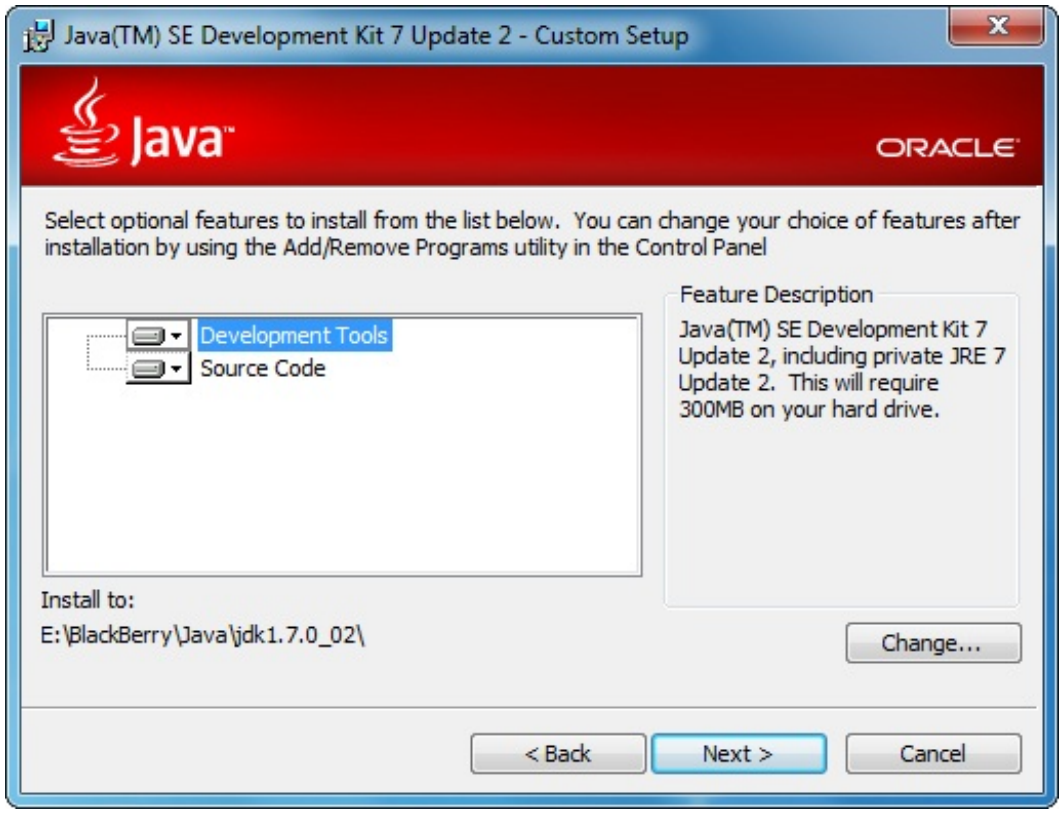
Avant de lancer le téléchargement vous devrez accepter la licence en cliquant sur la case à cocher `Accept License Agreement`, et spécifier votre système d'exploitation dans la liste proposée. Enfin, sélectionnez l'emplacement et enregistrez le fichier d'installation sur votre disque dur.

Installation du JDK

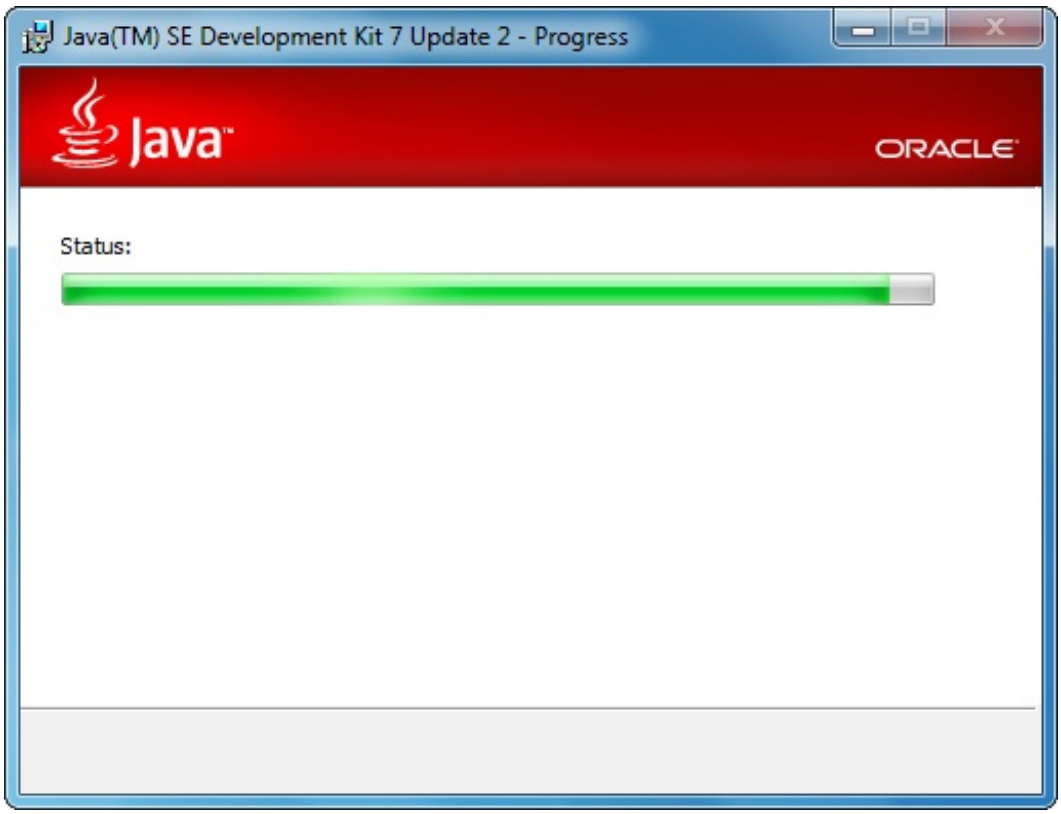
Pour lancer l'installation, double-cliquez sur le fichier d'installation téléchargé précédemment, puis suivez les étapes ci-dessous.

Pour ma part, j'ai créé un dossier BlackBerry, qui contiendra toutes les installations que nous allons réaliser. Je vous conseille vivement de faire la même chose pour éviter d'en semer de partout.

Étape 1 sur 4 : Cliquez sur Next.



Étape 2 sur 4 : Spécifiez le dossier d'installation et cliquez sur Next.



Étape 3 sur 4 : Patientez pendant la durée d'installation.

Étape 4 sur 4 : Terminez l'installation en cliquant sur Continue.

Le JDK est maintenant installé. Nous allons pouvoir installer notre environnement de développement, ainsi que le plugin Java BlackBerry.

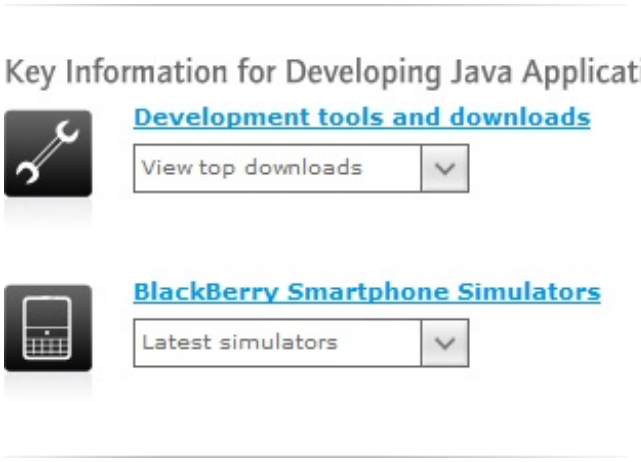
Eclipse et son plugin pour BlackBerry

Eclipse est un IDE qui a été développé pour faciliter l'ajout de *plugins*. Il est à la base configuré pour écrire des programmes en Java, mais il est possible de lui ajouter des modules d'extension pour apporter de nouvelles fonctionnalités au logiciel. Il faut alors télécharger individuellement les plugins désirés et les installer.
Le développement d'applications BlackBerry sous Eclipse IDE nécessite l'ajout d'un plugin. Cependant, Research In Motion propose un kit d'installation qui intègre déjà l'IDE, le plugin ainsi que toutes les bibliothèques de classes. Grâce à lui, une seule

installation est suffisante pour avoir un IDE configuré et prêt à l'emploi.

Téléchargement du plugin

Pour télécharger Eclipse et son plugin Java BlackBerry intégré, nous devons nous rendre sur leur site à l'adresse www.blackberry.com/developers/java, puis suivre les étapes suivantes :



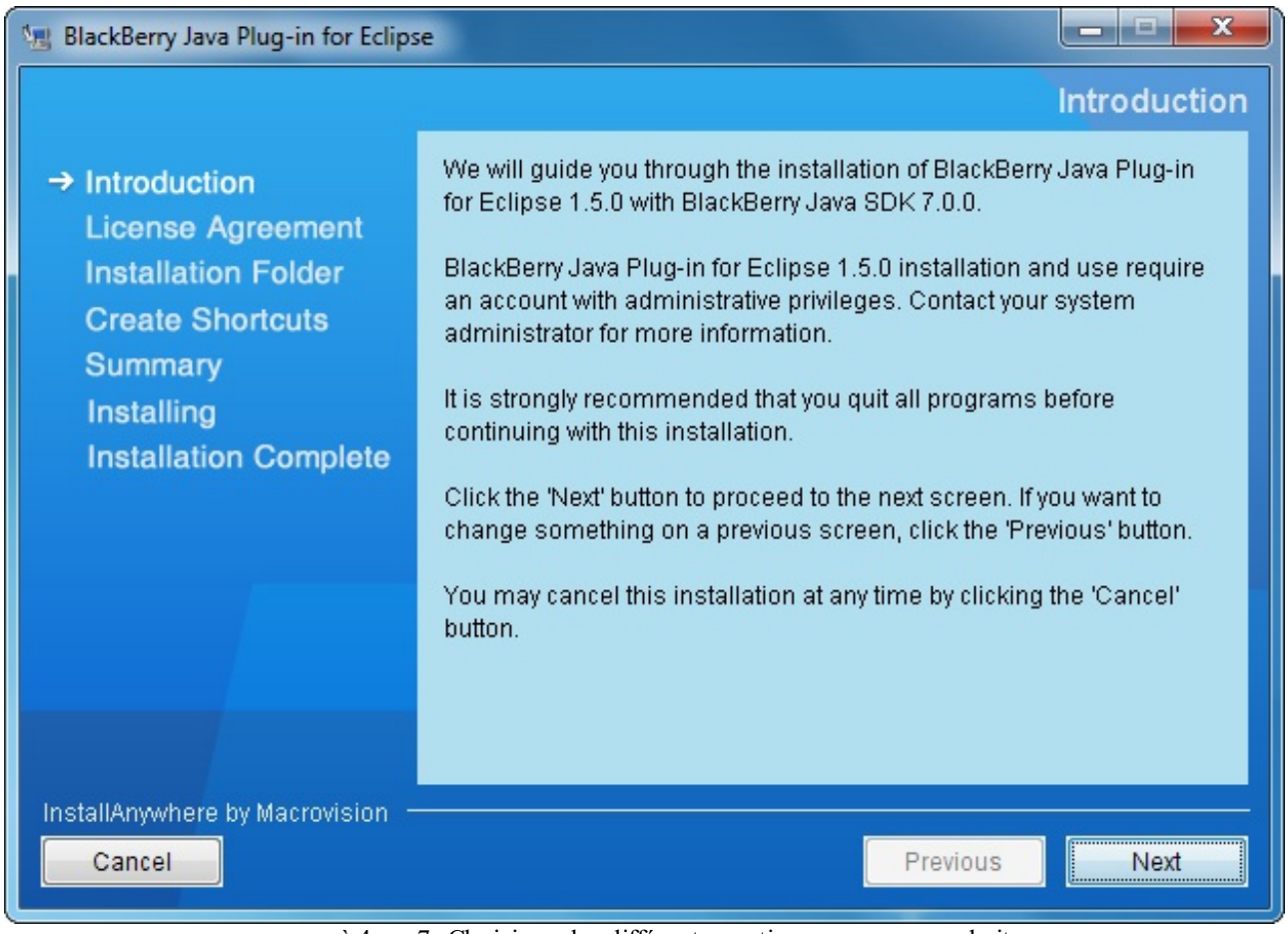
Étape 1 sur 2 : Dans la rubrique Development tools and downloads, sélectionnez votre système d'exploitation.

Étape 2 sur 2 : Cliquez sur Download.

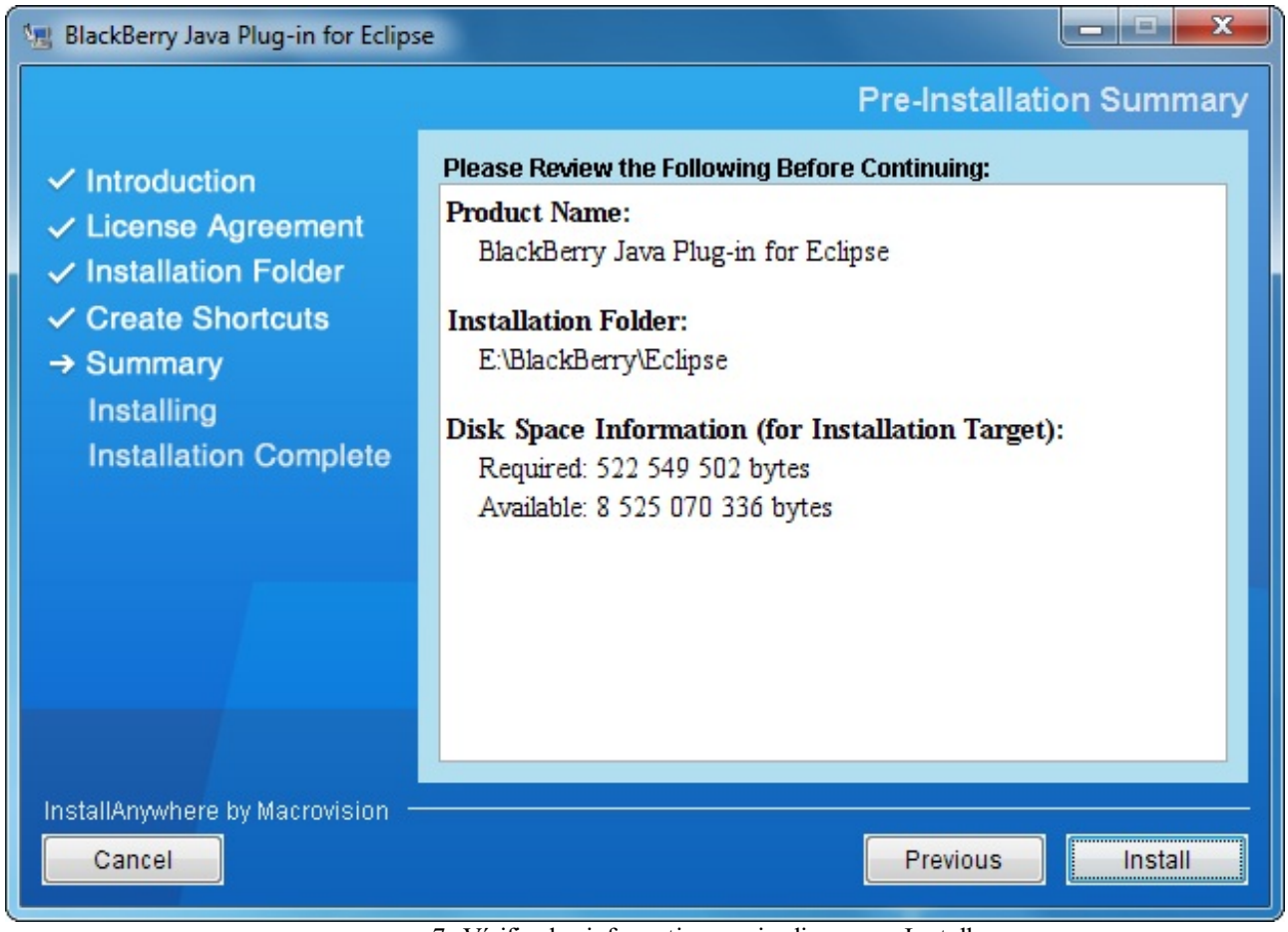
Une fois le téléchargement terminé, vous devriez donc avoir un fichier exécutable si vous êtes sous Windows, ou une archive Zip si vous êtes sous Mac. Dans la suite, je détaillerai les étapes d'*installation de l'IDE sous Windows*. Je laisse le soin aux initiés Mac de décompresser leur archive et de mettre en place leur IDE.

Installation sous Windows

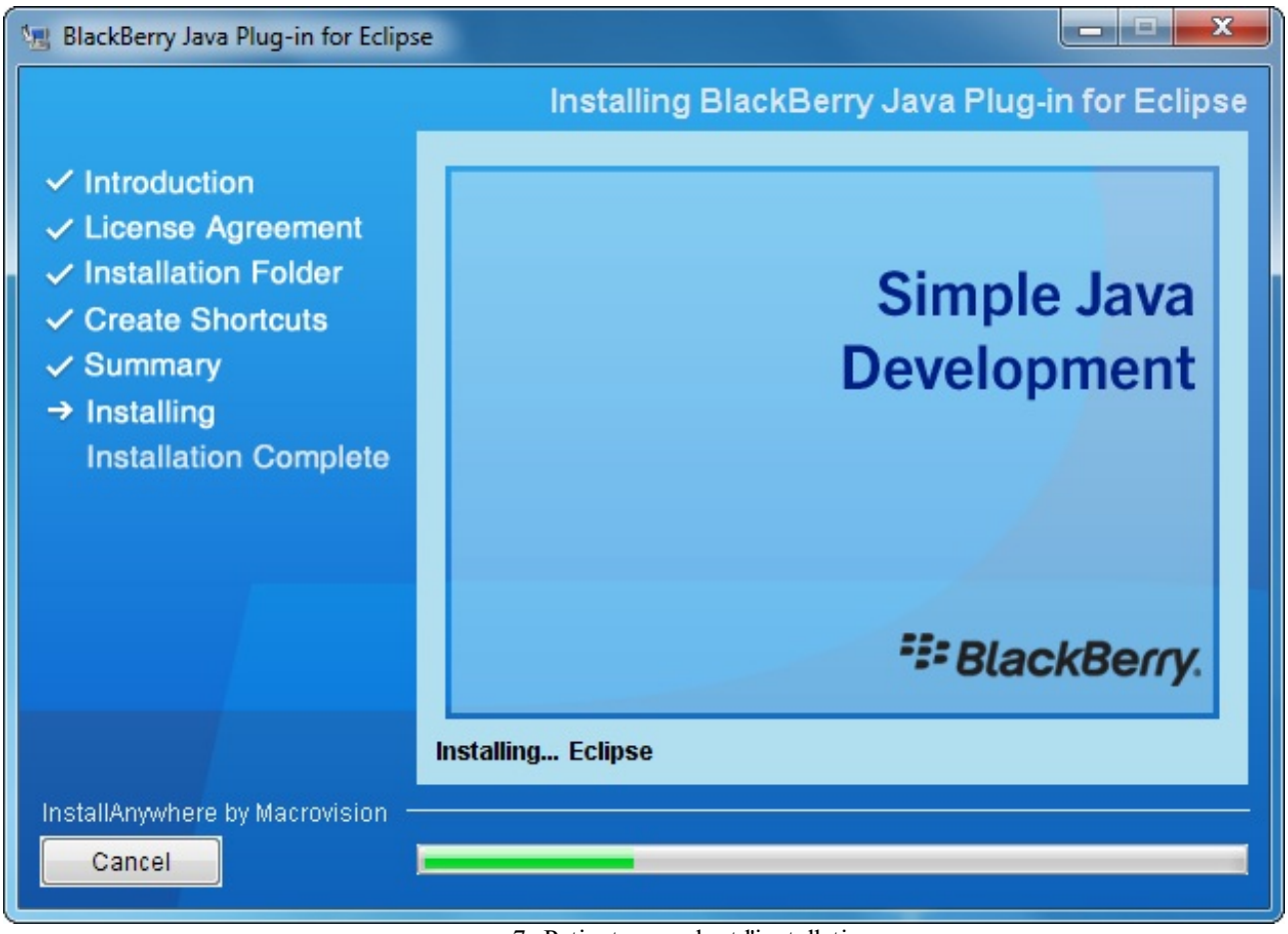
Double-cliquez sur le fichier exécutable pour démarrer l'installation. Une fois celle-ci terminée, nous aurons alors notre environnement de développement Eclipse qui intègre déjà le plugin Java BlackBerry. Les étapes à suivre durant la procédure d'installation sont détaillées ci-dessous.



Étapes 1 à 4 sur 7 : Choisissez les différentes options que vous souhaitez.

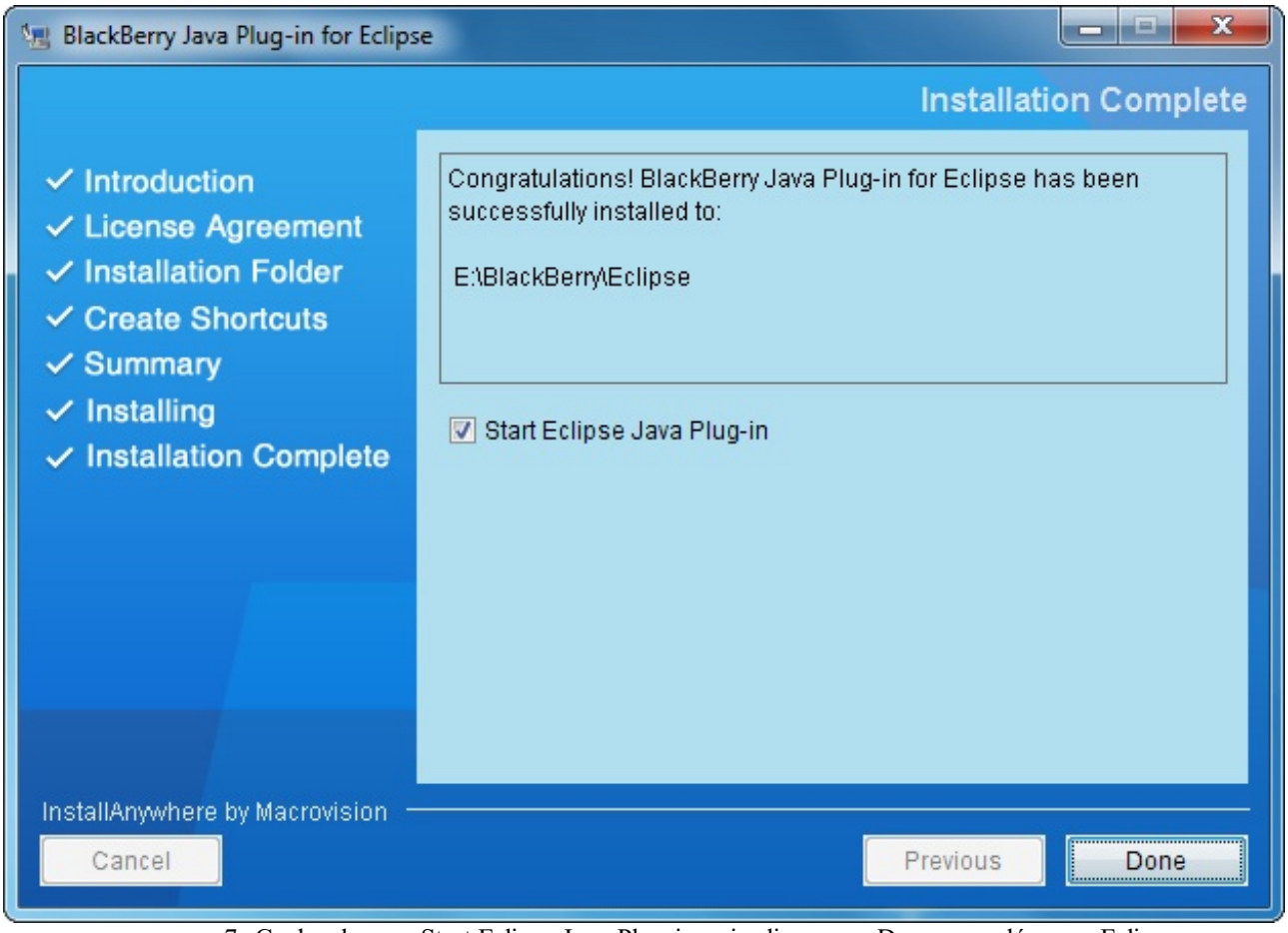


Étape 5 sur 7 : Vérifiez les informations puis cliquez sur Install.



Étape 6

sur 7 : Patientez pendant l'installation...



Étape 7

sur 7 : Cochez la case Start Eclipse Java Plug-in puis cliquez sur Done pour démarrer Eclipse.

L'interface d'Eclipse

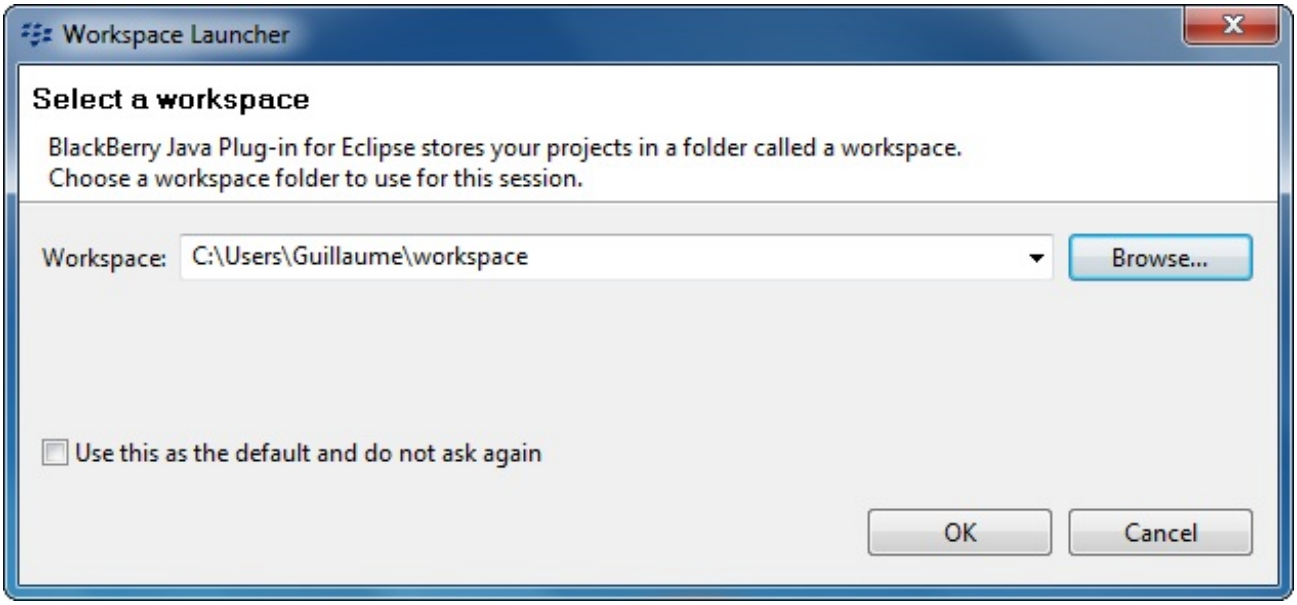
Nous avons maintenant tous les outils nécessaires au développement d'applications BlackBerry. Dès le prochain chapitre, nous

commencerons à écrire nos premières lignes de code. Mais avant cela, nous allons faire un petit tour d'horizon de l'interface d'Eclipse. En effet, il se peut que certains d'entre vous n'aient pas l'habitude de programmer avec Eclipse.

Normalement, vous devriez déjà avoir ouvert Eclipse après l'installation, mais vous pouvez également lancer l'environnement de développement en double-cliquant sur *eclipse.exe* dans le dossier d'installation du logiciel.

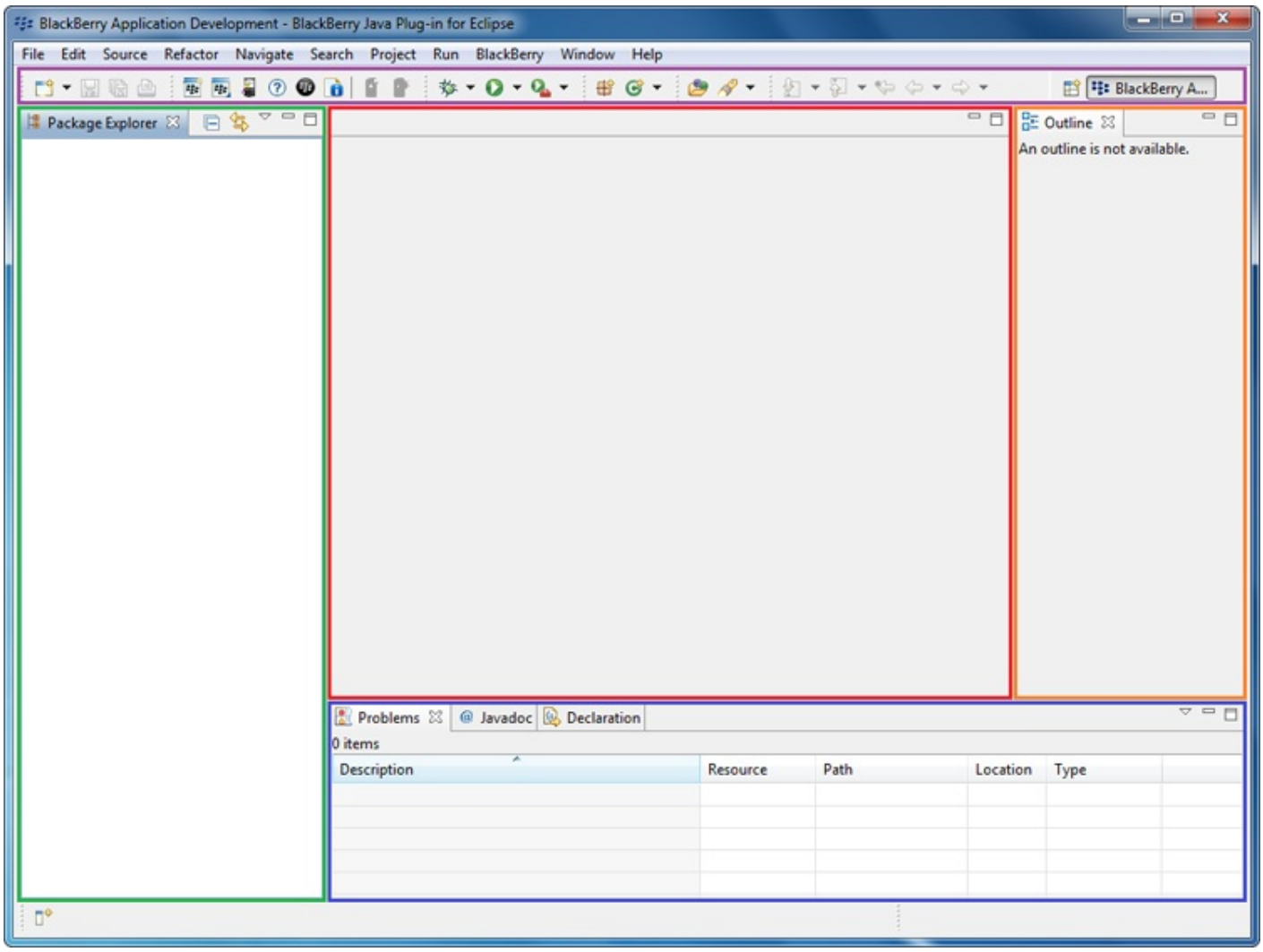
Nous ne ferons ici qu'un rapide tour de l'interface, nous ne verrons donc pas l'intégralité des fonctionnalités du logiciel. Mais je vous conseille grandement de fouiller dans les recoins de l'IDE.

Voici donc ce qui apparaît au lancement du logiciel :



Dans le `Workspace Launcher`, il vous est demandé de renseigner le dossier de destination de l'ensemble de vos projets. Si vous souhaitez conserver le même dossier pour tous vos projets, vous pouvez cocher la case `Use this as default and do not ask again` puis cliquer sur le bouton `OK`.

Nous voilà enfin dans notre environnement de développement. Celui-ci est divisé en plusieurs zones découpées dans la figure ci-dessous.



Voici une liste des fonctionnalités correspondant aux encadrés ci-dessus :

- Tout d'abord en haut, dans l'encadré violet, nous avons la barre de navigation. Nous y retrouvons les commandes les plus courantes présentes dans les différents menus au-dessus. Notamment, nous pouvons créer un nouveau projet, l'enregistrer, ou encore l'exécuter à travers un simulateur. Ces commandes sont régulièrement utilisées, et possèdent pour la plupart des raccourcis clavier. Vous trouverez ces raccourcis dans les menus correspondants.
- À gauche, dans l'encadré vert, nous avons un explorateur de projet. Nous y trouverons les différents dossiers des projets ainsi que leurs contenus : codes sources, images, etc.
- Au centre, dans l'encadré rouge, nous avons un éditeur de code source. C'est ici que nous écrirons nos différentes classes du projet.
- À droite, dans l'encadré orange, nous retrouverons une liste des attributs et méthodes présents dans la classe en cours d'édition.
- Enfin dans l'encadré bleu, en bas, nous avons principalement la console de sortie des programmes ainsi qu'une liste des erreurs éventuelles de vos programmes.

- L'IDE conseillé pour développer des applications pour BlackBerry est **Eclipse**.
- Pour programmer en Java, vous devez disposer d'un JDK.
- RIM propose directement Eclipse et son plugin pour BlackBerry intégré qui comprend l'ensemble des outils dont nous aurons besoin.
- L'interface d'Eclipse est très proche des différents IDE que vous avez déjà pu utiliser.

Votre première application

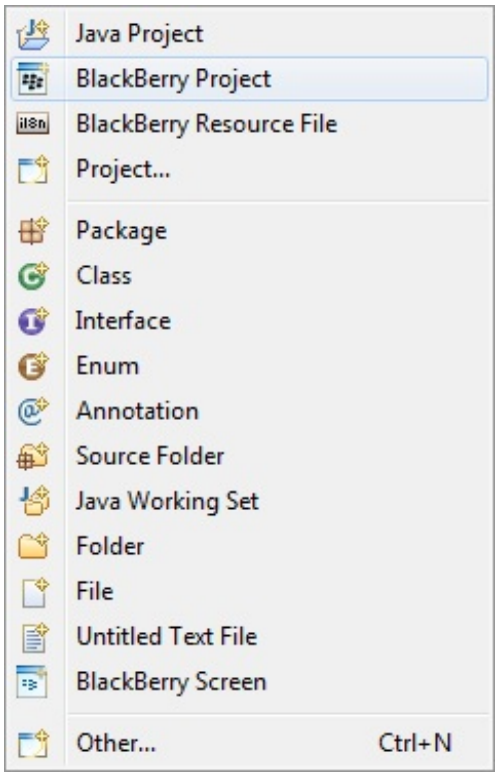
La création du premier projet est une étape importante dans l'apprentissage de la programmation. C'est en général l'occasion de se familiariser avec son environnement de développement et avec les bases du langage utilisé. Ce chapitre n'est pas destiné à réaliser une application complexe, nous aurons tout le temps pour perfectionner nos applications plus tard.
Dans ce chapitre, nous allons :

- Créer notre premier projet BlackBerry dans Eclipse
- Analyser le contenu du projet et le code source de base d'une application
- Lancer notre application dans un simulateur

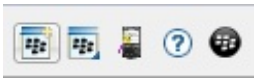
Au cours de ce chapitre, il ne vous est pas demandé de comprendre toutes les subtilités du code. Tout deviendra plus clair au fur et à mesure, à la lecture des chapitres.

Création du projet

Maintenant que nous avons un environnement de développement configuré, nous allons pouvoir créer notre premier projet.
Nous verrons quels sont les différents paramètres qui le définissent. Puis nous décortiquerons son contenu en termes de fichiers et de code source.
Lançons-nous donc dans la création du nouveau projet. Pour cela, cliquez sur `File > New` puis sur `BlackBerry Project` :

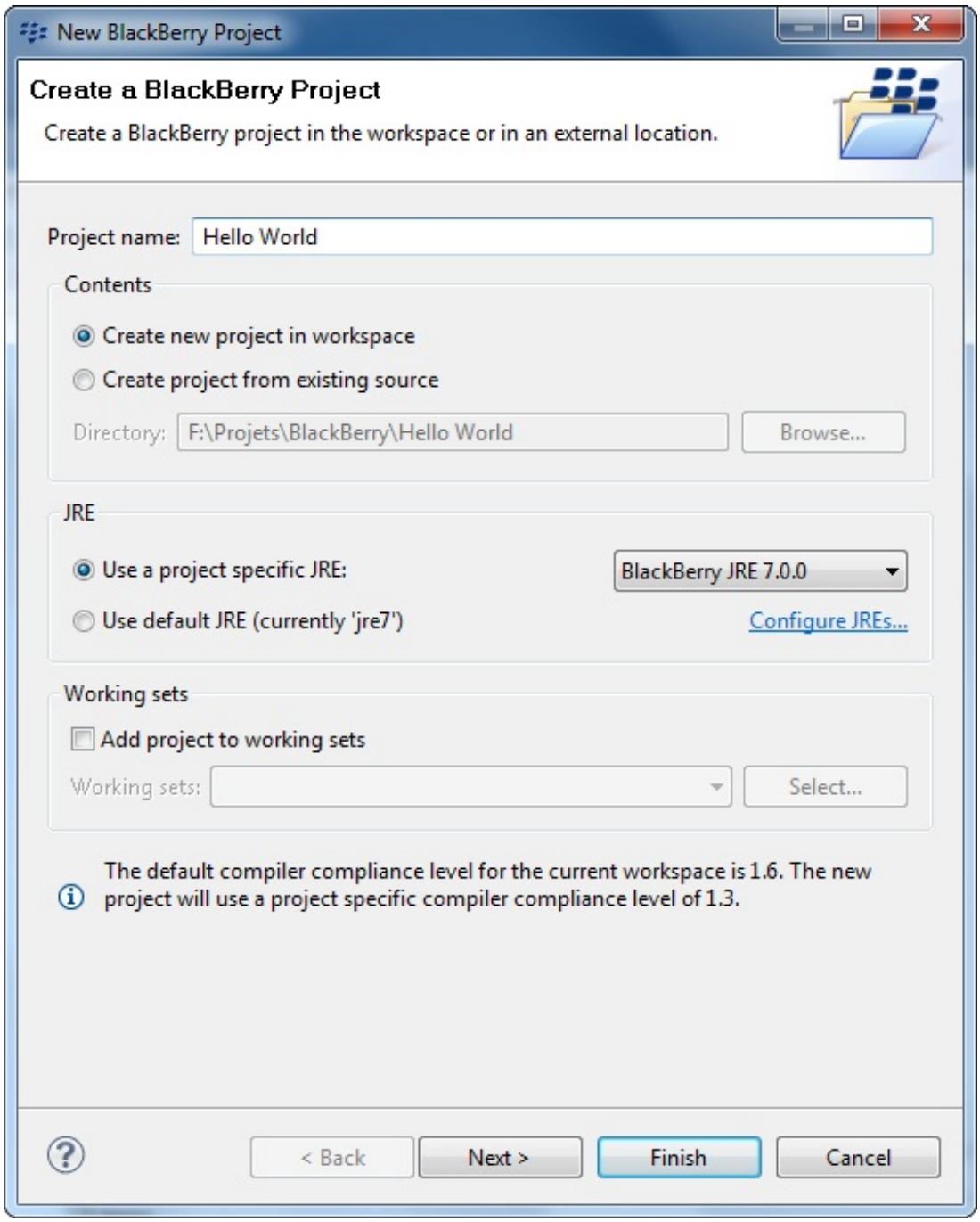


Il est également possible de créer un nouveau projet en cliquant sur l'icône `New BlackBerry Project` de la barre de navigation. Il s'agit du bouton de gauche présenté sur l'image ci-dessous.



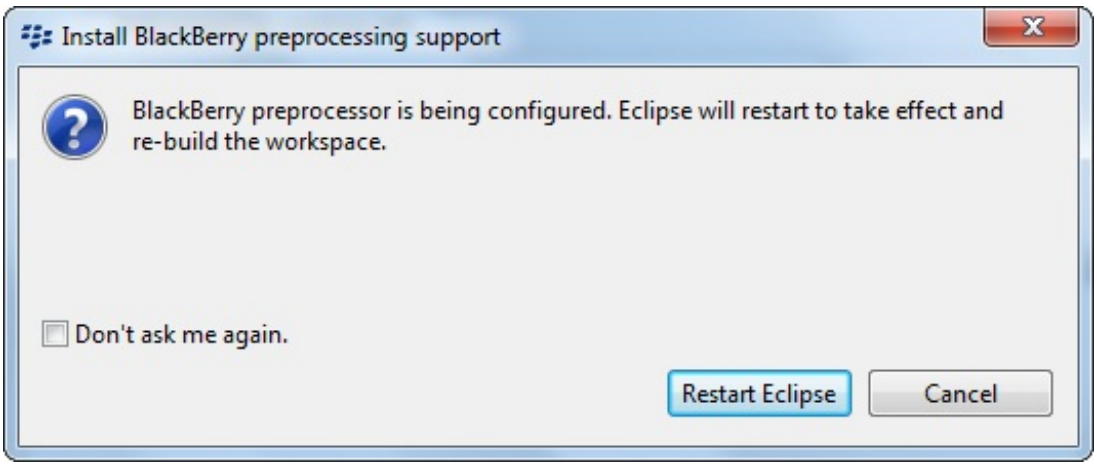
Pour voir à quoi correspondent les boutons de la barre de navigation, il suffit de survoler les icônes avec la souris pour faire apparaître la légende.

Dans le champ `Project name`, entrez le nom du projet : « Hello World » puis cliquez sur `Finish` :

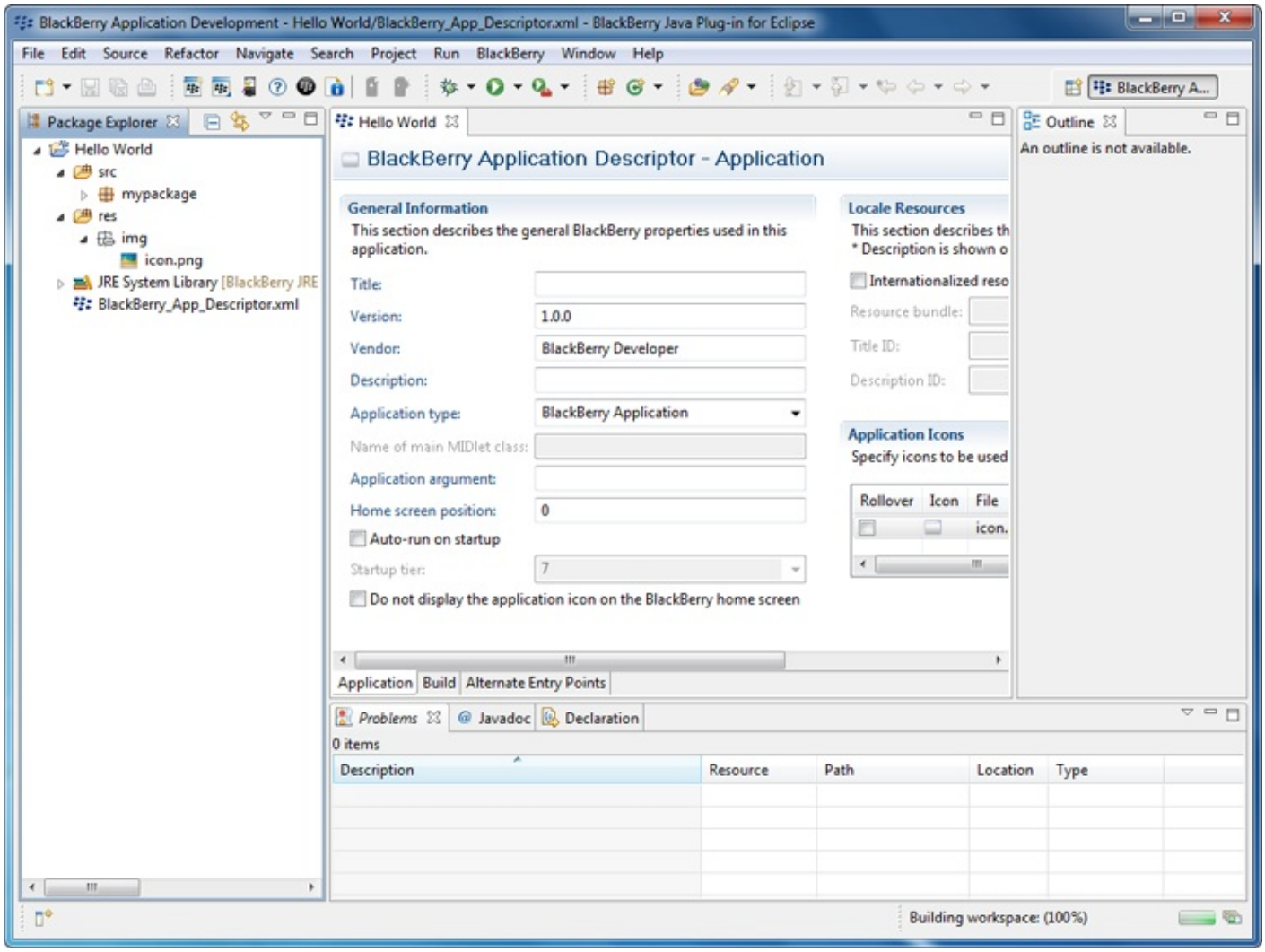


Grâce à cette interface, nous pouvons créer un projet de plusieurs manières. Nous allons nous attarder un peu sur les différents champs. Dans `Project name` nous devons renseigner le nom du projet. Il ne s'agit pas forcément du nom définitif de l'application, mais uniquement du nom du projet dans Eclipse. Dans `Contents` nous avons deux possibilités : créer un nouveau projet à partir de rien ou importer des sources existantes. Enfin dans `JRE` est spécifiée la version du JRE BlackBerry avec laquelle nous créerons le projet.

Il se peut que vous deviez redémarrer Eclipse pour que celui-ci reconfigure l'espace de travail. Vous aurez alors le message suivant :

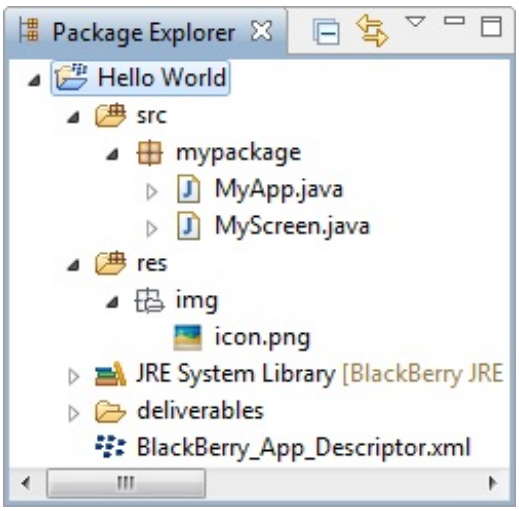


Une fois le projet créé, vous devriez normalement le voir apparaître dans l'onglet `Package Explorer` avec le nom que vous avez choisi. Nous étudierons son contenu dans le paragraphe suivant, même si celui-ci est très proche d'un projet Eclipse quelconque. Au centre de l'IDE est ouvert le fichier XML de description de l'application. Nous reviendrons sur ce type de fichier plus tard, et nous étudierons les différents champs qui le composent.
L'environnement de développement devrait donc ressembler à ceci :



Analyse du contenu

Dans ce paragraphe, nous étudierons le contenu de notre projet. En premier lieu, nous allons analyser son arborescence dans l'onglet `Package Explorer`. Pour cela, déroulez les différents dossiers en cliquant sur la petite flèche à gauche de ceux-ci.



Voici donc les deux dossiers principaux que nous trouvons dans le projet :

- `src` : Il s'agit du dossier source, qui contiendra le code source écrit avec différentes classes rangées en package. Nous avons pour l'instant deux classes nommées `MyApp` et `MyScreen`, dont nous étudierons le contenu juste après.
- `res` : Dans ce répertoire, nous mettrons l'ensemble de nos ressources. Actuellement, il ne contient qu'une seule ressource, qui est une image. Il s'agit de l'icône de notre application telle qu'elle apparaîtra sur le smartphone.

Si vous visualisez le fichier `icon.png`, vous devriez voir cette image : 

Maintenant nous allons étudier le code source de ce projet de base. Nous verrons alors le code minimal requis pour lancer une application dans BlackBerry OS.

La classe MyApp

Je vous propose d'ouvrir le fichier `MyApp.java` dans l'éditeur de code en double-cliquant dessus.

Code : Java - MyApp.java

```
package mypackage;

import net.rim.device.api.ui.UiApplication;

/**
* This class extends the UiApplication class, providing a
* graphical user interface.
*/
public class MyApp extends UiApplication
{
    /**
* Entry point for application
* @param args Command line arguments (not used)
*/
    public static void main(String[] args)
    {
        // Create a new instance of the application and make the
currently
        // running thread the application's event dispatch thread.
        MyApp theApp = new MyApp();
        theApp.enterEventDispatcher();
    }


    /**
* Creates a new MyApp object
*/
    public MyApp()
    {
```

```
            // Push a screen onto the UI stack for rendering.
            pushScreen(new MyScreen());
        }
    }
```

Pour comprendre ce code, nous l'analyserons ligne par ligne. Tout d'abord, nous déclarons que notre programme se situe dans le package `mypackage`.

Code : Java

```
package mypackage;
```

Pour créer une interface utilisateur (ou **UI**), nous devons créer une classe qui hérite de la classe `UiApplication`. Nous pourrons ainsi afficher des choses à l'écran et interagir avec l'utilisateur. Voici donc le code correspondant :

Code : Java

```
public class MyApp extends UiApplication
{
    ...
}
```

Pour pouvoir utiliser la classe `UiApplication`, nous devons l'importer. Celle-ci se trouve dans le package `net.rim.device.api.ui`. Pour importer cette classe, il nous faudra écrire la ligne de code suivante :

Code : Java

```
import net.rim.device.api.ui.UiApplication;
```

Le point d'entrée de l'application est la méthode `main()`. À l'intérieur de celle-ci, nous devrons créer une instance de notre application `MyApp`.
Au lancement de l'application, un *Thread* (ou pile d'exécution) sera lancé. Ce dernier fera appel à la méthode `enterEventDispatcher()` et deviendra l'*event-dispatching Thread* . Ce qu'il faut comprendre, c'est que nous pourrons ainsi dessiner à l'écran et gérer les évènements. Tout ceci est peut-être un peu vague pour l'instant, mais nous reviendrons sur ces notions dans la partie suivante. En attendant, utilisez la méthode `main()` telle qu'elle vous est proposée :

Code : Java

```
public static void main(String[] args)
{
    MyApp theApp = new MyApp();
    theApp.enterEventDispatcher();
}
```

Enfin pour finir, nous devons créer un constructeur à notre classe `MyApp`. Dans ce constructeur, nous allons tout simplement afficher un écran défini par une nouvelle classe `MyScreen`, écrite dans le fichier `MyScreen.java`. Voici le code correspondant :

Code : Java

```
public MyApp()
{
    pushScreen(new MyScreen());
}
```

Cependant avant de la « pousser » sur l'écran avec la méthode `pushScreen()`, l'interface devra avoir été totalement définie. Il faudra donc la décrire entièrement à l'intérieur du constructeur de la classe `MyScreen`.

La classe MyScreen

Nous allons détailler maintenant le fichier `MyScreen.java` qui va nous servir à décrire notre interface affichée à l'écran. Celui-ci se situe également dans le package `mypackage`. Voici le code complet de cette classe :

Code : Java - MyScreen.java

```
package mypackage;

import net.rim.device.api.ui.container.MainScreen;

/**
* A class extending the MainScreen class, which provides default
standard
* behavior for BlackBerry GUI applications.
*/
public final class MyScreen extends MainScreen
{
    /**
* Creates a new MyScreen object
*/
    public MyScreen()
    {
        // Set the displayed title of the screen
        setTitle("MyTitle");
    }
}
```

Pour créer des interfaces, notre classe `MyScreen` doit hériter de la classe `MainScreen`. Celle-ci doit être importée du package `net.rim.device.api.ui.container`. Voici le code correspondant :

Code : Java

```
package mypackage;

import net.rim.device.api.ui.container.MainScreen;

public final class MyScreen extends MainScreen
{
    ...
}
```

Comme nous l'avons dit précédemment, l'ensemble de l'interface doit être défini dans le constructeur de la classe `MyScreen`. Cette dernière contiendra donc uniquement son constructeur que voici :

Code : Java

```
public MyScreen()
{
    setTitle("MyTitle");
}
```

Actuellement, l'application se contente uniquement d'afficher son titre : « MyTitle ». Nous pourrions peut-être essayer de faire un peu mieux en ajoutant un petit texte de bienvenue. Voici donc le constructeur que je vous propose :

Code : Java

```
public MyScreen()
```

```
    {
        setTitle("Site du Zéro");
        add(new RichTextField("Salut les zéros!"));
    }
```

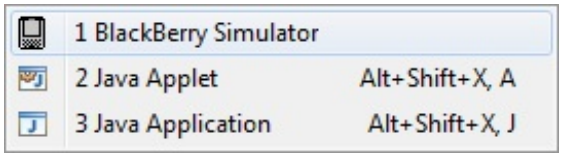
Il ne faudra pas oublier d'importer la classe `RichTextField`. Je vous propose donc le code complet de la classe modifiée :

Code : Java

```
package mypackage;

import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.MainScreen;

public final class MyScreen extends MainScreen
{
    public MyScreen()
    {
        setTitle("Site du Zéro");
        add(new RichTextField("Salut les zéros!"));
    }
}
```

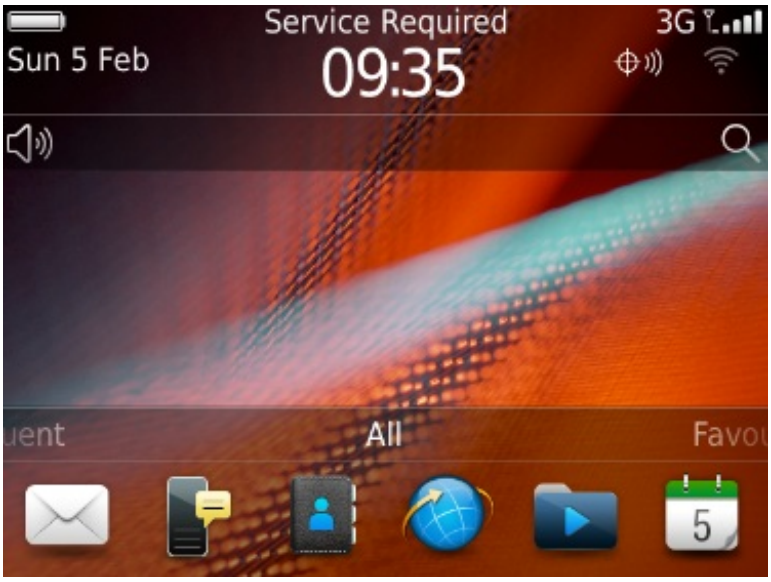
Lancement de l'application

Maintenant que notre projet est prêt, nous allons pouvoir le tester. Il est bien entendu possible de le faire sur un appareil physique, cependant il n'est pas nécessaire de posséder un smartphone BlackBerry pour lancer ses applications. Dans ce cours nous utiliserons un **simulateur** ou **émulateur**. Pour lancer votre application dans le simulateur BlackBerry, commencez par cliquer sur `Run > Run As > BlackBerry Simulator` :

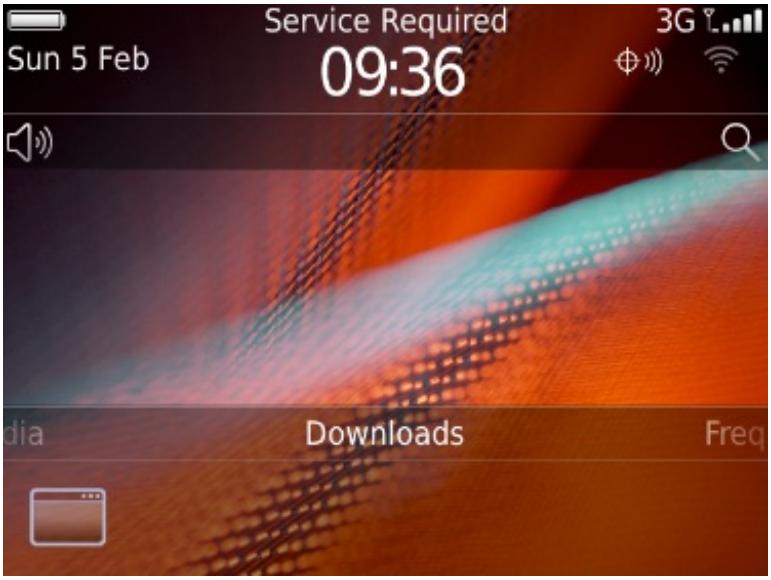


Patientez le temps que l'OS démarre...

Vous l'aurez sûrement remarqué, mais le simulateur met un certain temps à démarrer. C'est pourquoi, si vous avez plusieurs tests à réaliser, il peut être préférable de laisser le simulateur en marche. Ainsi vous ferez l'économie du temps de démarrage du système d'exploitation.



Une fois démarré, allez dans la catégorie `Downloads` présentée ci-dessous :



Enfin dans `Downloads`, cliquez sur l'icône de votre application.

Vous l'aurez remarqué, l'icône de votre application correspond à l'image `icon.png` du répertoire `res` présentée plus haut. Après avoir cliqué sur votre application dans le dossier `Downloads`, celle-ci devrait se lancer dans votre simulateur. Voici ce que vous devriez avoir sur l'écran :

Le simulateur BlackBerry possède un certain nombre de paramètres et l'émulateur peut avoir des apparences diverses suivant la version de votre plugin BlackBerry. Nous ne détaillerons pas la manière d'utiliser le simulateur, je pense que vous devriez être capable de vous en sortir tout seul. Naviguez dans les différents menus, et vous verrez l'ensemble des possibilités du simulateur BlackBerry.

- Pour développer une application, vous devez créer un nouveau projet de type `BlackBerry Project`.

- De base, un projet est déjà composé de deux classes : `MyApp` et `MyScreen`.
- La classe de base `MyApp` hérite de `UiApplication` et permet de créer une application.
- Pour gérer les interfaces graphiques, la classe `MyScreen` hérite de `MainScreen`.
- Le point d'entrée pour votre code est alors à l'intérieur du constructeur de la classe `MyScreen`.
- Pour tester une application nous utiliserons un **simulateur** grâce à la commande `Run > Run As > BlackBerry Simulator`.
- L'application compilée est alors placée dans la catégorie `Downloads` du simulateur.

Partie 2 : Les interfaces graphiques

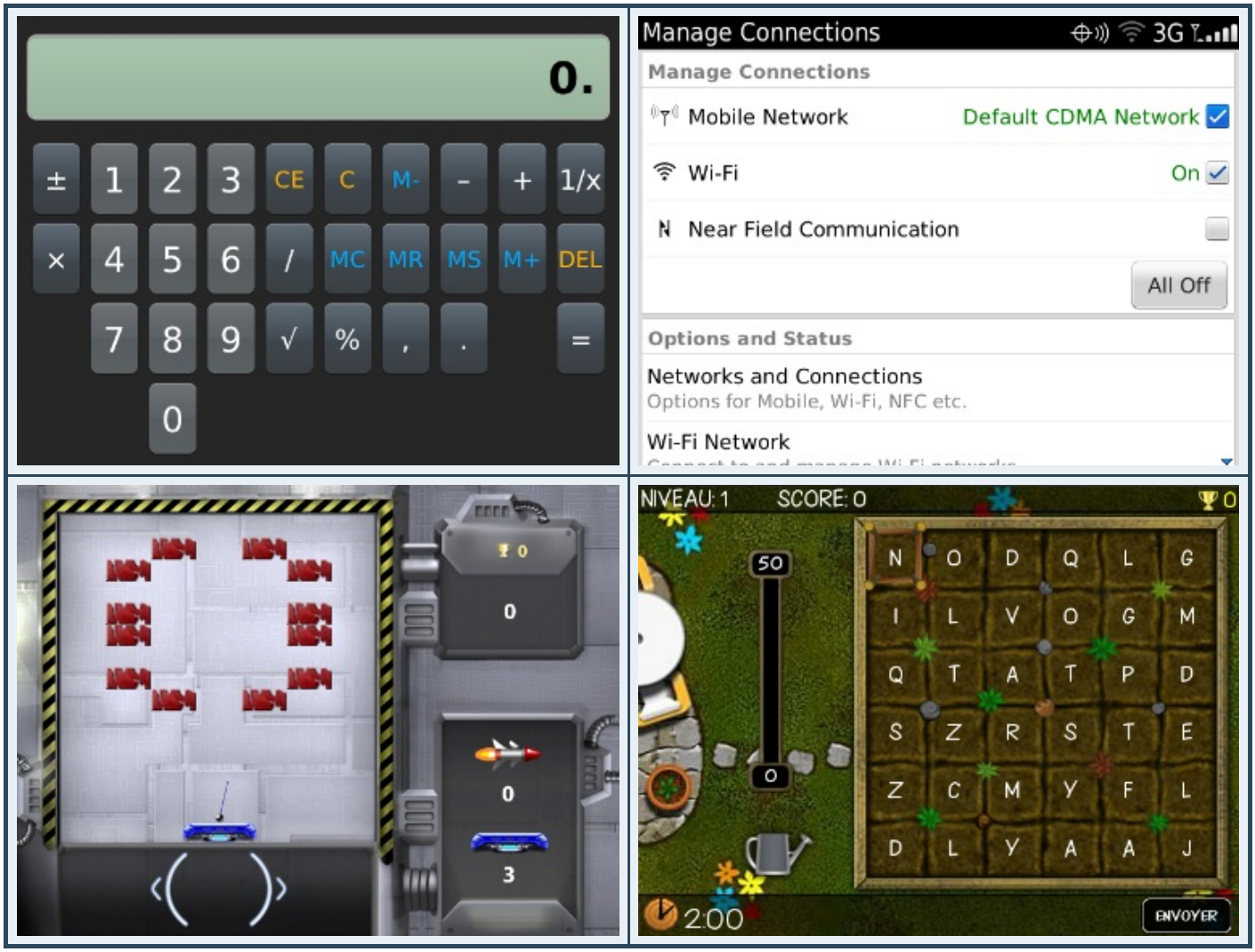
La constitution des vues

Nous voici arrivés dans le premier chapitre de la partie portant sur les **UI** pour User Interface ou interface utilisateur.
Nous allons tout au long de cette partie découvrir comment afficher des objets sur l'écran et ainsi pouvoir créer des applications correspondant visuellement à vos attentes. Pour démarrer en douceur dans ce chapitre, nous allons découvrir les principes de base des interfaces utilisateur.
Sans plus attendre, démarrons cette partie passionnante !

Introduction

Comme promis nous allons démarrer en douceur cette partie. Au cours de ce chapitre nous n'entrerons pas vraiment dans les lignes de code, mais nous verrons plutôt les grands principes de création d'interfaces graphiques.
Lorsque nous créons des applications, une grosse partie du développement peut être tournée vers le graphisme. Principalement dans les jeux, vous aurez besoin de créer une interface graphique complexe que l'utilisateur aura à l'écran. Suivant le type d'applications, ces interfaces seront différentes et pourront être réalisées de différentes façons.

Pour illustrer tout ce qui vient d'être dit, voici quelques exemples d'interfaces graphiques :



Notez les différences de contenu de ces différentes applications. La première est une calculatrice, vous constaterez qu'elle est principalement constituée d'un « affichage numérique » ainsi que de boutons. Cette interface graphique a été conçue à l'aide de composants prédéfinis, ce qui facilite la création de telles interfaces. Nous retrouvons parmi ces composants des boutons, des champs de texte, des listes et pleins d'autres ; nous y reviendrons !
Juste à côté le menu des réglages intègre également des composants de base comme des champs de texte, des images et différents types de boutons.
Enfin les deux derniers sont des jeux, vous remarquerez que ces interfaces sont « dessinées », principalement à l'aide d'images.

Ainsi chaque style d'interface graphique est réalisé avec une technique spécifique, que nous verrons avant la fin de ce chapitre.

Les classes fondamentales

Nous allons à présent voir les différentes classes à la base de tous les composants graphiques. Celles-ci sont contenues dans le package `net.rim.device.api.ui`. En particulier nous verrons les trois classes présentées ci-dessous. Tout ce que nous verrons par la suite découle de ces classes par héritage :

- Field
- Manager
- Screen

Commençons tout de suite par la première : la classe `Field`.

Les Fields

La classe `Field` est la classe fondamentale de tout composant graphique d'une application. En effet tous les composants graphiques tels que les boutons, les champs de texte ou autres héritent de cette classe.
Nous pourrions voir ceci comme une zone rectangulaire traçable à l'écran à l'intérieur de ce que nous appelons des conteneurs.

La classe `Field` est une classe abstraite, dans le sens où elle ne peut pas être utilisée directement. En général, il est préférable d'utiliser les composants prédéfinis qui héritent de cette classe. Toutefois, il peut être envisageable de créer un composant personnalisé en créant une nouvelle classe héritant de la classe `Field`.

Comme cela vient d'être dit, nous utiliserons donc jamais en pratique cette classe telle quelle. En revanche, ce qu'il intéressant de noter c'est que tous les éléments graphiques sont donc des classes filles de cette classe `Field`. Celles-ci hériteront donc de toutes ses propriétés et méthodes.
Ainsi nous pourrons par exemple utiliser les propriétés `FIELD_LEFT`, `FIELD_HCENTER` ou `FIELD_RIGHT` pour positionner l'objet à l'écran. Bien évidemment nous ne détaillerons pas ici toutes les propriétés et méthodes de cette classe. Notez cependant les méthodes `getHeight()` et `getWidth()` qui permettent de récupérer la largeur et la hauteur de l'objet `Field`.

Je pense en avoir assez dit sur cette classe, nous allons donc à présent nous intéresser aux classes `Manager` et `Screen` qui elles-mêmes héritent de la classe `Field`.

Les Managers

Grâce à la classe `Field`, nous avons pu mettre en évidence les bases de tout composant graphique. Ceux-ci peuvent être des champs de texte, de saisie de texte ou encore des boutons, d'ailleurs le prochain chapitre vous introduira les composants les plus utilisés. Ces composants sont indispensables pour créer des vues mais ne suffisent pas. C'est là qu'entrent en jeu les Managers ou conteneurs.
Comme son nom l'indique, un conteneur sert à *contenir* d'autres éléments, c'est-à-dire des composants mais également d'autres conteneurs. Nous aurons également le temps de revenir sur le concept de conteneur dans un chapitre qui leur est dédié.

En attendant, sachez que ces conteneurs permettent de mettre en place les différents composants à l'écran tels que vous pouvez le voir sur cette application que vous connaissez très certainement :

L'application Facebook

Pour ceux qui auraient déjà réalisé de la programmation d'interfaces graphiques en Java, sachez que le principe est ici exactement le même. Les objets graphiques que nous verrons sont très similaires à ceux utilisés en Java. Nous retrouverons donc les mêmes composants basiques, cependant ceux-ci peuvent avoir un nom légèrement différent.

Les Screens

Enfin pour finir sur le côté « technique » de ce chapitre, nous allons étudier la classe `Screen` ainsi que des classes héritant de celle-ci. Il s'agit de l'élément principal de l'affichage, c'est-à-dire que c'est lui qui englobe l'ensemble des éléments à tracer. D'ailleurs il s'agit d'une classe fille de la classe `Manager` vue précédemment ; en effet puisqu'elle sert de conteneur pour l'ensemble des éléments.
Pour vous prouver que cette classe `Screen` est à la base de tout affichage, sachez que nous l'avons déjà utilisé. Effectivement, si vous regardez bien nos codes précédents, vous verrez apparaître une classe fille de celle-ci qui se nomme `MainScreen` :

Code : Java

```
/**
* A class extending the MainScreen class, which provides default
standard
* behavior for BlackBerry GUI applications.
*/
public final class MyScreen extends MainScreen
{
 public MyScreen()
  {
  }
}
```

Comme nous venons de le voir, il existe différentes classes pour créer un écran. Voici les trois principales : `MainScreen` héritant de la classe `FullScreen` elle-même héritant de la classe `Screen`. Celles-ci se distinguent principalement par leur contenu par défaut. Voici une brève description de chacune d'entre elles :

Classe	Description
`Screen`	Il s'agit de la classe de base : vous pouvez créer des écrans en utilisant un conteneur.
`FullScreen`	Elle permet également de créer une vue à partir d'un écran vide, mais avec ici un conteneur de type vertical.
`MainScreen`	Cette classe permet de créer des écrans possédant déjà quelques éléments standards : • un titre d'écran • un conteneur de type vertical avec défilement (nous reviendrons là-dessus dans le chapitre consacré aux conteneurs) • un menu par défaut

En pratique, nous utiliserons principalement la classe `MainScreen` : il s'agit de la classe privilégiée pour un premier écran, cependant rien ne vous empêche d'utiliser l'une des deux autres présentées ci-dessus.

Maintenant que nous connaissons les différents éléments de base, je vais vous présenter les différentes méthodes que nous étudierons dans ce cours pour concevoir des interfaces graphiques.

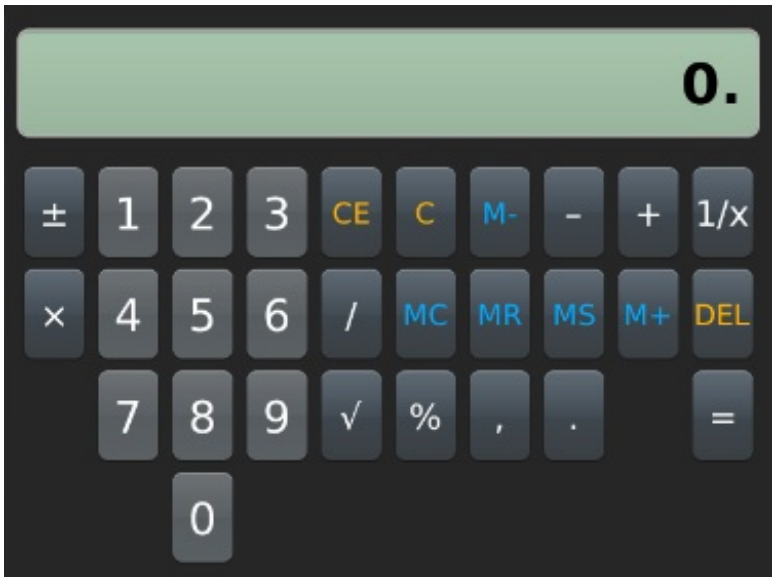
Les différentes techniques

Comme nous avons vu, la conception d'interfaces graphiques peut être réalisée de différentes manières. Nous distinguerons tout d'abord l'utilisation de composants prédéfinis et prêts à l'emploi, mais il nous est également possible de dessiner directement à l'écran pour créer des vues personnalisées.
Nous allons présenter ici brièvement la mise en œuvre de ces techniques. Puis nous les détaillerons davantage tout au long de cette partie.

Utilisation de composants prédéfinis

La première chose que nous allons apprendre dans cette partie est l'utilisation des composants prédéfinis ainsi que leur

disposition à l'écran. C'est l'objet des deux prochains chapitres de ce cours.
Grâce à cette technique, vous pourrez réaliser des interfaces graphiques telles que celle de la calculatrice présentée plus haut.
Pour rappel, voici l'interface graphique en question :



Une calculatrice intégrée.

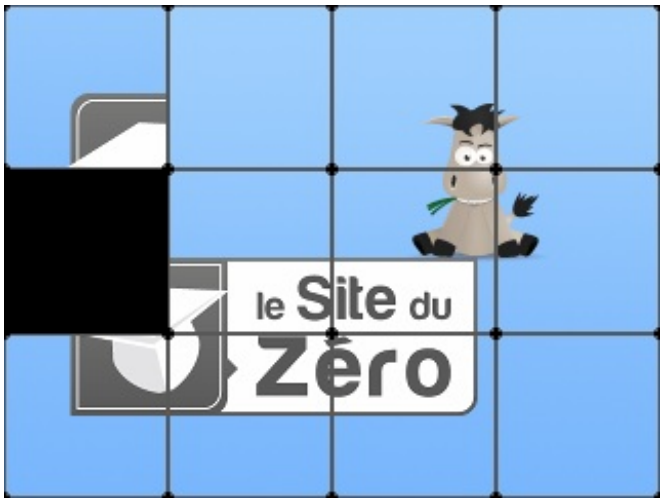
Sachez qu'une application n'est pas forcément composée d'une unique interface. Il est tout à fait possible de travailler avec plusieurs écrans que l'on fait défiler les uns après les autres. Cependant, *un seul* écran peut être affiché à la fois, les autres se contentent d'être ouverts en arrière-plan.

Pour concevoir ce type d'interfaces nous devrons utiliser des conteneurs, que nous pouvons voir comme des gestionnaires de positionnement. Pour cela nous utiliserons donc notre classe principale `MainScreen`, puis nous insèrerons des conteneurs qui contiendront eux-mêmes des composants.
Nous aurons tout le temps pour bien comprendre, mais saisissez qu'il s'agit uniquement d'éléments imbriqués les uns dans les autres. Il n'y aura donc aucune difficulté dans le code source.

Les vues personnalisées

Enfin pour finir nous allons brièvement présenter la conception de vues personnalisées. Ceci devrait ravir les futurs programmeurs de jeux vidéo !
Dans ce cas nous n'utiliserons pas de composants mais nous allons tracer différents éléments à l'écran. Nous utiliserons également la classe `MainScreen` qui est indispensable. Cependant nous allons ici nous servir de la méthode `paint()` de cette classe. Grâce à cette méthode nous pourrons utiliser la classe `Graphics` qui regorge de méthodes permettant de tracer des lignes, des cercles, des rectangles, etc, mais également afficher différentes images pour recréer tout ce qui vous passe par la tête.

À l'aide de cette technique, nous pourrons en fin de partie réaliser un jeu de taquin que voici :



Un jeu de taquin réalisé en fin de partie !

Je pense que certains sont déjà impatients de voir tout ça. Je vous invite donc sans plus attendre à passer au chapitre suivant !

- Tout objet graphique hérite de la classe `Field`.
- Les composants de base sont alors des champs de texte, de saisie de texte ou bien des boutons.
- La classe `Manager` sert à créer des conteneurs permettant de gérer la disposition des éléments internes.
- Les éléments de type `Screen` sont les conteneurs globaux qui englobent l'ensemble d'un « écran ».
- Une vue peut donc être créée en disposant divers composants prédéfinis.
- Une autre manière de concevoir une vue est de tracer différents éléments à l'écran pour créer une vue personnalisée.

Les composants de base

Dans ce chapitre nous allons présenter les composants les plus simples, mais surtout les plus utilisés, à savoir :

- les champs de texte
- les boutons
- les images.

En combinant ces différents éléments, vous pourrez ainsi créer des vues relativement complexes en les agençant correctement grâce aux conteneurs que nous verrons dans le prochain chapitre.
Ce chapitre est *long*, mais il n'y a aucune difficulté si vous prenez bien le temps de tout lire.

Les champs de texte

Nous allons dans ce chapitre présenter différents composants permettant de mettre en place une interface graphique. Chaque composant est associé à une classe avec différentes méthodes pour le manipuler. Dans ce cours nous survolerons brièvement les méthodes disponibles, c'est pourquoi je vous invite fortement à visiter la documentation officielle pour vous perfectionner. Notamment vous pouvez visiter la rubrique `Documentation` pour plus de renseignements ou encore la rubrique `API reference` pour visualiser l'ensemble des classes disponibles.

Introduction

Pour démarrer ce chapitre, nous allons reprendre notre classe `MyScreen` écrite pour notre premier projet `Hello World`. Si vous vous souvenez bien, nous avions utilisé la classe `RichTextField` pour pouvoir afficher un message à l'écran. Il s'agit du premier composant que nous étudierons dans ce chapitre.
Pour rappel, voici le code nous avions pour notre classe `MyScreen` :

Code : Java

```
package mypackage;

import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.MainScreen;

public final class MyScreen extends MainScreen
{
    public MyScreen()
    {
        setTitle("Site du Zéro");
        add(new RichTextField("Salut les zéros!"));
    }
}
```

Vous aurez certainement remarqué la présence de la méthode `add()` que nous avions introduit dans la première partie sans explications. Cette méthode de la classe `MainScreen` est une méthode commune à tous les conteneurs, qui permet d'ajouter un élément à ce conteneur. Pour l'utiliser il suffit de lui indiquer en paramètre l'instance du composant à ajouter. Nous reparlerons plus amplement des conteneurs dans le prochain chapitre.

Pour l'instant nous nous contenterons encore d'écrire notre code à l'intérieur du constructeur de la classe `MyScreen`.
Sans plus attendre, je vous propose de faire un vaste tour des différents composants disponibles pour la création d'interfaces graphiques. Commençons tout de suite avec `RichTextField` !

Afficher du texte

Pour afficher du texte à l'écran, nous disposons des deux classes `RichTextField` et `LabelField`. Ces deux composants sont très similaires, et il est tout à fait possible d'utiliser l'un ou l'autre. Peut-être la classe `LabelField` est plutôt utilisée en tant qu'étiquette pour d'autres contrôles, alors que l'utilisation d'un `RichTextField` permet plus de paramétrages. Ceci dit, dans la plupart des cas les deux feront tout à fait l'affaire.

RichTextField

Dans notre projet `Hello World`, nous avions créé une instance de la classe `RichTextField` directement à l'intérieur de la méthode `add()`. Toutefois il est possible de créer une instance de cette classe en la stockant dans une variable. C'est en effet la solution que je vous recommande dans la majorité des cas.

Nous pourrions alors réécrire le code précédent de la manière suivante :

Code : Java

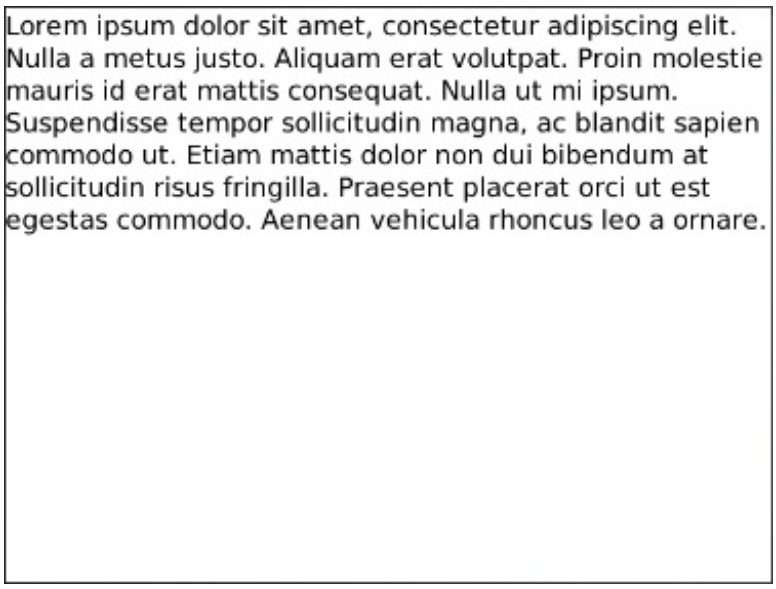
```
RichTextField monTexte = new RichTextField("Salut les zéros!");
add(monTexte);
```

Les champs de texte de type `RichTextField` sont multilignes, vous pouvez donc insérer un long texte qui s'étalera sur plusieurs lignes si besoin. Nous allons essayer ceci en utilisant la méthode `setText()` qui permet de modifier le texte de notre `RichTextField`.
Pour cela écrivez le code suivant :

Code : Java

```
monTexte.setText("Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Nulla a metus justo. Aliquam erat volutpat. Proin molestie
mauris id erat mattis consequat. Nulla ut mi ipsum. Suspendisse
tempor sollicitudin magna, ac blandit sapien commodo ut. Etiam
mattis dolor non dui bibendum at sollicitudin risus fringilla.
Praesent placerat orci ut est egestas commodo. Aenean vehicula
rhoncus leo a ornare.");
```

Vous obtiendriez alors un champ de texte sur plusieurs lignes comme présenté sur l'écran ci-dessous :



La classe `RichTextField` possède de nombreuses autres méthodes telles que `getText()` pour récupérer son texte ou encore la méthode `setFont` héritée de `TextField` qui permet de changer la police de celui-ci. Je vous laisserai le soin de les découvrir par vous-mêmes, en revanche nous parlerons des polices de caractères juste après la classe `LabelField`.

LabelField

Comme dit précédemment, la classe `LabelField` s'utilise principalement de la même manière que `RichTextField`. Nous pourrons donc déclarer une instance de cette classe comme ceci :

Code : Java

```
LabelField monTexte = new LabelField("Votre texte ici.");
add(monTexte);
```

Notez également qu'il n'est pas nécessaire de renseigner le texte à afficher dès la déclaration, et il en est de même pour la classe `RichTextField`. Si vous le désirez vous pouvez procéder de la manière suivante :

Code : Java

```
LabelField monTexte = new LabelField();
monTexte.setText("Votre texte ici.");
add(monTexte);
```

N'oubliez pas d'importer les différentes classes que vous utilisez. Pour éviter de les importer une par une, il est possible d'importer tout le package en insérant la ligne suivante :

Code : Java

```
import net.rim.device.api.ui.component.*;
```

Utilisation des polices

L'utilisation des polices de caractères est toujours un peu délicate, c'est pourquoi nous allons présenter ici la manière de faire. Pour appliquer une police et un style à un champ de texte nous devrons utiliser les deux classes `Font` et `FontFamily`. Une fois ces classes instanciées, il est alors possible de faire passer la police de type `Font` en paramètre de la méthode `setFont()` d'un champ de texte. Voici un exemple d'utilisation de ces classes :

Code : Java

```
FontFamily typeCaracteres = FontFamily.forName("Comic Sans MS");
Font maPolice = typeCaracteres.getFont(Font.BOLD, 26);
monTexte.setFont(maPolice);
```

Cependant si vous utilisez ce code tel quel, vous aurez des problèmes de compilation. En effet, ce code pourrait générer une exception. Pour gérer ces exceptions, il faudra donc utiliser le bloc `try{....} catch{...}` du langage Java. Pour rappel, ce bloc d'instructions permet de gérer des morceaux de code pouvant générer des exceptions ; par exemple une division par zéro. Dans notre cas, voici comment l'utiliser :

Code : Java

```
try
{
    FontFamily typeCaracteres = FontFamily.forName("Comic Sans MS");
    Font maPolice = typeCaracteres.getFont(Font.BOLD, 26);
    monTexte.setFont(maPolice);
}
catch (ClassNotFoundException e)
{
    System.out.println(e.getMessage());
}
```

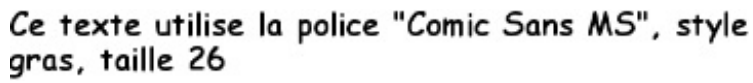
Enfin pour que personne ne soit perdu, je vous propose un code reprenant la classe `MySreen` intégralement. Ce code affichera donc un champ de texte de type `RichTextField` utilisant la police « Comic Sans MS » en gras et de taille 26.
Voici donc la classe `MySreen` écrite au complet :

Code : Java

```
package mypackage;

import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.Font;
import net.rim.device.api.ui.FontFamily;
```

```
public final class MyScreen extends MainScreen
{
    public MyScreen()
    {
     RichTextField monTexte = new RichTextField();
     try
     {
      FontFamily typeCaracteres = FontFamily.forName("Comic Sans
MS");
      Font maPolice = typeCaracteres.getFont(Font.BOLD, 26);
      monTexte.setFont(maPolice);
     }
     catch (ClassNotFoundException e)
     {
      System.out.println(e.getMessage());
     }
     monTexte.setText("Ce texte utilise la police \"Comic Sans MS\",
style gras, taille 26");
     add(monTexte);
      }
}
```

Je rappelle également que pour insérer des guillemets dans votre texte, vous devez insérer un antislash « \ » avant le caractère.
Lancez votre projet et vous devriez voir apparaître ceci à l'écran :



Les autres champs de texte

Nous allons maintenant voir différents champs de texte spécifiques qui permettent notamment à l'utilisateur de saisir du texte sous différentes formes depuis son clavier. Dans ce cours nous verrons ceux qui s'avèrent les plus utiles et performants.

EditField

Pour la saisie de texte, nous possédons diverses classes telles que `TextField`, `BasicEditField`, `EditField` et `AutoTextEditField`. Nous verrons uniquement les champs de type `EditField` qui sont les plus utilisés. Néanmoins l'utilisation d'un champ de saisie de texte de type `BasicEditField` est quasiment identique à celle que nous allons voir.
Je vous propose donc de découvrir le constructeur de ce nouveau type de champ de texte :

Code : Java

```
EditField monTexte = new EditField("Etiquette: ", "");
```

Vous remarquerez l'apparition d'une seconde chaîne de caractères dans le constructeur. En effet contrairement aux champs de texte `LabelField` ou `RichTextField`, les champs de saisie de texte possèdent deux champs distincts :

- un **label** ou **étiquette** : il s'agit de l'étiquette ou du nom associé à la saisie
- la **saisie** : il s'agit du texte renseigné par l'utilisateur à l'aide du clavier. Celui-ci peut être vide au départ mais peut également posséder une valeur initiale.

Vous trouverez donc des méthodes différentes pour chacun des champs. Par exemple la méthode `setLabel()` permettra de modifier l'étiquette alors que les méthodes `setText()` et `getText()` permettent de traiter la saisie.

Après quelques frappes au clavier, vous pourriez avoir ceci :

Les saisies de texte peuvent avoir des finalités très différentes et leur gestion peut entraîner facilement des erreurs ou des choses inattendues. C'est pourquoi la saisie de texte peut être facilitée à l'aide d'un filtre.

Par exemple, nous pourrions forcer l'utilisateur à renseigner uniquement des nombres entiers :

Code : Java

```
EditField monTexte = new EditField("Etiquette: ", "", 10,
EditField.FILTER_INTEGER);
```

Ou encore, nous pourrions imposer un texte en majuscules :

Code : Java

```
EditField monTexte = new EditField("Etiquette: ", "", 10,
EditField.FILTER_UPPERCASE);
```

Il existe de nombreux filtres mais voici les principaux et certainement les plus utilisés : `FILTER_DEFAULT`, `FILTER_INTEGER`, `FILTER_LOWERCASE`, `FILTER_NUMERIC`, `FILTER_PHONE`, `FILTER_UPPERCASE` et `FILTER_URL`.

Enfin vous pouvez utiliser les champs de type `AutoTextEditField` plus « intelligents ». Par exemple ils permettent d'insérer un point, un espace et une majuscule par double pression de la barre d'espace. Leur utilisation est très similaire à celle des `EditField`.

DateField

Vous pourriez également avoir besoin d'afficher la date, et heureusement il existe un champ spécial nommé `DateField`. Grâce à lui vous pouvez afficher date et heure selon le format désiré.
Voici comment créer un composant de type `DateField` :

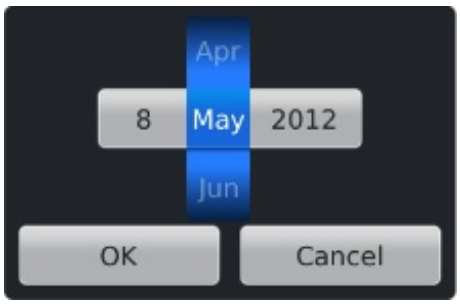
Code : Java

```
DateField monTexte = new DateField("Date: ",
System.currentTimeMillis(), DateField.DATE_TIME);
```

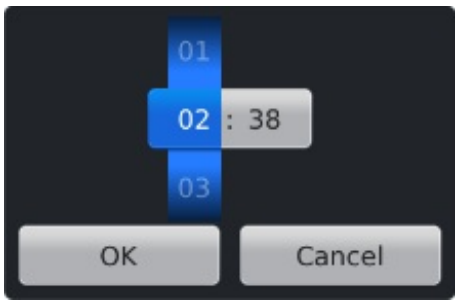
La méthode `currentTimeMillis()` permet de récupérer la date et l'heure exacte actuelle. Celle-ci renvoie le nombre de millisecondes écoulées depuis un point de départ : le 1er Janvier 1970 à 00:00:00 GMT. Ainsi cette valeur en millisecondes nous donne en même temps l'heure exacte ainsi que la date.

Pour en revenir à notre champ de texte, voici ce que donne un `DateField` à l'écran :

Ces champs sont également éditables, ainsi si vous cliquez sur une partie de la date vous verrez apparaître un nouveau contrôle que voici :



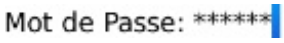
Bien évidemment l'heure est éditable au même titre que la date. Si vous cliquez sur l'heure, vous verrez également apparaître ceci :



Le format d'heure en millisecondes est spécial et dur à utiliser. D'autre part pour ceux qui penseraient que la milliseconde n'est pas très adaptée, sachez qu'avec une variable de type `long` nous pouvons comptabiliser plusieurs millions d'années en millisecondes !

PasswordEditField

Pour finir avec les champs de texte, nous allons maintenant voir la classe `PasswordEditField`.
Comme son nom l'indique, ce type de champs de texte est réservé aux mots de passe ou aux saisies cachées. C'est-à-dire que lors de la saisie, chaque caractère est remplacé par un astérisque « * » sur l'écran, comme ceci :



Mise à part l'affichage qui est différent, ces champs de texte sont identiques aux `EditField` vus précédemment. Je vous montre tout de même le constructeur pour éviter d'en perdre quelques-uns :

Code : Java

```
PasswordEditField monTexte = new PasswordEditField("Mot de Passe: ",
"");
```

Ainsi nous retrouverons donc les méthodes `setLabel()`, `setText()` et `getText()` pour gérer notre étiquette et notre texte éditable.

Les boutons

Nous allons à présent examiner les différents types de boutons proposés pour les interfaces. Nous y retrouvons tous les grands classiques, à savoir boutons simples, cases à cocher, boutons radios et d'autres. Dans ce chapitre nous ne présenterons que la mise en page de ces éléments. C'est-à-dire que nous ne verrons pas comment gérer une action sur un bouton ; ceci sera traité dans le chapitre sur les **évènements**.
Sans plus attendre, découvrons ces composants graphiques.

Les boutons classiques

ButtonField

Pour démarrer, voyons les boutons simples de type `ButtonField`.
Ce sont les boutons les plus utilisés, ils permettent de réaliser une action lorsqu'on clique dessus. Voilà leur apparence dans les interfaces graphiques BlackBerry :

Pour créer des boutons, la démarche est la même que pour les champs de texte. Nous allons donc instancier un élément de type `ButtonField` en appelant son constructeur. Le plus simple est de renseigner le texte qui s'affichera sur le bouton. Voici donc la manière de faire :

Code : Java

```
ButtonField monBouton = new ButtonField("Button");
```

```
add(monBouton);
```

Il est possible de modifier le texte ou étiquette ou encore label du bouton en utilisant la méthode `setLabel()`. Pour ceux qui auraient déjà pensé mettre une icône à la place du texte ou encore combiner icône et texte, nous verrons comment faire à la fin du chapitre. Pour réaliser cela nous devrons utiliser la méthode `setImage()` qui prend en paramètre un objet de type `Image` que nous introduirons un peu plus loin.

CheckboxField

Les composants de type `CheckboxField` sont des cases à cocher. Ces composants sont très utilisés, notamment pour activer ou non certains paramètres dans des menus d'options quelconques ou bien pour des acceptations de licences et de conditions. Ces composants possèdent deux états : actif par la valeur **`true`**, ou inactif par un **`false`**.
Voici comment utiliser un composant de la classe `CheckboxField` :

Code : Java

```
CheckboxField monBouton = new CheckboxField("Checkbox",true);
add(monBouton);
```

Voici à quoi ressemblent ces cases à cocher :

Les méthodes `setLabel()` et `getLabel()` nous permettent de saisir ou de récupérer l'étiquette de la case à cocher, de la même manière que pour un `ButtonField`. Enfin vous pouvez gérer l'état de la case à cocher soit en récupérant sa valeur par la méthode `getChecked()`, soit imposer un état par `setChecked()`.

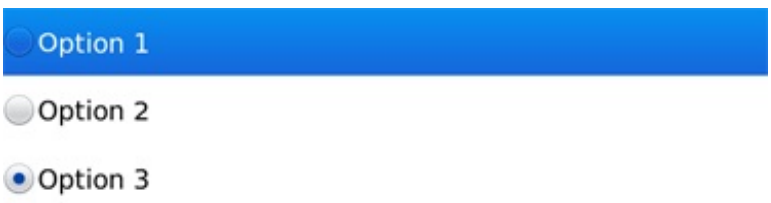
RadioButtonField

Les boutons `RadioButtonField` sont des boutons radios. Ils sont proches des cases à cocher sauf qu'un seul bouton ne peut être actif à la fois dans un même groupe. Je m'explique : pour créer ce genre de boutons, nous devons au préalable définir un groupe `RadioButtonGroup`, puis nous ajouterons alors des boutons à ce groupe.
Pour mieux comprendre nous allons prendre un exemple dont voici le code :

Code : Java

```
RadioButtonGroup monGroupe = new RadioButtonGroup();
add(new RadioButtonField("Option 1",monGroupe,true));
add(new RadioButtonField("Option 2",monGroupe,false));
add(new RadioButtonField("Option 3",monGroupe,true));
```

Nous avons ici déclaré trois boutons de type `RadioButtonField` que nous avons ajouté au même groupe. Je vous propose alors de tester ce code, vous devriez alors voir apparaître ceci sur votre écran :



Remarquez alors que dans notre code nous avons défini plusieurs boutons à l'état actif. Or seul le dernier l'est vraiment. Ainsi si vous activez un bouton, tous les autres seront alors désactivés, c'est la particularité de ce type de boutons.

Bien évidemment en temps normal vous aurez besoin de récupérer l'état d'un bouton avec la méthode `isSelected()`, c'est pourquoi vous devrez déclarer une variable pour chacun de vos boutons :

Code : Java

```
RadioButtonGroup monGroupe = new RadioButtonGroup();
RadioButtonField monBouton = new RadioButtonField("Option
1",monGroupe,true);
add(monBouton);
```

Les boutons déroulants

Nous avons déjà vu les cases à cocher et les boutons radios qui servent à faire des choix parmi plusieurs options. Ces boutons sont très utiles mais possèdent l'inconvénient de prendre de la place surtout lorsqu'il y a beaucoup de choix. Ceci est d'autant plus vrai que les écrans des terminaux mobiles sont petits. C'est pourquoi il existe une alternative : les *listes déroulantes*.

ObjectChoiceField

La première liste déroulante est de type `ObjectChoiceField` qui est propice à l'affichage de texte. Pour faire cela nous devons au préalable définir l'ensemble de nos différents choix à l'intérieur d'un tableau. La liste sera alors créée à partir des différents éléments de votre tableau, c'est aussi simple que cela.
Voici comment procéder :

Code : Java

```
String mesChoix[] = {"Option 1","Option 2","Option 3","Option 4"};
ObjectChoiceField maListe = new ObjectChoiceField("Label
:",mesChoix,2);
add(maListe);
```

Vous remarquerez que les listes déroulantes possèdent également une étiquette à renseigner dans le constructeur. Enfin le dernier paramètre correspond tout simplement à l'élément sélectionné initialement à l'apparition de l'écran. Voici à quoi ressemble le composant à l'écran :

Notez que les éléments de la liste possèdent les mêmes indices que le tableau. Ainsi l'élément d'indice 2 est en fait le troisième élément.

Si vous faites dérouler la liste vous verrez alors apparaitre l'ensemble des choix disponibles :

Les méthodes à retenir pour cette classe sont `setChoices()`, `getSelectedIndex()` et `getChoice()`. La première sert à définir ou redéfinir la liste de choix, la deuxième à récupérer l'indice de l'élément sélectionné et la dernière à récupérer la valeur de l'élément dont l'indice est renseigné en paramètre.
Nous avons ici utilisé des éléments de type `String` comme vous le ferez souvent, mais sachez qu'il est possible d'y insérer toutes sortes d'objets.

NumericChoiceField

Il existe une classe spécifique pour des listes déroulantes à valeurs numériques qui est `NumericChoiceField`. Pour définir

la liste voici les paramètres à renseigner *dans l'ordre* :

- premier élément
- dernier élément
- valeur d'incrémentation
- indice de l'élément initialement sélectionné

Ce qui nous donne par exemple :

Code : Java

```
NumericChoiceField maListe = new
NumericChoiceField("Label",1,12,1,4);
add(maListe);
```

Vous obtiendrez alors une liste très similaire à la précédente :

Puis en cliquant :

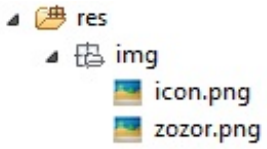
Vous pourrez également récupérer l'élément sélectionné à l'aide de la méthode `getSelectedValue()`.

Les images

Préparation des images

Nous allons maintenant nous attaquer aux images que vous attendiez certainement avec impatience. Nous verrons comment charger une image à partir de son nom, puis nous apprendrons à les afficher.
Avant de continuer, nous avons besoin d'une image. Je vous propose donc cette image de Zozor au format PNG, directement tirée de la page d'accueil du Site du Zéro.

Pour pouvoir charger une image, celle-ci doit se trouver dans le dossier du projet. Pour cela copier l'image dans `res/img` au même endroit que l'icône de l'application, puis rafraichissez le projet dans Eclipse pour voir apparaître votre image :



Nous voilà fin prêts pour démarrer !

Chargement des images

Pour afficher des images, celles-ci doivent être stockées à l'intérieur de variables. Nous en avons trois types qui sont : `Bitmap`, `EncodedImage`, `Image`.

Bitmap

Le chargement d'une image se fait ici à l'aide de la méthode `getBitmapResource()` de cette classe, en renseignant le nom de l'image à charger.
Voici comment charger une image dans une variable de type `Bitmap` :

Code : Java

```
Bitmap monImage = Bitmap.getBitmapResource("zozor.png");
```

EncodedImage

La classe `EncodedImage` permet également de charger une image, et c'est la classe que je vous *recommande* d'utiliser. Le chargement d'une image se fait de manière similaire que précédemment :

Code : Java

```
EncodedImage monImage =
EncodedImage.getEncodedImageResource("zozor.png");
```

Les images de type `EncodedImage` ont l'avantage d'être facilement manipulables, notamment si vous souhaitez redimensionner votre image. Pour cela utilisez la méthode `scaleImage32()` :

Code : Java

```
int taille = (int) Math.ceil(10000/echelle);
EncodedImage monImage =
EncodedImage.getEncodedImageResource("zozor.png").scaleImage32(Fixed32.tenThouToF
Fixed32.tenThouToFP(taille));
```

Attention, les valeurs rentrées pour la mise à l'échelle sont un peu particulières. Il s'agit en réalité de renseigner la valeur `10000/échelle`, c'est pourquoi il est préférable de faire le calcul avant.

Image

Enfin pour finir, la classe `Image` ne permet pas de charger directement une image. Cependant le format `Image` est parfois utilisé à la place du format `EncodedImage` pour afficher l'image. Cependant il est très facile de passer de l'un à l'autre grâce à la méthode `createImage()` de la classe `ImageFactory`. Cette méthode nous renvoie alors une image de type `Image`, utilisez donc l'expression :

Code : Java

```
ImageFactory.createImage(EncodedImage.getEncodedImageResource("zozor.png"))
```

Affichage d'images

Maintenant que les images sont chargées, nous allons pouvoir les afficher à l'écran. Nous verrons donc comment afficher directement l'image, puis nous verrons comment l'insérer en tant qu'icône dans un bouton de type `ButtonField`.

BitmapField

Le composant graphique `BitmapField` permet d'afficher une image à l'écran à partir d'une variable `Bitmap` ou `EncodedImage`. Pour un Bitmap, la manipulation est simple puisqu'il suffit de mettre l'image en question en paramètre du constructeur comme ceci :

Code : Java

```
Bitmap monImage = Bitmap.getBitmapResource("zozor.png");
BitmapField monBitmap = new BitmapField(monImage);
add(monBitmap);
```

En revanche aucun constructeur ne prend en paramètre des images de type `EncodedImage`, c'est pourquoi nous devrons procéder autrement. Nous utiliserons alors la méthode `setImage()` après avoir déclaré notre `BitmapField` :

Code : Java

```
EncodedImage monImage =
EncodedImage.getEncodedImageResource("zozor.png");
BitmapField monBitmap = new BitmapField();
monBitmap.setImage(monImage);
add(monBitmap);
```

Dans les deux cas vous verrez apparaître Zozor dans l'angle supérieur gauche de votre écran, comme ceci :

Retour sur les ButtonField

Rappelez-vous je vous avais dit qu'il était possible de mettre une icône avec ou à la place d'un texte sur un bouton de type `ButtonField`. Maintenant que nous savons charger une image, nous allons pouvoir le faire. Pour cela nous nous servirons de la méthode `setImage()` de la classe `ButtonField`, qui prend en paramètre une image de type `Image`.
Pour faire cela, il suffit simplement de réaliser les différentes opérations dans l'ordre :

Code : Java

```
ButtonField monBouton = new ButtonField("Zozor");
monBouton.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResource
add(monBouton);
```

Vous verrez alors l'image de Zozor insérée dans le bouton à côté du texte :

Il est possible de choisir de placer le texte à gauche ou à droite de l'image grâce aux méthodes `setLabelLeft()` et `setLabelRight()`. Enfin si vous ne désirez pas de texte mais uniquement l'icône, il suffit de n'en renseigner aucun dans le constructeur de la classe `ButtonField`.

- Tout composant graphique est représenté par une classe qui hérite de `Field`.
- Pour afficher du texte, il existe les classes `RichTextField` et `LabelField`.
- Des champs de saisie sont disponibles grâce aux classes `EditField`, `DateField` et `PasswordEditField`.
- Les boutons de base sont de type `ButtonField`.
- Des boutons évolués avec plusieurs choix sont définissables à l'aide des classes `CheckboxField`, `RadioButtonField`, `ObjectChoiceField` et `NumericChoiceField`.
- Les images sont chargées à travers les classes `Bitmap`, `EncodedImage` et `Image`.
- Pour afficher une image à l'écran, il faut utiliser le composant `BitmapField`.

Un peu d'ordre avec les conteneurs

Maintenant que nous avons appris à créer différents composants, nous allons maintenant apprendre à les disposer suivant nos envies grâce aux **conteneurs** !
Nous introduirons les différents types de conteneurs disponibles et nous verrons donc comment les organiser avec d'autres conteneurs et composants. Ainsi vous commencer à réaliser de vrais interfaces graphiques, correspondant à vos souhaits.
Enfin, nous réaliserons un album photo dans deux versions, que nous finaliserons dans le prochain chapitre !

Le principe des conteneurs

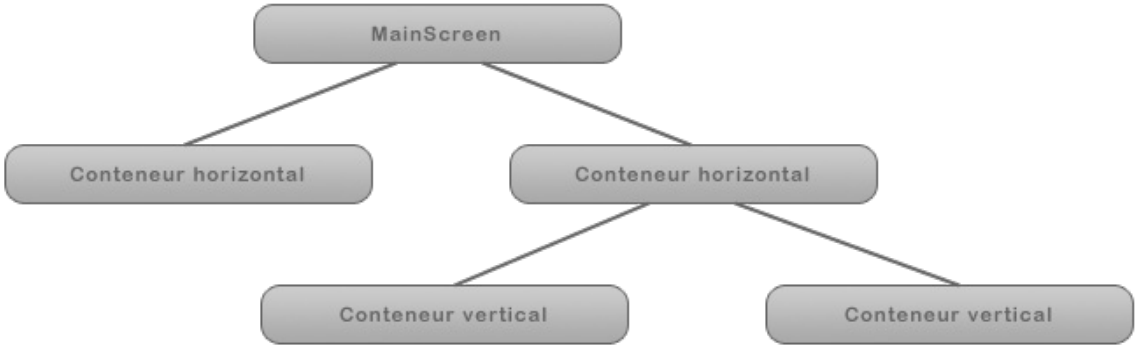
Introduction

Nous avions vu que la classe `Mainscreen` était un conteneur, qui est de type vertical. En effet, essayez d'y mettre plusieurs composants, vous vous apercevrez que ceux-ci s'insèrent verticalement. C'est pourquoi il existe différents types de conteneurs qui permettent d'aligner les éléments autrement.
Ainsi en combinant divers conteneurs, nous pourrons positionner les composants suivant nos envies. Voici par exemple une mise en page de conteneurs de types vertical et horizontal :

Chaque conteneur peut accueillir à la fois d'autres conteneurs ou bien des composants tels que ceux vus dans le chapitre précédent. Pour insérer un nouvel objet à un conteneur, nous devons utiliser une méthode déjà bien connue : `add()`.
Bien entendu il existe d'autres manières d'aligner les composants que verticalement et horizontalement. Nous détaillerons tous les types de conteneurs qui vous permettront de réaliser les mises en pages que vous souhaitez.

Le principe de hiérarchie

Pour bien comprendre la philosophie des conteneurs, il est nécessaire de bien saisir la manière dont ceux-ci sont *hiérarchisés*.
Pour cela, reprenons la mise en page précédente sous la forme d'un arbre, pour bien comprendre comment les différents conteneurs sont liés entre eux :



Voici comment analyser cet arbre :

- Les deux conteneurs horizontaux sont placés dans le `MainScreen` de type vertical. Ainsi ces deux conteneurs seront donc l'un en dessous de l'autre.
- Les deux conteneurs verticaux en bas de la hiérarchie sont à l'intérieur d'un conteneur horizontal. Ceux-ci seront donc côte à côte dans le sens horizontal.

Enfin, il faut noter que les conteneurs se remplissent suivant l'ordre dans lequel sont ajoutés les objets. C'est un point important lors de l'organisation de votre mise à page à l'aide de la méthode `add()`.

Voyons maintenant les différents types de conteneurs proposés !

Les différents conteneurs

Dans cette sous-partie, nous allons travailler sur les différents conteneurs proposés. Pour comprendre leur fonctionnement nous devrons utiliser plusieurs éléments à afficher. C'est pourquoi je vous propose de nous servir de ces images de Zozor :

Nous utiliserons donc ici trois composants de type `BitmapField` contenant chacun une des trois images. Maintenant que vous savez utiliser cet élément, nous ne réécrirons pas à chaque fois le code servant à charger les images.
Toutefois, je le vous donne quand même une fois pour que nous partions sur de bonnes bases :

Code : Java

```
BitmapField zozor1 = new BitmapField();
BitmapField zozor2 = new BitmapField();
BitmapField zozor3 = new BitmapField();
zozor1.setImage(EncodedImage.getEncodedImageResource("Zozor1.png"));
zozor2.setImage(EncodedImage.getEncodedImageResource("Zozor2.png"));
zozor3.setImage(EncodedImage.getEncodedImageResource("Zozor3.png"));
```

Les conteneurs

VerticalFieldManager

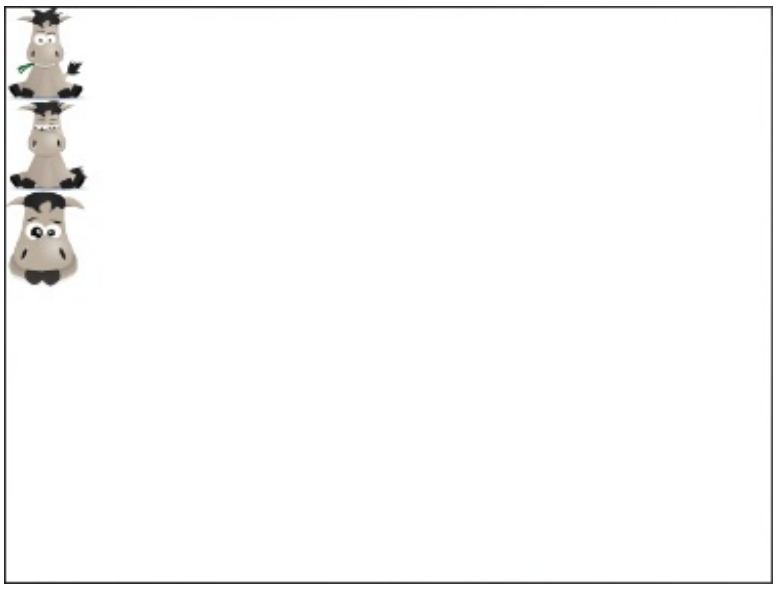
Les conteneurs de type `VerticalFieldManager` servent à ajouter les éléments verticalement. Pour l'utiliser nous devrons donc l'ajouter au `MainScreen`, lui-même de type vertical. Puis nous lui insèrerons alors les éléments les uns après les autres *dans l'ordre d'affichage*.
Sans plus attendre, voici comment l'utiliser :

Code : Java

```
VerticalFieldManager monManager = new VerticalFieldManager();
```

```
        add(monManager);
        monManager.add(zozor1);
        monManager.add(zozor2);
        monManager.add(zozor3);
```

Sans grande surprise, vous verrez les trois images tracées les unes en dessous des autres :



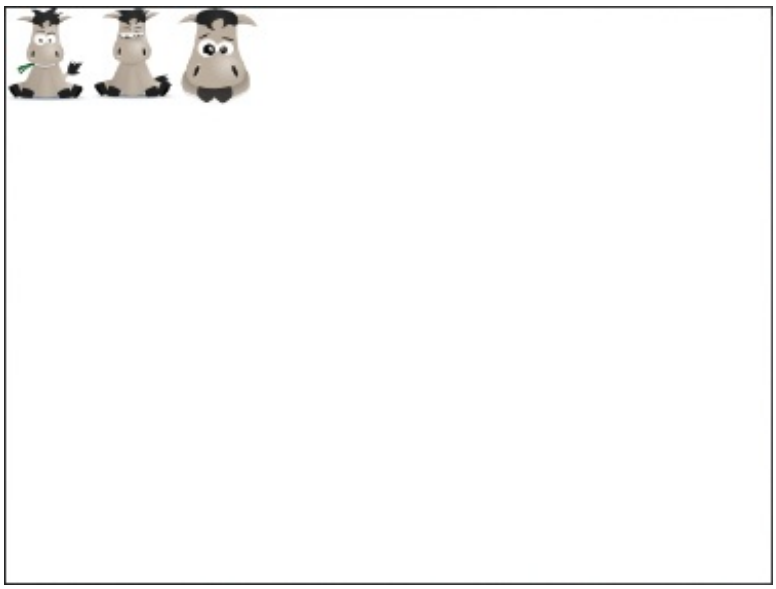
HorizontalFieldManager

Comme son nom l'indique, ce conteneur permet d'aligner les éléments horizontalement. L'utilisation de ce type de conteneur est identique que précédemment.
Voici donc le code correspondant :

Code : Java

```
        HorizontalFieldManager monManager = new HorizontalFieldManager();
        add(monManager);
        monManager.add(zozor1);
        monManager.add(zozor2);
        monManager.add(zozor3);
```

Là encore pas de surprise, les composants sont affichés horizontalement :



FlowFieldManager

Les conteneurs de type `FlowFieldManager` sont un mixe des deux précédents : les éléments sont rangés horizontalement puis verticalement. C'est-à-dire que les composants vont s'ajouter horizontalement jusqu'au bord de l'écran, puis sur une nouvelle ligne, etc.
Encore une fois, seul le nom du conteneur change à l'intérieur du code :

Code : Java

```
FlowFieldManager monManager = new FlowFieldManager();
add(monManager);
monManager.add(zozor1);
monManager.add(zozor2);
monManager.add(zozor3);
```

Ce type de conteneur n'est pas très recommandé dans la majorité des cas. En effet, il est très difficile de maîtriser la mise en page qui peut varier énormément d'un écran à un autre.

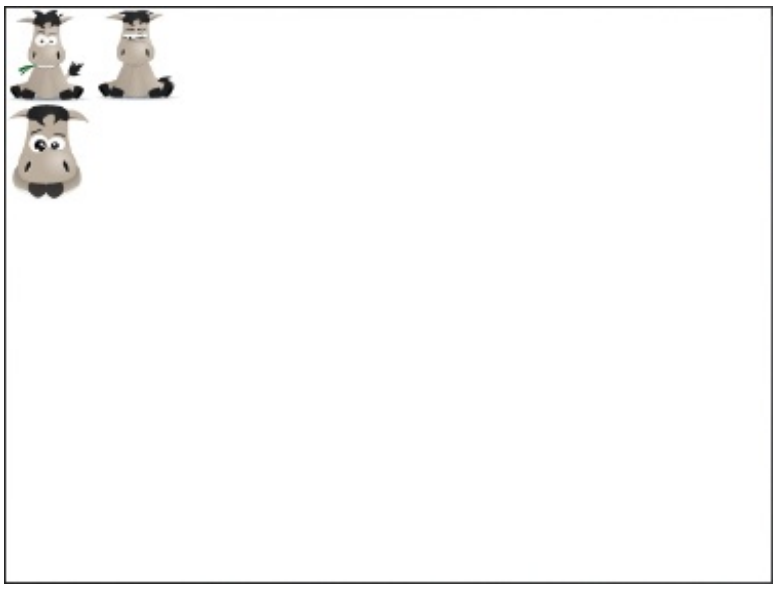
GridFieldManager

Les `GridFieldManager` sont des conteneurs sous forme de grilles, organisées en lignes et colonnes. Vous définissez le nombre de lignes et colonnes dans le constructeur, puis vous remplissez la grille par lignes.
Voici par exemple, une grille de dimension 2×2 contenant nos trois images :

Code : Java

```
GridFieldManager monManager = new
GridFieldManager(2,2,GridFieldManager.FIELD_LEFT);
add(monManager);
monManager.add(zozor1);
monManager.add(zozor2);
monManager.add(zozor3);
```

Pour l'instant ne vous souciez pas du `GridFieldManager.FIELD_FIELD_LEFT`, nous reviendrons dessus plus tard.
Étant donné que la grille possède uniquement deux colonnes, la troisième image sera placée dans la première colonne de la deuxième ligne :



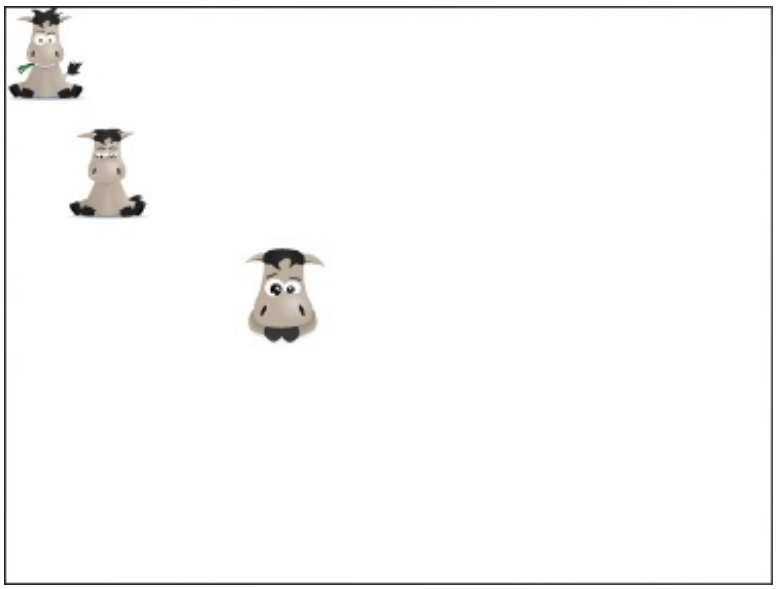
AbsoluteFieldManager

Pour finir nous allons voir les conteneurs de type `AbsoluteFieldManager`. Ceux-ci permettent de placer les éléments suivant leur position absolue par rapport à l'origine qui est l'*angle supérieur gauche*. Cette distance est exprimée en pixels suivant les axes *X horizontal* et *Y vertical* .
Voici nos trois images placées en différents points de l'écran :

Code : Java

```
AbsoluteFieldManager monManager = new AbsoluteFieldManager() ;
add(monManager);
monManager.add(zozor1,0,0);
monManager.add(zozor2,50,100);
monManager.add(zozor3,200,200);
```

Ce type de conteneur permet de placer les éléments plus « aléatoirement » ou disons plutôt de manière moins géométrique :



Attention, suivant la résolution de l'écran la même distance en pixel ne produira pas le même effet à l'écran. Il est donc nécessaire de prendre en compte un facteur d'échelle suivant la résolution de l'écran : nous reparlerons de ceci plus tard !

Quelques propriétés

L'arrière-plan

Il est probable que la couleur blanche en arrière-plan ne vous convienne pas. C'est pourquoi nous allons voir comment changer la couleur de fond d'un conteneur. Pour cela nous devrons charger une couleur à l'aide de la classe `BackgroundFactory` puis la stocker dans une variable de type `Background`.
Voici comment procéder pour le `MainScreen` :

Code : Java

```
VerticalFieldManager monManager =
(VerticalFieldManager)getMainManager();
Background maCouleur =
BackgroundFactory.createSolidBackground(Color.GRAY);
monManager.setBackground(maCouleur);
```

Vous remarquerez qu'il est nécessaire de récupérer le conteneur `MainScreen` avant de lui affecter sa couleur de fond.
Voici le résultat :

Pour tout autre conteneur, procédez comme suit :

Code : Java

```
HorizontalFieldManager monManager = new HorizontalFieldManager();
Background maCouleur =
BackgroundFactory.createSolidBackground(Color.RED);
monManager.setBackground(maCouleur);
```

La classe `Color` est une classe servant uniquement à utiliser des couleurs prédéfinies. Celle-ci ne possède aucune méthode propre mais uniquement de nombreuses couleurs définies en propriétés que je vous invite à regarder ici.

L'alignement

Les conteneurs et les composants possèdent des propriétés qui servent à les aligner. En voici une liste non exhaustive de ces propriétés :

- `Field.FIELD_LEFT` : aligné à gauche
- `Field.FIELD_HCENTER` : centré horizontalement
- `Field.FIELD_RIGHT` : aligné à droite
- `Field.FIELD_TOP` : aligné en haut
- `Field.FIELD_VCENTER` : centré verticalement
- `Field.FIELD_BOTTOM` : aligné en bas.

Ces propriétés sont renseignées dans le constructeur de l'élément lui-même. Voici un exemple reprenant nos trois images de Zozor :

Code : Java

```
BitmapField zozor1 = new
BitmapField(Bitmap.getBitmapResource("Zozor1.png"),BitmapField.FIELD_RIGHT);
BitmapField zozor2 = new
BitmapField(Bitmap.getBitmapResource("Zozor2.png"),BitmapField.FIELD_HCENTER);
BitmapField zozor3 = new
BitmapField(Bitmap.getBitmapResource("Zozor3.png"),BitmapField.FIELD_LEFT);
```

À l'intérieur d'un conteneur vertical tel que le `MainScreen`, les alignements précédents nous donnent alors un écran dans ce style :

Notez qu'il n'est pas possible d'aligner horizontalement des composants à l'intérieur d'un conteneur horizontal et inversement.

Un album photo

Après ces deux longs chapitres, nous allons pouvoir renter dans le concret. Pour cela, je vous propose de réaliser un petit album photo simplifié !

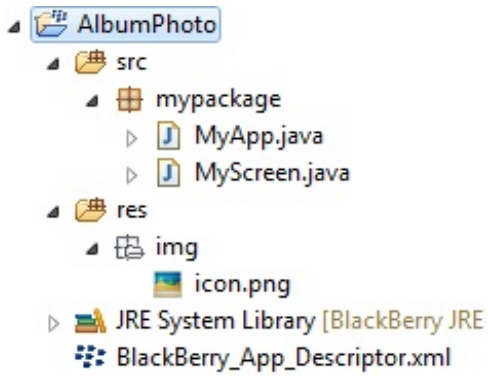
Dès à présent nous allons réaliser la mise en page de celui-ci, puis nous finirons l'application dans le chapitre suivant qui porte sur la gestion des **évènements**. Pour cet album photo nous concevrons deux versions ; la première comportant des boutons « Suivante » et « Précédente » pour faire défiler les photos, et la seconde sera composée de miniatures des photos pour accéder directement à la photo désirée.
Pour vous mettre l'eau à la bouche, voici un aperçu des deux versions de l'album photo :

Pour réaliser cet album photo, j'ai utilisé des photos gratuites et libres disponibles sur le site **stock**vault.net. Vous pouvez bien entendu utiliser vos propres photos, cependant pour simplifier le code il est préférable que celles-ci aient toutes les mêmes dimensions. Autrement voici les photos que j'ai sélectionné :

Sans plus attendre, lançons-nous dans la conception de cet album photo !

Version avec boutons

En premier nous devons créer un nouveau projet que j'ai nommé `AlbumPhoto`, qui est donc identique au `HelloWorld` de départ :



Ensuite insérez-y les différentes photos de votre album dans le répertoire `res/img`. De mon côté j'y ai également mis deux icônes pour les boutons « Suivante » et « Précédente ».

Un ButtonField personnalisé

Tout d'abord nous allons créer un composant personnalisé à partir d'un `ButtonField`. Ce que nous voulons ici c'est redéfinir la largeur des boutons pour que ceux-ci prennent la totalité de la largeur de l'écran. Pour faire cela nous devrons réécrire la méthode `getPreferredWidth()`, qui permet également de récupérer la largeur d'un composant.
Je vous propose donc de créer une nouvelle classe héritant de `ButtonField` comme ceci :

Code : Java

```
package mypackage;

import net.rim.device.api.system.Display;
```

```
import net.rim.device.api.ui.component.ButtonField;

public class Bouton extends ButtonField{

    public Bouton(String nom){
        super(nom);
    }

    public int getPreferredWidth() {
        return (Display.getWidth()/2 - 35);
    }

}
```

Pour récupérer la hauteur et la largeur de l'écran, utilisez les expressions `Display.getHeight()` et `Display.getWidth()`. Ces méthodes renvoient ces données en pixels, ce qui vous permet d'adapter votre mise en page en fonction des différents écrans de BlackBerry.

La mise en page

Nous allons maintenant nous intéresser à la classe `MyScreen` qui hérite de `MainScreen`.
Tout d'abord, insérez ces attributs qui nous servirons pour le chapitre suivant :

Code : Java

```
int photoActuelle;
EncodedImage[] mesImages;
BitmapField maPhoto;
int echelle;
```

Nous avons donc un tableau d'`EncodedImage` nommé `mesImages` qui regroupe l'ensemble de nos photos. Ensuite nous avons un entier `photoActuelle` qui contiendra l'indice dans le tableau de la photo affichée en grand ainsi que le `BitmapField maPhoto` qui la contiendra. Enfin l'entier `echelle` retiendra l'échelle de l'image que nous pourrons réutiliser pour la mise à jour de l'affichage.

Maintenant concentrons-nous sur ce que nous allons écrire à l'intérieur du constructeur `MyScreen`.
Commençons par définir la couleur d'arrière-plan en noir comme nous avons appris à le faire :

Code : Java

```
VerticalFieldManager monEcran =
(VerticalFieldManager)getMainManager();
Background maCouleur =
BackgroundFactory.createSolidBackground(Color.BLACK);
monEcran.setBackground(maCouleur);
```

Ensuite nous allons initialiser nos attributs sauf l'entier `echelle` qui nécessite au préalable d'avoir effectué quelques tracés. Pour cela nous allons prendre la première image du tableau comme image de départ. Ensuite nous allons charger nos `EncodedImage` dans le tableau à l'aide d'une boucle, puis nous initialiserons notre `BitmapField` avec la première photo. Voici l'ensemble de ces opérations :

Code : Java

```
photoActuelle = 0;
mesImages = new EncodedImage[6];
for(int i=0;i<6;i++){
    mesImages[i] =
EncodedImage.getEncodedImageResource("photo"+(i+1)+".jpeg");
}
maPhoto = new BitmapField(new Bitmap(0,0),FIELD_HCENTER);
```

```
maPhoto.setImage(mesImages[photoActuelle]);
```

Occupons-nous à présent de nos boutons personnalisés créés dans la classe `Bouton`. Ayant déjà redéfini la largeur du composant dans la classe `Bouton`, nous pouvons maintenant utiliser ce composant de la même façon qu'un `ButtonField`. Nous allons donc déclarer nos deux boutons « Précédente » et « Suivante », puis nous y ajouterons une icône grâce à la méthode `setImage()`. Pour finir nous déplacerons le texte à gauche de l'icône pour le deuxième bouton, voici ce que cela donne :

Code : Java

```
Bouton boutonPrec = new Bouton(" Précédente");
boutonPrec.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResource
Bouton boutonSuiv = new Bouton(" Suivante ");
boutonSuiv.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResource
boutonSuiv.setLabelLeft();
```

Comme vous le voyez, les boutons sont alignés horizontalement, nous aurons donc besoin d'un conteneur de type `HorizontalFieldManager`. Nous pourrons alors ajouter nos différents éléments aux divers conteneurs dans un ordre précis :

Code : Java

```
HorizontalFieldManager mesBoutons = new
HorizontalFieldManager(FIELD_HCENTER);
add(maPhoto);
add(new SeparatorField());
add(mesBoutons);
mesBoutons.add(boutonPrec);
mesBoutons.add(boutonSuiv);
```

L'élément `SeparatorField` n'a pas été présenté, il sert à séparer facilement des composants de manière plus propre et aérée. N'hésitez pas à les utiliser également dans vos mises en page.

Il ne nous reste plus qu'à redimensionner l'image centrale en fonction de la hauteur des boutons. Pour cela, nous initialiserons notre variable `echelle` dans un premier temps, puis nous mettrons à jour le `BitmapField` :

Code : Java

```
echelle = (int) Math.ceil(10000*mesImages[0].getHeight()/(Display.getHeight() -
mesBoutons.getPreferredHeight() - 35));
maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(echel
Fixed32.tenThouToFP(echelle)));
```

Étant donné que ce code sera indispensable pour la suite, je vous ai réécrit l'ensemble de la classe `MyScreen` :

Code : Java

```
package mypackage;

import net.rim.device.api.math.Fixed32;
import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
import net.rim.device.api.system.EncodedImage;
import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.TouchEvent;
```

```
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;
import net.rim.device.api.ui.image.ImageFactory;

public final class MyScreen extends MainScreen
{
    int photoActuelle;
    EncodedImage[] mesImages;
    BitmapField maPhoto;
    int echelle;

    public MyScreen()
    {
        //Couleur d'arrière-plan
        VerticalFieldManager monEcran = (VerticalFieldManager)getMainManager();
        Background maCouleur = BackgroundFactory.createSolidBackground(Color.BLA
        monEcran.setBackground(maCouleur);

        //Initialisation des attributs
        photoActuelle = 0;
        mesImages = new EncodedImage[6];
     for(int i=0;i<6;i++){
         mesImages[i] = EncodedImage.getEncodedImageResource("photo"+(i+1)+".jpe
     }
        maPhoto = new BitmapField(new Bitmap(0,0),FIELD_HCENTER);
        maPhoto.setImage(mesImages[photoActuelle]);

        //Préparation des boutons
     Bouton boutonPrec = new Bouton(" Précédente");

boutonPrec.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResourc
        Bouton boutonSuiv = new Bouton(" Suivante ");

boutonSuiv.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResourc
        boutonSuiv.setLabelLeft();

        //Mise en page des conteneurs
     HorizontalFieldManager mesBoutons = new HorizontalFieldManager(FIELD_HCENTE
        add(maPhoto);
        add(new SeparatorField());
        add(mesBoutons);
        mesBoutons.add(boutonPrec);
        mesBoutons.add(boutonSuiv);

        //Redimensionnement de l'image en cours d'affichage
     echelle = (int) Math.ceil(10000*mesImages[0].getHeight()/(Display.getHeight
mesBoutons.getPreferredHeight() - 35));
        maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouTo
Fixed32.tenThouToFP(echelle)));
    }
}
```

Version avec miniatures

Pour concevoir l'album photo avec miniatures, nous allons reprendre le code précédent sur un certain nombre de points.
Pour commencer, ici nous n'aurons pas besoin de la classe `Bouton`. Dans la classe `MyScreen`, nous garderons exactement les mêmes attributs. Ensuite nous allons dans un premier temps garder les deux premiers blocs d'instructions du constructeur, à savoir :

Code : Java

```
    //Couleur d'arrière-plan
```

```
    VerticalFieldManager monEcran =
    (VerticalFieldManager)getMainManager();
    Background maCouleur =
    BackgroundFactory.createSolidBackground(Color.BLACK);
    monEcran.setBackground(maCouleur);

    //Initialisation des attributs
    photoActuelle = 0;
    mesImages = new EncodedImage[6];
    for(int i=0;i<6;i++){
        mesImages[i] =
    EncodedImage.getEncodedImageResource("photo"+(i+1)+".jpeg");
    }
    maPhoto = new BitmapField(new Bitmap(0,0),FIELD_HCENTER);
    maPhoto.setImage(mesImages[photoActuelle]);
```

N'ayant plus de boutons ici, nous reprendrons directement la mise en page des conteneurs. Mettons en place le conteneur horizontal que j'ai renommé `mesMiniatures`, ainsi que le `BitmapField` et le `SeparatorField`, ce qui nous donne :

Code : Java

```
    //Mise en page des conteneurs
    HorizontalFieldManager mesMiniatures = new
    HorizontalFieldManager(FIELD_HCENTER);
    add(maPhoto);
    add(new SeparatorField());
    add(mesMiniatures);
```

Nous arrivons maintenant à la partie intéressante de cette version : l'*affichage des miniatures* !
Pour réaliser cela nous allons utiliser une boucle. À l'intérieur, nous créerons pour chaque miniature un `BitmapField` à partir des `EncodedImage` de notre tableau `mesImages`. En considérant que ces photos ont toutes les mêmes dimensions, nous les remettrons à l'échelle de manière à ce que chacune d'elles fasse $\frac{1}{6}$ de la largeur de l'écran. Puis nous les ajouterons une à une dans notre conteneur horizontal.
Voici le code qui résume tout ça :

Code : Java

```
//Affichage des vignettes
int taille = (int)
Math.ceil(10000*6*mesImages[0].getWidth()/Display.getWidth());
for(int i=0;i<6;i++){
    BitmapField maMiniature = new BitmapField(new
Bitmap(0,0),FIELD_HCENTER);

maMiniature.setImage(mesImages[i].scaleImage32(Fixed32.tenThouToFP(taille),
Fixed32.tenThouToFP(taille)));
    mesMiniatures.add(maMiniature);
}
```

Enfin, nous n'avons plus qu'à redimensionner l'image centrale en fonction de la hauteur des miniatures de la même manière que précédemment.

Pour récapituler tout cela, je vous propose la classe `MyScreen` dans son intégralité :

Code : Java

```
package mypackage;

import net.rim.device.api.math.Fixed32;
import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
```

```
import net.rim.device.api.system.EncodedImage;
import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.TouchEvent;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;
import net.rim.device.api.ui.image.ImageFactory;

public final class MyScreen extends MainScreen
{
    int photoActuelle;
    EncodedImage[] mesImages;
    BitmapField maPhoto;
    int echelle;

    public MyScreen()
    {
        //Couleur d'arrière-plan
        VerticalFieldManager monEcran = (VerticalFieldManager)getMainManager();
        Background maCouleur =
BackgroundFactory.createSolidBackground(Color.BLACK);
        monEcran.setBackground(maCouleur);

        //Initialisation des attributs
        photoActuelle = 0;
        mesImages = new EncodedImage[6];
        for(int i=0;i<6;i++){
            mesImages[i] =
EncodedImage.getEncodedImageResource("photo"+(i+1)+".jpeg");
        }
        maPhoto = new BitmapField(new Bitmap(0,0),FIELD_HCENTER);
        maPhoto.setImage(mesImages[photoActuelle]);

        //Mise en page des conteneurs
        HorizontalFieldManager mesMiniatures = new
HorizontalFieldManager(FIELD_HCENTER);
        add(maPhoto);
        add(new SeparatorField());
        add(mesMiniatures);

        //Affichage des vignettes
        int taille = (int)
Math.ceil(10000*6*mesImages[0].getWidth()/Display.getWidth());
        for(int i=0;i<6;i++){
            BitmapField maMiniature = new BitmapField(new
Bitmap(0,0),FIELD_HCENTER);

maMiniature.setImage(mesImages[i].scaleImage32(Fixed32.tenThouToFP(taille),
Fixed32.tenThouToFP(taille)));
            mesMiniatures.add(maMiniature);
        }

        //Redimensionnement de l'image en cours d'affichage
        echelle = (int)
Math.ceil(10000*mesImages[0].getHeight()/(Display.getHeight() -
mesMiniatures.getPreferredHeight() - 35));

maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(eche
Fixed32.tenThouToFP(echelle)));
    }
}
```

- Il existe des conteneurs de différents types, notamment horizontaux et verticaux.
- Les conteneurs et les composants s'ordonnent suivant une *hiérarchie*.

- L'ensemble des conteneurs sont définis dans le package `net.rim.device.api.ui.container`.
- Les deux plus courants des conteneurs sont `VerticalFieldManager` et `HorizontalFieldManager`.

La gestion des évènements

Il est probable que vous vous demandiez ce qu'est un **évènement**, je vais donc essayer de vous expliquer ce concept !
Lorsqu'on conçoit une application, il est intéressant d'avoir de l'interaction avec l'utilisateur. Par exemple dans notre album photo, nous voudrions que l'image centrale soit mise à jour si on clique sur un des boutons ou encore si on touche une miniature sur l'écran. Hé bien les évènements servent justement à déclencher une fonction lorsqu'une action s'est produite.
Au cours de ce chapitre, nous allons donc découvrir comment gérer différents types d'évènements et mettre cela en pratique dans notre album photo !

Les évènements de boutons

La théorie

Pour exécuter une fonction à l'appui d'un bouton, nous utilisons ce que l'on appelle un **écouteur**.
Comme son nom l'indique, il va « écouter » d'éventuels évènements associés à un bouton. Pour cela nous allons utiliser la classe `FieldChangeListener`, et plus particulièrement nous allons redéfinir la méthode `fieldChanged()`.
Ensuite il ne restera plus qu'à associer l'écouteur au composant grâce à la méthode `setChangeListener()` disponible pour tout type de bouton :

Code : Java

```
monBouton.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        // Instructions
    }
});
```

Notez qu'il est tout fait possible de créer une nouvelle classe dans le projet qui hérite de `FieldChangeListener` et d'y réécrire la méthode `fieldChanged()` à l'intérieur. Cette pratique peut être utile pour séparer et structurer votre projet, le rendant ainsi plus aéré.

La gestion des boutons de l'album photo

Pour mieux comprendre, je vous propose de paramétrer nos boutons « Précédente » et « Suivante » de notre album photo.
Pour changer l'image affichée en fonction des boutons, il suffit de mettre à jour la variable `photoActuelle`, puis le `BitmapField` correspondant. Par exemple pour le bouton « Précédente », nous allons décrémenter photoActuelle puis appeler la méthode `setImage()` de la même manière que dans le chapitre précédent.
Voici ce que donne la gestion des évènements des deux boutons de l'album photo :

Code : Java

```
// Ecouteur du bouton "Précédente"
boutonPrec.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        if(photoActuelle > 0){
            photoActuelle--;

maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(echel
Fixed32.tenThouToFP(echelle)));
        }
    }
});

// Ecouteur du bouton "Suivante"
boutonSuiv.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        if(photoActuelle < 5){
            photoActuelle++;

maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(echel
Fixed32.tenThouToFP(echelle)));
        }
    }
```

```
});
```

Également je vous redonne l'intégralité de la classe `MyScreen` pour que tout le monde puisse tester cet album photo :

Code : Java

```
package mypackage;

import net.rim.device.api.math.Fixed32;
import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
import net.rim.device.api.system.EncodedImage;
import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;
import net.rim.device.api.ui.image.ImageFactory;

public final class MyScreen extends MainScreen
{
    // Attributs
    int photoActuelle;
    EncodedImage[] mesImages;
    BitmapField maPhoto;
    int echelle;

    public MyScreen()
    {
        //Couleur d'arrière-plan
        VerticalFieldManager monEcran = (VerticalFieldManager)getMainManager();
        Background maCouleur = BackgroundFactory.createSolidBackground(Color.BLA
        monEcran.setBackground(maCouleur);

        //Initialisation des attributs
        photoActuelle = 0;
        mesImages = new EncodedImage[6];
        for(int i=0;i<6;i++){
            mesImages[i] = EncodedImage.getEncodedImageResource("photo"+(i+1)+".
        }
        maPhoto = new BitmapField(new Bitmap(0,0),FIELD_HCENTER);
        maPhoto.setImage(mesImages[photoActuelle]);

        //Préparation des boutons
        Bouton boutonPrec = new Bouton(" Précédente");

boutonPrec.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResourc
        Bouton boutonSuiv = new Bouton(" Suivante ");

boutonSuiv.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResourc
        boutonSuiv.setLabelLeft();

        //Mise en page des conteneurs
        HorizontalFieldManager mesBoutons = new HorizontalFieldManager(FIELD_HCE
        add(maPhoto);
        add(new SeparatorField());
        add(mesBoutons);
        mesBoutons.add(boutonPrec);
        mesBoutons.add(boutonSuiv);

        //Redimensionnement de l'image en cours d'affichage
        echelle = (int) Math.ceil(10000*mesImages[0].getHeight()/(Display.getHei
mesBoutons.getPreferredHeight() - 35));
        maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouTo
```

```
Fixed32.tenThouToFP(echelle)));

        // Ecouteur du bouton "Précédente"
        boutonPrec.setChangeListener(new FieldChangeListener() {
            public void fieldChanged(Field field, int context) {
                if(photoActuelle > 0){
                    photoActuelle--;

maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(echel
Fixed32.tenThouToFP(echelle)));
                }
            }
        });

        // Ecouteur du bouton "Suivante"
        boutonSuiv.setChangeListener(new FieldChangeListener() {
            public void fieldChanged(Field field, int context) {
                if(photoActuelle < 5){
                    photoActuelle++;

maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(echel
Fixed32.tenThouToFP(echelle)));
                }
            }
        });
    }
}
```

Pour l'instant nous n'avons parlé de la méthode `setChangeListener()` uniquement pour les boutons. Mais sachez que cette méthode appartient en réalité à la classe `Field`, elle est donc disponible pour tout composant notamment les champs de texte éditables et autres.

Les évènements « tactiles »

La théorie

Les écrans tactiles sont maintenant omniprésents sur les terminaux mobiles. Nous allons maintenant découvrir comment gérer les évènements générés par l'utilisateur sur l'écran.
Pour gérer ces évènements nous allons devoir redéfinir la méthode `touchEvent()` de notre `MainScreen`. Pour cela je vous propose de découvrir directement la méthode en question que nous analyserons juste après :

Code : Java

```
protected boolean touchEvent(TouchEvent message) {

    // Récupération des informations liées à l'évènement
    int eventCode = message.getEvent();
    int touchX =  message.getX(1);
    int touchY =  message.getY(1);

    // Gestion des différents évènements
    if(eventCode == TouchEvent.DOWN) {
        // Instructions
    }
    if(eventCode == TouchEvent.UP) {
        // Instructions
    }
    if(eventCode == TouchEvent.MOVE) {
        // Instructions
    }
    return true;
}
```

Dans un premier temps nous devons récupérer les informations liées à l'évènement. Pour cela nous utilisons la méthode

`getEvent()` de la classe `TouchEvent` pour obtenir le type de l'évènement. Également nous allons récupérer la position de l'évènement par les méthodes `getX()` et `getY()` comme présenté ci-dessus.
Une fois ces informations connues, il est alors possible d'exécuter des instructions suivant l'évènement apparu ou la position de celui-ci à l'écran.

La gestion des miniatures de l'album photo

Revenons-en à notre album photo !
Dans le cas des miniatures, la gestion des évènements est également assez simple. Ici, nous nous occuperons uniquement du cas où l'utilisateur pose son doigt sur l'écran soit l'évènement `TouchEvent.DOWN`. L'opération consiste alors dans un premier temps à vérifier si la zone de l'écran touchée correspond à celle des miniatures. Pour cela il nous suffit de tester si la variable `touchY` est supérieure à la hauteur de l'image `maPhoto.getPreferredHeight()`. Ensuite nous devons récupérer le numéro de l'image sélectionné en faisant un petit calcul. Enfin nous n'aurons plus qu'à mettre à jour le `BitmapField` comme nous l'avons toujours fait jusqu'à présent :

Code : Java

```
protected boolean touchEvent(TouchEvent message) {

    // Récupération des informations liées à l'évènement
    int eventCode = message.getEvent();
    int touchX =  message.getX(1);
    int touchY =  message.getY(1);

    // Gestion de l'évènement
    if(eventCode == TouchEvent.DOWN) {
        if(touchY > maPhoto.getPreferredHeight()){
            photoActuelle = (int) Math.ceil(6*touchX/Display.getWidth());

maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(eche]
Fixed32.tenThouToFP(echelle)));
        }
    }
    return true;
}
```

Pour ceux qui seraient intéressés, voici l'intégralité de la classe `MyScreen` :

Code : Java

```
package mypackage;

import net.rim.device.api.math.Fixed32;
import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
import net.rim.device.api.system.EncodedImage;
import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.TouchEvent;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;

public final class MyScreen extends MainScreen
{
    int photoActuelle;
    EncodedImage[] mesImages;
    BitmapField maPhoto;
    int echelle;

    public MyScreen()
    {
```

```
        //Couleur d'arrière-plan
        VerticalFieldManager monEcran = (VerticalFieldManager)getMainManager();
        Background maCouleur =
BackgroundFactory.createSolidBackground(Color.BLACK);
        monEcran.setBackground(maCouleur);

        //Initialisation des attributs
        photoActuelle = 0;
        mesImages = new EncodedImage[6];
        for(int i=0;i<6;i++){
            mesImages[i] =
EncodedImage.getEncodedImageResource("photo"+(i+1)+".jpeg");
        }
        maPhoto = new BitmapField(new Bitmap(0,0),FIELD_HCENTER);
        maPhoto.setImage(mesImages[photoActuelle]);

        //Mise en page des conteneurs
        HorizontalFieldManager mesMiniatures = new
HorizontalFieldManager(FIELD_HCENTER);
        add(maPhoto);
        add(new SeparatorField());
        add(mesMiniatures);

        //Affichage des vignettes
        int taille = (int)
Math.ceil(10000*6*mesImages[0].getWidth()/Display.getWidth());
        for(int i=0;i<6;i++){
            BitmapField maMiniature = new BitmapField(new
Bitmap(0,0),FIELD_HCENTER);

maMiniature.setImage(mesImages[i].scaleImage32(Fixed32.tenThouToFP(taille),
Fixed32.tenThouToFP(taille)));
            mesMiniatures.add(maMiniature);
        }

        //Redimensionnement de l'image en cours d'affichage
        echelle = (int)
Math.ceil(10000*mesImages[0].getHeight()/(Display.getHeight() -
mesMiniatures.getPreferredHeight()));

maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(echel
Fixed32.tenThouToFP(echelle)));

    }

    protected boolean touchEvent(TouchEvent message) {

        //Récupération des informations liées à l'évènement
        int eventCode = message.getEvent();
        int touchX =  message.getX(1);
        int touchY =  message.getY(1);

        // Gestion de l'évènement
        if(eventCode == TouchEvent.DOWN) {
            if(touchY > maPhoto.getPreferredHeight()){
                photoActuelle = (int) Math.ceil(6*touchX/Display.getWidth());

maPhoto.setImage(mesImages[photoActuelle].scaleImage32(Fixed32.tenThouToFP(echel
Fixed32.tenThouToFP(echelle)));
            }
        }
        return true;
    }

}
```

Je vous invite maintenant à tester cet album photo. Pour ceux qui voudraient aller plus loin, n'hésitez pas à ajouter de nouvelles

fonctionnalités ou de retravailler l'interface à votre guise. Voici le visuel final de cette application :

Un peu plus loin...

Pour terminer ce chapitre sur les évènements, nous allons introduire un dernier point qui peut vous être utile dans vos futures applications.
Pour cela, revenons sur les instructions de récupération de position de l'évènement. Rappelez-vous :

Code : Java

```
int touchX =  message.getX(1);
int touchY =  message.getY(1);
```

Je vous avais introduit ces méthodes `getX()` et `getY()` sans vous expliquer pourquoi nous passions en paramètre le nombre **1**.
En réalité ce chiffre correspond au numéro de l'évènement, je m'explique. Sur les écrans tactiles il est possible d'utiliser deux doigts en mode multipoints, et donc de gérer deux évènements simultanés à l'écran.
Ainsi, un deuxième évènement peut être traité en utilisant le code suivant :

Code : Java

```
    protected boolean touchEvent(TouchEvent message) {
        switch(message.getEvent()) {
        case TouchEvent.MOVE:
            if(message.getX(1) > 0 && message.getY(1) > 0) {
                // Instructions pour la première position
            }
            if(message.getX(2) > 0 && message.getY(2) > 0) {
                // Instructions pour la seconde position
            }
            return true;
        }
        return false;
    }
```

Comme vous avez dû le remarquer, un évènement est actif lorsque chacune de ses coordonnées est non nulle. Vous pouvez alors gérer chacun des évènements séparément pour ajouter toujours plus de fonctionnalités à votre application.

Pour tester des évènements multiples sur le simulateur BlackBerry, vous devrez utiliser le *Multitouch Mode* en allant dans la rubrique `Simulate` ou en appuyant sur Alt + M. Si vous utilisez ce type de codage, il est préférable que vous le testiez un terminal physique. Également, soyez vigilants avec ce type de gestion car l'utilisation de plusieurs évènements simultanément peut surcharger l'écran de votre appareil et ne laisser que très peu de place au visuel.

- Nous utilisons des **écouteurs** pour exécuter des fonctions à l'appui d'un bouton.
- Un écouteur se conçoit en utilisant la classe `FieldChangeListener` et en redéfinissant sa méthode `fieldChanged()`.
- Pour associer un écouteur à un composant, il faut utiliser la méthode `setChangeListener()`.
- Pour gérer des évènements « tactiles », il est nécessaire de redéfinir la méthode `touchEvent()` de notre `MainScreen`.
- La récupération des informations liées à un évènement « tactile » se fait via la classe `TouchEvent`.

Les vues personnalisées

Nous avons vu jusqu'à maintenant comment réaliser des interfaces graphiques à l'aide de composants prédéfinis. À présent nous allons découvrir comment créer une vue à partir de zéro, pour créer une vue personnalisée.
Ce type de vue est plus complexe à mettre en place, notamment en ce qui concerne la gestion de différentes tailles d'écrans. Néanmoins cela permet d'avoir beaucoup plus de contrôle sur l'affichage, et sert également à personnaliser plus efficacement vos différentes vues.

La structure de base

Nous allons ici voir le code minimal nécessaire pour créer des interfaces graphiques personnalisées. Pour cela nous allons créer une nouvelle classe nommée `MaClasseDessin` qui héritera de la classe `MainScreen`. Sans plus attendre, je vous laisse découvrir la structure de base que nous utiliserons pour notre nouvelle classe :

Code : Java - MaClasseDessin.java

```
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.TouchEvent;
import net.rim.device.api.ui.container.MainScreen;

public class MaClasseDessin extends MainScreen{

    //Attributs


    public MaClasseDessin(){
        //Initialisations des attributs
    }

    public void paint(Graphics g){
        super.paint(g);
        //Mise en place des éléments à l'écran
    }

    protected boolean touchEvent(TouchEvent message) {

        //Récupération des informations liées à l'évènement
        int eventCode = message.getEvent();
        int touchX =  message.getX(1);
        int touchY =  message.getY(1);

        if(eventCode == TouchEvent.DOWN) {
            //Instructions lorsque l'utilisateur pose son doigt sur
l'écran
            invalidate();
        }

        if(eventCode == TouchEvent.UP) {
            //Instructions lorsque l'utilisateur retire son doigt
de l'écran
            invalidate();
        }

        if(eventCode == TouchEvent.MOVE) {
            //Instructions lorsque l'utilisateur bouge son doigt de
l'écran
            invalidate();
        }
        return true;
    }

}
```

Pour ceux qui l'auraient déjà utilisée, vous constaterez que cette structure est très proche de celle utilisée en Java :

Code : Java

```
package monpackage;

import java.awt.*;
import javax.swing.*;

public class JPanelDessin extends JPanel {

    //Attributs


    public JPanelDessin() {
        super();
        //Initialisations des attributs
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        //Mise en place des éléments à l'écran
    }
}
```

Pour étudier la structure de cette classe, je vous propose d'y aller étape par étape.

Pour commencer, voyons les *attributs*. Les attributs correspondent aux variables que nous manipulerons à l'intérieur de nos méthodes. Nous pourrons y trouver des variables de tous types, cependant nous y trouverons en particulier tous les éléments ou paramètres permettant de tracer l'interface.
Par exemple nous pourrions stocker une image à tracer ainsi que ses coordonnées à l'écran :

Code : Java

```
Bitmap monImage;
int[] position;
```

N'oubliez pas d'importer les classes que vous utilisez :

Code : Java

```
import net.rim.device.api.system.Bitmap;
```

Qui dit attributs dit *initialisations*, et pour cela nous avons le constructeur. Lorsque vous créez une classe qui hérite d'une autre, n'oubliez pas d'appeler le constructeur de la classe parente : ici la classe `MainScreen`. Voici donc notre constructeur :

Code : Java

```
public MaClasseDessin(){
    super();
    monImage = Bitmap.getBitmapResource("image.png");
    position = new int[2];
    position[0] = 0;
    position[1] = 0;
}
```

La méthode suivante est la plus importante : il s'agit de la méthode `paint()`. C'est à l'intérieur de celle-ci que nous définirons comment tracer les différents éléments à l'écran. Lorsque nous voudrons rafraîchir l'écran, c'est cette méthode qui sera appelée. C'est à nous de faire en sorte que l'élément soit affiché selon nos désirs. Dans notre exemple nous allons tracer l'image à la position souhaitée, c'est-à-dire aux coordonnées définies dans le tableau `position`.
Pour tracer nos éléments nous utiliserons des méthodes de la classe `Graphics` :

Code : Java

```
    public void paint(Graphics g){
        super.paint(g);
        g.drawImage(position[0], position[1], 200, 200, monImage, 0, 0);
    }
```

Enfin pour que notre application est de l'utilité, il va falloir gérer les évènements. Pour cela, je vous propose de tracer l'image `monImage` à la position où l'utilisateur pose le doigt. Voici donc la méthode `touchEvent()` proposée :

Code : Java

```
    protected boolean touchEvent(TouchEvent message) {

        int eventCode = message.getEvent();
        int touchX =  message.getX(1);
        int touchY =  message.getY(1);

        if(eventCode == TouchEvent.DOWN) {
            position[0] = touchX;
            position[1] = touchY;
            invalidate();
        }
        return true;
    }
```

Dans cette méthode, on reçoit le paramètre de type `TouchEvent` grâce auquel nous allons pouvoir récupérer toutes les informations liées à l'évènement. Ainsi si l'utilisateur pose un doigt sur l'écran, alors nous mettons à jour la position de l'image. La méthode `invalidate()` est *très importante*. C'est elle qui va appeler la méthode `paint()` et permettre de retracer les éléments à l'écran.

Voilà, nous avons fait le tour de cette classe. Pour finir, je vous ai donné la classe entière :

Code : Java - MaClasseDessin.java

```
    import net.rim.device.api.ui.Graphics;
    import net.rim.device.api.ui.TouchEvent;
    import net.rim.device.api.ui.container.MainScreen;
    import net.rim.device.api.system.Bitmap;

    public class MaClasseDessin extends MainScreen{

        Bitmap monImage;
        int[] position;

        public MaClasseDessin(){
            super();
            monImage = Bitmap.getBitmapResource("image.png");
            position = new int[2];
            position[0] = 0;
            position[1] = 0;
        }

        public void paint(Graphics g){
            super.paint(g);
            g.drawImage(position[0], position[1], 200, 200, monImage, 0,
    0);
        }

        protected boolean touchEvent(TouchEvent message) {

            int eventCode = message.getEvent();
            int touchX =  message.getX(1);
            int touchY =  message.getY(1);

            if(eventCode == TouchEvent.DOWN) {
                position[0] = touchX;
```

```
                position[0] = touchX;
                position[1] = touchY;
                invalidate();
            }
            return true;
        }

    }
```

Insérer des contours

Nous allons à présent découvrir les différentes méthodes qui permettent de tracer différentes formes et contours. Toutes les dimensions sont ici exprimées en *pixels*.

Nous verrons ici quelques-unes de ses méthodes, cependant la classe `Graphics` regorge encore de plein de méthodes que je vous invite à découvrir sur cette page.

Les bases

drawPoint

Rien de plus classique qu'un point !
Pour tracer un point utilisez la méthode `drawPoint()` en spécifiant les coordonnées du point, comme ceci :

Code : Java

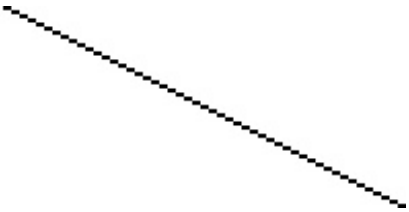
```
g.drawPoint(10, 10);
```

drawLine

Comme son nom l'indique, cette méthode sert à tracer des lignes. Pour l'utiliser, renseignez les coordonnées du point de départ puis celles du point d'arrivée.
Voici un exemple :

Code : Java

```
g.drawLine(0, 0, 100, 50);
```



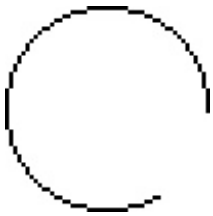
Les formes elliptiques

drawArc

La méthode `drawArc()` permet de tracer des arcs de cercle ou des cercles complets. Les différents paramètres dans l'ordre sont : les coordonnées x et y du centre du cercle, la largeur et la hauteur du cercle, et enfin l'angle de départ et celui de fin du tracé.
L'exemple ci-dessous trace un cercle presque complet entre 0° et 300° :

Code : Java

```
g.drawArc(100, 100, 50, 50, 0, 300);
```

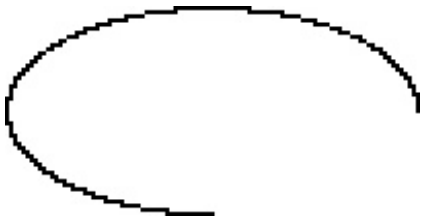


drawEllipse

Cette méthode trace également un arc mais de forme elliptique. Pour l'utiliser, spécifiez dans l'ordre : les coordonnées du centre, les coordonnées d'un point appartenant à l'arc, les coordonnées d'un second point appartenant à l'arc, puis les angles de départ et d'arrivée.

Code : Java

```
g.drawEllipse(100, 200, 50, 200, 100, 225, 0, 270);
```



Les rectangles

drawRect

Pour la méthode `drawRect()`, il n'y a rien de compliqué. Placez en paramètres les coordonnées de l'angle supérieur gauche ainsi que la largeur et la hauteur du rectangle, comme ceci :

Code : Java

```
g.drawRect(100, 10, 100, 50);
```

drawRoundRect

Cette méthode est identique à la précédente, simplement rajoutez la largeur et la hauteur de l'arrondi.

Code : Java

```
g.drawRoundRect(100, 10, 100, 50, 30, 30);
```

Quelques plus

drawText

Il est également possible d'écrire du texte grâce à la méthode `drawText()`. Renseignez votre texte à afficher ainsi que sa position.

Code : Java

```
g.drawText("Texte", 100, 25);
```

setColor

La méthode `setColor()` permet de choisir la couleur des contours mais *également* celle des remplissages. La couleur s'écrit sous la forme : 0xAARRVVBB, où AA est la valeur hexadécimale de la transparence et RR, VV, BB celles des couleurs rouge, vert et bleu.
Voici un exemple qui modifie la couleur des tracés en bleu :

Code : Java

```
g.setColor(0x000000FF);
g.drawText("Texte", 100, 25);
```

clear

Enfin, notez l'existence de la méthode `clear()` qui sert à effacer l'écran, c'est-à-dire le rafraîchir entièrement de la couleur d'arrière-plan.

Code : Java

```
g.clear();
```

Insérer des remplissages
Les essentiels

fillArc

Cette méthode fonctionne identiquement à `drawArc()`, cependant celle-ci remplit l'intérieur de l'arc de la couleur actuelle de dessin.

Code : Java

```
g.fillArc(50, 100, 50, 50, 0, 180);
```

fillEllipse

La méthode `fillEllipse()` fonctionne également comme `drawEllipse()`.

Code : Java

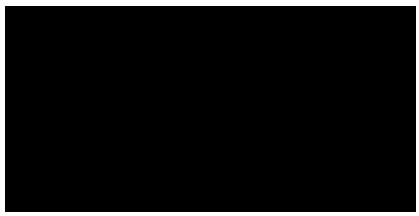
```
g.fillEllipse(300, 100, 280, 100, 280, 140, 0, 360);
```

fillRect

Pour créer des remplissages de rectangles, utilisez la méthode `fillRect()`.

Code : Java

```
g.fillRect(100, 10, 100, 50);
```

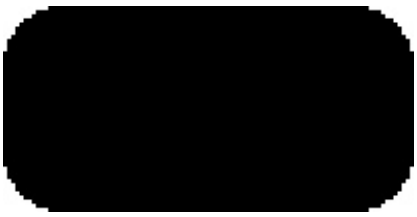


fillRoundRect

Enfin pour finir, voici la méthode `fillRoundRect()` :

Code : Java

```
g.fillRoundRect(100, 10, 100, 50, 30, 30);
```



Un peu de couleur

drawGradientFilledRect

La méthode `drawGradientFilledRect()` permet de créer un remplissage de couleur dégradée à l'intérieur d'un rectangle. Celle-ci s'utilise de la même manière que `fillRect()`, en précisant en plus les couleurs de départ et d'arrivée du dégradé.
Voici un exemple de dégradé du bleu au rouge :

Code : Java

```
g.drawGradientFilledRect(100, 10, 100, 50, 0x000000FF, 0x00FF0000);
```

drawGradientFilledRoundedRect

Cette méthode est similaire à la précédente, en y ajoutant les arrondis :

Code : Java

```
g.drawGradientFilledRoundedRect(100, 10, 100, 50, 0x00FFFF00,
0x00FF00FF, 30, 30);
```

setBackbroungColor

Pendant que nous sommes dans les couleurs, nous allons présenter la méthode `setBackbroungColor()` qui sert à choisir la couleur d'arrière-plan. Celle-ci s'utilise simplement en renseignant la couleur désirée.
Voici par exemple comment créer un fond noir :

Code : Java

```
g.setBackgroundColor(0x00000000);
```

Les images

Pour terminer ce chapitre, nous allons voir comment vous pouvez tracer des images grâce à la classe `Graphics`. Pour cela, vous disposez de deux méthodes qui dépendent du type d'image que vous avez chargé.

drawBitmap

Si vous avez chargé votre image avec la classe `Bitmap`, vous devrez alors utiliser la méthode `drawBitmap()`, comme ceci :

Code : Java

```
Bitmap monImage = Bitmap.getBitmapResource("zozor.png");
g.drawBitmap(100, 50, 200, 200, monImage, 0, 0);
```

Les deux premiers paramètres correspondent à la position de l'image. Nous reviendrons sur les autres paramètres dans très peu de temps.
Vous devriez donc voir apparaître Zozor à la position spécifiée :

drawImage

Si vous utilisez des images de type `EncodedImage`, vous devrez alors utiliser la méthode `drawImage()` pour afficher cette image à l'écran. Celle-ci est quasiment identique à la méthode précédente :

Code : Java

```
EncodedImage monImage =
EncodedImage.getEncodedImageResource("zozor.png");
g.drawImage(100, 50, 200, 200, monImage, 0, 0, 0);
```

Le rendu est donc le même :

Rogner l'image

Dans les deux méthodes, différents paramètres permettent de rogner l'image selon vos désirs. Ainsi dans la méthode `drawImage()`, les troisième et quatrième paramètres permettent de définir les dimensions de l'image. Si celles-ci sont inférieures à la taille de l'image, alors l'image est rognée. Il est également possible de décaler ce rognage grâce aux deux derniers paramètres. Pour mieux comprendre comment cela fonctionne, n'hésitez pas à tester différentes valeurs.
Voici un exemple que je vous propose :

Code : Java

```
g.drawImage(100, 30, 20, 50, monImage, 0, 20, 0);
```

Cet exemple va donc rogner l'image à gauche, à droite ainsi qu'en bas comme montré ci-dessous :

- Les vues personnalisées permettent d'avoir un meilleur contrôle et plus de possibilités sur l'affichage.
- Pour dessiner à l'écran, il est nécessaire de redéfinir la méthode `paint()` de votre `MainScreen`.
- Les différents tracés sont réalisés à l'aide des nombreuses méthodes de la classe `Graphics`.
- Pour ce type de vue, les évènements peuvent être gérés à l'aide de `touchEvent()`.
- Pour mettre à jour l'affichage, il faut faire appel à la méthode `invalidate()` qui appellera notamment la méthode `paint()` que vous avez redéfinie.

TP : un taquin

Il est temps pour vous de pratiquer un peu !
Nous allons donc dans ce TP réaliser un jeu de taquin depuis zéro. Pour la correction, nous procèderons étape par étape pour vous permettre de comprendre au mieux. L'objectif de ce chapitre n'est donc pas de vous faire voir de nouvelles notions, mais uniquement de vous faire pratiquer avec les connaissances que vous avez déjà. Ce sera donc l'occasion de faire un point sur ce que vous avez retenu jusqu'à présent.

Le cahier des charges

Qu'est-ce qu'un taquin ?

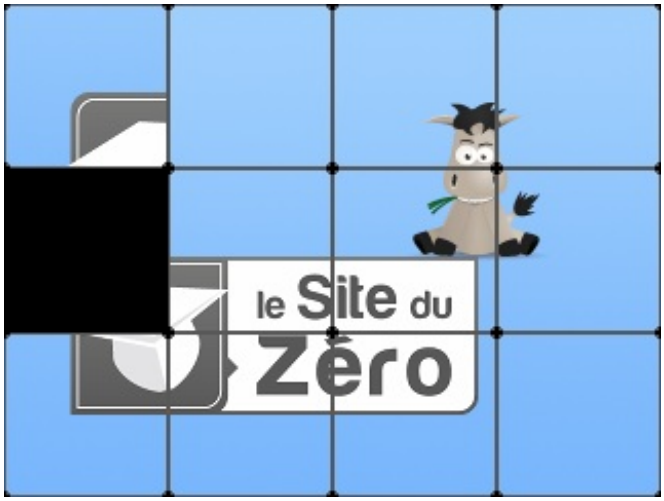
Le taquin est un jeu solitaire créé il y a environ 150 ans.
À l'origine, ce jeu était composé de 15 carreaux numérotés de 1 à 15 glissant dans un cadre prévu pour 16. La pièce manquante s'appelait le « blanc ». Il fallait alors déplacer une pièce adjacente au « blanc » pour réorganiser le jeu. Dans un taquin, le but du jeu consiste alors à remettre les carreaux dans l'ordre d'après une configuration initiale quelconque.
De nos jours, le taquin est souvent représenté sous la forme d'une image à reconstituer. On en retrouve de partout, notamment sur internet.

Spécifications du projet

Au cours de ce TP, nous allons réaliser un taquin aux couleurs du site. Vous pourrez utiliser vos propres images ou bien récupérer les miennes plus bas. Nous devrons donc créer une application qui sera soumise à certaines contraintes. Je vous propose de faire un rapide tour de ce que nous devrons prendre en compte lors de sa conception :

- au démarrage, le jeu doit être mélangé mais doit être *faisable*. Il ne faudra pas mélanger les vignettes n'importe comment.
- l'utilisateur doit pouvoir déplacer les cases, mais *uniquement* celles qui sont à côté du « blanc ».
- la partie est terminée une fois que toutes les vignettes sont en place, une petite boîte de dialogue avertira alors l'utilisateur.

Enfin, voici un aperçu de ce que nous pourrions avoir à l'écran lors du déroulement du jeu :



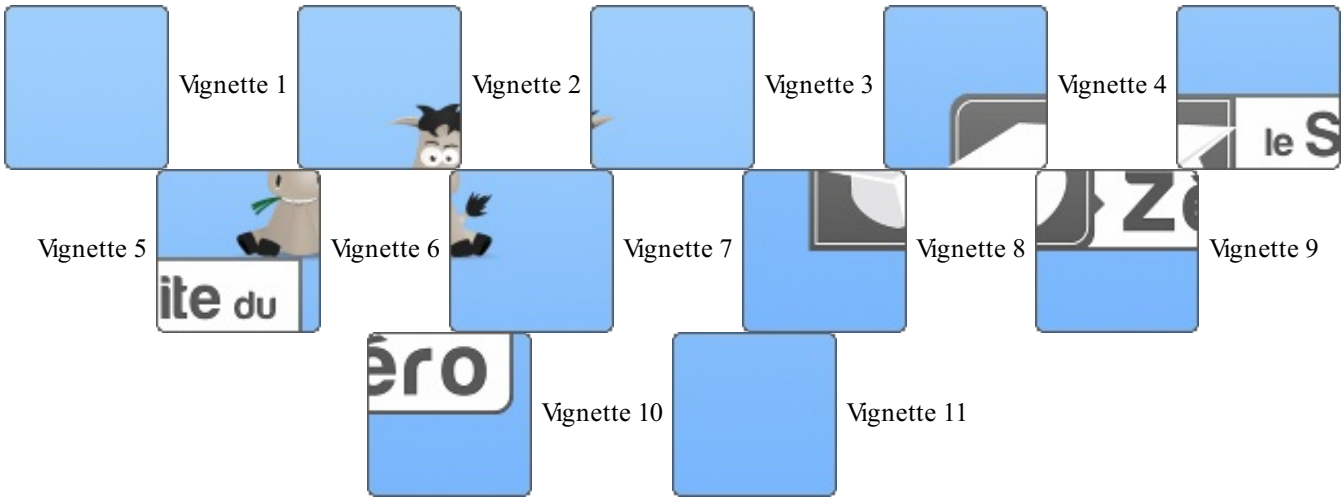
Étant donné que ce n'est pas évident, je me permets de préciser que le « blanc » correspond ici à la première case en haut à gauche.

Pourquoi avoir choisi ce jeu ?

Pour commencer le taquin est un jeu assez populaire, vous y avez probablement tous joué !
Ce jeu est intéressant, je vous assure que vous y jouerez un sacré moment une fois le TP terminé. Les règles sont simples, ce qui facilite la mise en place du cahier des charges. De plus, la logique simple du jeu facilitera la conception de l'application, bien qu'il nous faudra plusieurs classes pour parvenir au bout du TP.
D'autre part, je trouve qu'un jeu « manuel » tel que le taquin s'intègre très bien sur les plateformes tactiles. En effet, comme sur un vrai jeu nous déplacerons les cases à l'aide des doigts. Bien gérer ces déplacements rendra le jeu plus interactif et plus fluide, c'est ce qui fait la popularité de ce genre d'applications.

Avant de commencer

Ce jeu peut être codé de plusieurs manières différentes, c'est pourquoi je vous invite *fortement* à essayer de le réaliser par vous-même. Vous pourrez alors comparer votre façon d'appréhender le problème avec la mienne. Ma méthode n'est pas meilleure qu'une autre, il est possible que vous trouviez une méthode plus simple et demandant moins de lignes de code.
Quoi qu'il en soit, pour réaliser ce TP il se peut que vous soyez obligé d'y passer plusieurs jours. N'hésitez pas à prendre votre temps !
Voici les vignettes que j'ai utilisé pour réaliser mon taquin :



Allons-y !

La correction

Avant de passer à de plus amples explications, je vous ai ici mis l'ensemble du code source. Cela me permettra dans un premier temps, de vous présenter les différentes classes que j'ai utilisé. Pour réaliser cette application, je me suis servi de *cinq* classes dont la fonction vous est présentée juste après.
N'hésitez pas à tester cette application en copiant-collant le code source des différentes classes ci-dessous !

Le code que je vous propose pour chacune des classes ne possède aucun commentaire. En effet, les classes sont relativement conséquentes, c'est pourquoi je les ai allégé au maximum. De toute manière les explications seront données plus bas et rien ne vous empêche de commenter le code vous-mêmes.

La classe MonApp

La classe `MonApp` est comme depuis le début, la classe de départ de notre programme. Elle n'est pas différente des précédentes, c'est pourquoi nous n'y reviendrons pas dans les explications. Celle-ci se contente simplement de faire appel à la classe `MonDessin` décrite plus bas. Vous trouverez néanmoins le code source de cette classe ci-dessous :

Code : Java - MonApp.java

```
package sdz.taquin;

import net.rim.device.api.ui.UiApplication;

public class MonApp extends UiApplication
{
    public static void main(String[] args)
    {
        MonApp theApp = new MonApp();
        theApp.enterEventDispatcher();
    }

    public MonApp()
    {
        pushScreen(new MonDessin());
    }
}
```

La classe MonDessin

La classe `MonDessin` reprend la syntaxe de la classe `MaClasseDessin` décrite dans le chapitre sur les vues personnalisées. Nous utilisons dans cette classe tout ce qui avait été vu dans ce précédent chapitre. Nous la détaillerons plus bas, mais en attendant voici le code correspondant :

Code : Java - MonDessin.java

```
package sdz.taquin;

import net.rim.device.api.system.Display;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.TouchEvent;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.container.MainScreen;

public class MonDessin extends MainScreen{

    private int echelle;
    private Vignette[] mesVignettes;
    private Partie maPartie;
    private Mouvement monMouvement;


    public MonDessin(){
        super();
        echelle = Math.min(Display.getWidth()/320, Display.getHeight()/240);
        maPartie = new Partie();
        monMouvement = new Mouvement();
        mesVignettes = new Vignette[11];
        for(int i=0; i<4; i++){
            for(int j=0; j<3; j++){
                if(maPartie.getVignette(i,j)!=0){
                    mesVignettes[maPartie.getVignette(i,j)-1] = new
Vignette(maPartie.getVignette(i,j),echelle, i, j);
                }
            }
        }
    }

    public void paint(Graphics g){
        super.paint(g);
        g.setBackgroundColor(0x00000000);
        g.setColor(0x00FFFFFF);
        g.clear();
        for(int i=0; i<4; i++){
            for(int j=0; j<3; j++){
                if(maPartie.getVignette(i,j)!=0){
                    int x = mesVignettes[maPartie.getVignette(i,j)-1].getX();
                    int y = mesVignettes[maPartie.getVignette(i,j)-1].getY();
                    EncodedImage img = mesVignettes[maPartie.getVignette(i,j)-1
                    g.drawImage(x, y, 80*echelle, 80*echelle, img, 0, 0, 0);
                    g.drawText(""+maPartie.getVignette(i, j), x+10, y+10);
                }else{
                    g.drawText(""+maPartie.getVignette(i, j), i*80*echelle+10,
j*80*echelle+10);
                }
            }
        }
    }

    protected boolean touchEvent(TouchEvent message) {

        int eventCode = message.getEvent();
        int touchX =  message.getX(1);
        int touchY =  message.getY(1);
```

```
            if(eventCode == TouchEvent.DOWN) {
                int i = touchX/(80*echelle);
                int j = touchY/(80*echelle);
                monMouvement.setNumero(maPartie.getVignette(i,j));
                int[][] mouv = maPartie.estJouable(i,j, echelle);
                monMouvement.setMouvements(mouv);
                monMouvement.setAnterieur(touchX, touchY);
                invalidate();
            }

            if(eventCode == TouchEvent.UP) {
                if(monMouvement.getNumero()!=0 && monMouvement.estActif()==true){
                    int i =
    (mesVignettes[monMouvement.getNumero()-1].getX()+40*echelle)/(80*echelle);
                    int j =
    (mesVignettes[monMouvement.getNumero()-1].getY()+40*echelle)/(80*echelle);
                    monMouvement.setMouvements();
                    maPartie.placer(i, j, monMouvement.getNumero());
                    mesVignettes[monMouvement.getNumero()-1].setPosition(i*80*echel
    j*80*echelle);
                }
                invalidate();
                if(maPartie.estEnPlace()==true){
                    Dialog.alert("Bravo, vous avez réussi!");
                    System.exit(0);
                    return true;
                }
            }

            if(eventCode == TouchEvent.MOVE) {
                int mouvX = touchX - monMouvement.getAnterieur()[0];
                int mouvY = touchY - monMouvement.getAnterieur()[1];
                if(monMouvement.getNumero()!=0 && monMouvement.estActif()==true){

    if((mesVignettes[monMouvement.getNumero()-1].getX()+mouvX)<monMouvement.getMouv

    mesVignettes[monMouvement.getNumero()-1].setX(monMouvement.getMouvements()[0][0
                    }else
    if((mesVignettes[monMouvement.getNumero()-1].getX()+mouvX)>monMouvement.getMouv

    mesVignettes[monMouvement.getNumero()-1].setX(monMouvement.getMouvements()[0][1
                    }else{
                        mesVignettes[monMouvement.getNumero()-1].addX(mouvX);
                    }

    if((mesVignettes[monMouvement.getNumero()-1].getY()+mouvY)<monMouvement.getMouv

    mesVignettes[monMouvement.getNumero()-1].setY(monMouvement.getMouvements()[1][0
                    }else
    if((mesVignettes[monMouvement.getNumero()-1].getY()+mouvY)>monMouvement.getMouv

    mesVignettes[monMouvement.getNumero()-1].setY(monMouvement.getMouvements()[1][1
                    }else{
                        mesVignettes[monMouvement.getNumero()-1].addY(mouvY);
                    }
                }
                monMouvement.setAnterieur(touchX, touchY);
                invalidate();
            }
            return true;
        }

    }
```

La classe Vignette

Cette classe nommée `Vignette` rassemble toutes les informations liées à chaque case du jeu. On y retrouve notamment le numéro de la case, l'image à tracer ainsi que sa position à l'écran. Cette classe regroupe également l'ensemble des méthodes permettant de gérer et modifier ces attributs. Voici le code de cette classe :

Code : Java - Vignette.java

```
package sdz.taquin;

import net.rim.device.api.math.Fixed32;
import net.rim.device.api.system.EncodedImage;

public class Vignette {

    private int numero;
    private EncodedImage image;
    private int[] position;

    public Vignette(int i, int echelle, int x, int y){
        numero = i;
        int taille = (int) Math.ceil(10000/echelle);
        EncodedImage img =
EncodedImage.getEncodedImageResource("vignette"+numero+".png");
        image = img.scaleImage32(Fixed32.tenThouToFP(taille),
Fixed32.tenThouToFP(taille));
        position = new int[2];
        position[0] = x*echelle*80;
        position[1] = y*echelle*80;
    }

    public EncodedImage getImage(){
        return image;
    }

    public int getX(){
        return position[0];
    }

    public int getY(){
        return position[1];
    }

    public void setPosition(int x, int y){
        position[0] = x;
        position[1] = y;
    }

    public void setX(int x){
        position[0] = x;
    }

    public void setY(int y){
        position[1] = y;
    }

    public void addX(int x){
        position[0] += x;
    }

    public void addY(int y){
        position[1] += y;
    }
}
```

La classe Partie

La classe `Partie` sert à la gestion du jeu en cours. Elle possède en unique attribut un tableau décrivant l'état du jeu ainsi que toutes les méthodes nécessaires au bon déroulement du jeu. Le code source de cette classe est écrit ci-dessous :

Code : Java - Partie.java

```
package sdz.taquin;

import java.util.Random;

public class Partie {

    private int[][] jeu = new int[4][3];

    public Partie(){
        melanger(30);
    }

    public int getVignette(int i, int j){
        return jeu[i][j];
    }

    public void melanger(int n){
        for(int i=0; i<4; i++){
            for(int j=0; j<3; j++){
                jeu[i][j] = i+j*4;
            }
        }
        int k = n;
        int[] positionZero = {0,0};
        Random monRandom = new Random();
        do{
            int nb = monRandom.nextInt(4);
            switch(nb){
                case 0:
                    if(positionZero[0]>0){

jeu[positionZero[0]][positionZero[1]]=jeu[positionZero[0]-1][positionZero[1]];
                        jeu[positionZero[0]-1][positionZero[1]]=0;
                        positionZero[0]--;
                        k--;
                    }
                    break;
                case 1:
                    if(positionZero[1]>0){

jeu[positionZero[0]][positionZero[1]]=jeu[positionZero[0]][positionZero[1]-1];
                        jeu[positionZero[0]][positionZero[1]-1]=0;
                        positionZero[1]--;
                        k--;
                    }
                    break;
                case 2:
                    if(positionZero[0]<3){

jeu[positionZero[0]][positionZero[1]]=jeu[positionZero[0]+1][positionZero[1]];
                        jeu[positionZero[0]+1][positionZero[1]]=0;
                        positionZero[0]++;
                        k--;
                    }
                    break;
                case 3:
                    if(positionZero[1]<2){

jeu[positionZero[0]][positionZero[1]]=jeu[positionZero[0]][positionZero[1]+1];
                        jeu[positionZero[0]][positionZero[1]+1]=0;
                        positionZero[1]++;
                        k--;
                    }
                    break;
```

```
                }
            }while(k>0);
        }

        public int[][] estJouable(int i, int j, int echelle){
            int[][] mouvements =
    {{i*echelle*80,i*echelle*80},{j*echelle*80,j*echelle*80}};
            if(i>0){
                if(jeu[i-1][j]==0){
                    mouvements[0][0] = (i-1)*echelle*80;
                }
            }
            if(j>0){
                if(jeu[i][j-1]==0){
                    mouvements[1][0] = (j-1)*echelle*80;
                }
            }
            if(i<3){
                if(jeu[i+1][j]==0){
                    mouvements[0][1] = (i+1)*echelle*80;
                }
            }
            if(j<2){
                if(jeu[i][j+1]==0){
                    mouvements[1][1] = (j+1)*echelle*80;
                }
            }
            return mouvements;
        }

        public void placer(int i, int j, int numero){
            if(jeu[i][j]==0){
                jeu[i][j]=numero;
                if(i>0){
                    if(jeu[i-1][j]==numero){
                        jeu[i-1][j]=0;
                    }
                }
                if(j>0){
                    if(jeu[i][j-1]==numero){
                        jeu[i][j-1]=0;
                    }
                }
                if(i<3){
                    if(jeu[i+1][j]==numero){
                        jeu[i+1][j]=0;
                    }
                }
                if(j<2){
                    if(jeu[i][j+1]==numero){
                        jeu[i][j+1]=0;
                    }
                }
            }
        }

        public boolean estEnPlace(){
            boolean resultat = true;
            for(int i=0; i<4; i++){
                for(int j=0; j<3; j++){
                    if(jeu[i][j] != i+j*4){
                        resultat = false;
                    }
                }
            }
            return resultat;
        }
    }
```

La classe Mouvement

La classe `Mouvement` permet de gérer les mouvements d'une vignette, en particulier lors d'un mouvement *transitoire* de vignette d'une position à une autre. Ces mouvements ne modifient pas directement l'état du jeu, cependant l'affichage à l'écran doit tout de même être mis à jour. Voici le code de cette classe :

Code : Java - Mouvement.java

```
package sdz.taquin;

public class Mouvement {

    private int numero;
    public boolean actif;
    private int[][] mouvements;
    private int[] positionAnt;

    public Mouvement(){
        numero=0;
        actif = false;
        mouvements = new int[2][2];
        positionAnt = new int[2];
        positionAnt[0] = 160;
        positionAnt[1] = 0;
    }

    public void setMouvements(){
        actif = false;
    }

    public void setMouvements(int[][] mouv){
        mouvements = mouv;
        actif = false;
        if(mouv[0][0]!=mouv[0][1] || mouv[1][0]!=mouv[1][1]){
            actif = true;
        }
    }

    public void setNumero(int num){
        numero = num;
    }

    public void setAnterieur(int x, int y){
        positionAnt[0] = x;
        positionAnt[1] = y;
    }

    public int getNumero(){
        return numero;
    }

    public boolean estActif(){
        return actif;
    }

    public int[][] getMouvements(){
        return mouvements;
    }

    public int[] getAnterieur(){
        return positionAnt;
    }
}
```

Les explications

À présent, nous allons étudier le contenu de ce code source plus en détails. Encore une fois, la méthode utilisée pour réaliser ce jeu est propre à chacun. Ainsi, la « correction » proposée ci-dessous n'est pas la seule manière de concevoir un jeu de taquin.

La structure du jeu

Préparation des vignettes

Vous vous en doutez certainement, nous allons parler ici de la classe `Vignette`. Comme dit plus haut, cette classe rassemble toutes les informations liées à chaque case du jeu et nous y trouvons le numéro de la case, l'image à tracer ainsi que sa position à l'écran. Voici donc ces différents attributs :

Code : Java

```
private int numero;
private EncodedImage image;
private int[] position;
```

Attention, nous parlons ici de la position de l'image à l'écran en *pixels*.

Jusque-là, rien de bien impressionnant. En revanche, le constructeur de la classe est légèrement plus pointu : il va nous falloir initialiser ces attributs. Pour cela nous aurons besoin d'un certain nombre de paramètres :

- le numéro de la vignette
- l'échelle pour pouvoir tracer correctement les images à l'écran
- la position de la vignette dans le taquin

Voici donc le prototype de notre constructeur :

Code : Java

```
public Vignette(int i, int echelle, int x, int y){
    // Instructions
}
```

Le numéro de la vignette doit être stocké, nous pouvons directement initialiser notre attribut `numero`. De plus en sachant que nos vignettes font à l'origine 80×80 pixels, nous pouvons également initialiser la position de la vignette :

Code : Java

```
numero = i;
position = new int[2];
position[0] = x*echelle*80;
position[1] = y*echelle*80;
```

Enfin pour finir, il nous faut charger l'image correspondant à la vignette. Nous allons alors simplement utiliser la méthode présentée au chapitre précédent, à savoir :

Code : Java

```
int taille = (int) Math.ceil(10000/echelle);
EncodedImage img =
EncodedImage.getEncodedImageResource("vignette"+numero+".png");
image = img.scaleImage32(Fixed32.tenThouToFP(taille),
Fixed32.tenThouToFP(taille));
```

L'utilité de cette classe réside uniquement dans ses attributs. C'est pourquoi, les seules méthodes présentes dans cette classe sont exclusivement des accesseurs. Nous ne les détaillerons pas, je vous laisse le soin de les découvrir.

Le cœur du jeu

Comme l'indique le titre, nous allons ici nous occuper du cœur de jeu, à savoir la gestion de la partie et de la disposition des vignettes. Nous avons donc besoin d'une nouvelle classe nommée `Partie`. Celle-ci possède un seul attribut : il s'agit d'un tableau 2D comportant le numéro des vignettes à chaque position. Celles-ci sont numérotées de 1 à 11, le 0 étant réservé au « blanc ». Dans un premier temps, nous pouvons placer les vignettes dans l'ordre. Voici donc le code correspondant à la déclaration de l'attribut ainsi que le constructeur provisoire :

Code : Java

```
private int[][] jeu = new int[4][3];

public Partie(){
    for(int i=0; i<4; i++){
        for(int j=0; j<3; j++){
            jeu[i][j] = i+j*4;
        }
    }
}
```

Attaquons-nous maintenant aux différentes méthodes dont nous aurons besoin. Attaquons par un accesseur, voici une méthode permettant de récupérer le numéro d'une vignette suivant sa position dans le tableau :

Code : Java

```
public int getVignette(int i, int j){
    return jeu[i][j];
}
```

Ensuite, voici une méthode permettant de savoir si les vignettes sont en place. Celle-ci nous renvoie un booléen suivant le résultat.

Code : Java

```
public boolean estEnPlace(){
    boolean resultat = true;
    for(int i=0; i<4; i++){
        for(int j=0; j<3; j++){
            if(jeu[i][j] != i+j*4){
                resultat = false;
            }
        }
    }
    return resultat;
}
```

La méthode suivante permet de savoir si une vignette peut être déplacée. À partir de la position d'une vignette dans le tableau, nous allons scruter les éléments adjacents pour savoir si l'un d'eux est le « blanc ». Dans ce cas la vignette peut être déplacée dans cette direction. En retour, nous renvoyons un tableau 2D indiquant quels sont les déplacements possibles en pixels.

Code : Java

```
public int[][] estJouable(int i, int j, int echelle){
    int[][] mouvements =
{{i*echelle*80,i*echelle*80},{j*echelle*80,j*echelle*80}};
    if(i>0){
        if(jeu[i-1][j]==0){
            mouvements[0][0] = (i-1)*echelle*80;
        }
    }
```

```
        if(j>0){
            if(jeu[i][j-1]==0){
                mouvements[1][0] = (j-1)*echelle*80;
            }
        }
        if(i<3){
            if(jeu[i+1][j]==0){
                mouvements[0][1] = (i+1)*echelle*80;
            }
        }
        if(j<2){
            if(jeu[i][j+1]==0){
                mouvements[1][1] = (j+1)*echelle*80;
            }
        }
        return mouvements;
    }
```

Puis, nous aurons également besoin d'une méthode pour déplacer un élément. Nous plaçons alors cet élément à l'endroit spécifié, et nous remplaçons son ancienne position par le « blanc ».

Code : Java

```
    public void placer(int i, int j, int numero){
        if(jeu[i][j]==0){
            jeu[i][j]=numero;
            if(i>0){
                if(jeu[i-1][j]==numero){
                    jeu[i-1][j]=0;
                }
            }
            if(j>0){
                if(jeu[i][j-1]==numero){
                    jeu[i][j-1]=0;
                }
            }
            if(i<3){
                if(jeu[i+1][j]==numero){
                    jeu[i+1][j]=0;
                }
            }
            if(j<2){
                if(jeu[i][j+1]==numero){
                    jeu[i][j+1]=0;
                }
            }
        }
    }
```

Enfin pour finir, nous allons modifier le constructeur pour avoir un jeu mélangé en début de partie. C'est là que nous allons introduire la méthode `melanger()`. Pour mélanger ce jeu, nous devons retenir la position du « blanc ». Ensuite il nous suffit d'intervertir sa place avec l'un de ses voisins. Après avoir réitéré l'opération plusieurs fois, nous avons un jeu suffisamment mélangé. Voici le code correspondant :

Code : Java

```
    public Partie(){
        melanger(30);
    }

    public void melanger(int n){
        for(int i=0; i<4; i++){
            for(int j=0; j<3; j++){
                jeu[i][j] = i+j*4;
            }
```

```
        }
        int k = n;
        int[] positionZero = {0,0};
        Random monRandom = new Random();
        do{
            int nb = monRandom.nextInt(4);
            switch(nb){
                case 0:
                    if(positionZero[0]>0){

    jeu[positionZero[0]][positionZero[1]]=jeu[positionZero[0]-1][positionZero[1]];
                        jeu[positionZero[0]-1][positionZero[1]]=0;
                        positionZero[0]--;
                        k--;
                    }
                    break;
                case 1:
                    if(positionZero[1]>0){

    jeu[positionZero[0]][positionZero[1]]=jeu[positionZero[0]][positionZero[1]-1];
                        jeu[positionZero[0]][positionZero[1]-1]=0;
                        positionZero[1]--;
                        k--;
                    }
                    break;
                case 2:
                    if(positionZero[0]<3){

    jeu[positionZero[0]][positionZero[1]]=jeu[positionZero[0]+1][positionZero[1]];
                        jeu[positionZero[0]+1][positionZero[1]]=0;
                        positionZero[0]++;
                        k--;
                    }
                    break;
                case 3:
                    if(positionZero[1]<2){

    jeu[positionZero[0]][positionZero[1]]=jeu[positionZero[0]][positionZero[1]+1];
                        jeu[positionZero[0]][positionZero[1]+1]=0;
                        positionZero[1]++;
                        k--;
                    }
                    break;
            }
        }while(k>0);
    }
```

Nous voilà enfin au bout de cette classe. N'hésitez pas à relire cette partie plusieurs fois pour bien comprendre le fonctionnement.

La gestion des mouvements

Nous allons ici introduire la classe `Mouvement`. Celle-ci est en charge des mouvements *transitoires* de vignettes d'une position à une autre. Pour pouvoir gérer ces déplacements, nous aurons besoin de différents attributs. Nous allons définir le numéro de la vignette sélectionnée, les mouvements possibles de cette vignette ainsi que la position de l'évènement précédent pour pouvoir calculer le déplacement du doigt à l'écran. Un dernier attribut permet de définir si le déplacement est en cours ou non, c'est-à-dire si l'utilisateur a toujours le doigt sur l'écran ou non. Voici donc la liste des attributs définis pour cette classe :

Code : Java

```
    private int numero;
    public boolean actif;
    private int[][] mouvements;
    private int[] positionAnt;
```

Le constructeur ne mérite pas de longues explications. Les attributs peuvent être initialisés avec n'importe quelles valeurs, du moment que l'attribut `actif` est *faux*. Le constructeur est défini ci-dessous :

Code : Java

```
public Mouvement(){
    numero=0;
    actif = false;
    mouvements = new int[2][2];
    positionAnt = new int[2];
    positionAnt[0] = 160;
    positionAnt[1] = 0;
}
```

Dans cette classe encore, nous retrouvons principalement des accesseurs. Nous ne les détaillerons pas tous, mais nous étudierons quand même les méthodes nommées `setMouvements()`. La première permet de mettre à jour l'attribut `mouvements` ainsi que d'autoriser le mouvement si la vignette peut être déplacée :

Code : Java

```
public void setMouvements(int[][] mouv){
    mouvements = mouv;
    actif = false;
    if(mouv[0][0]!=mouv[0][1] || mouv[1][0]!=mouv[1][1]){
        actif = true;
    }
}
```

Et pour finir, la deuxième méthode permet de désactiver le mouvement :

Code : Java

```
public void setMouvements(){
    actif = false;
}
```

Nous voilà fin prêt, il ne nous reste plus qu'à tracer les éléments à l'écran.

Le traçage à l'écran

Maintenant que tout est prêt, il nous faut rassembler l'ensemble et tracer les différents éléments. Tout ceci, nous allons le faire à l'intérieur de la classe `MonDessin`. Nous utiliserons toutes les notions vues au chapitre précédent sur les vues personnalisées. Faisons à présent un tour des attributs dont nous aurons besoin. Tout d'abord, nous prendrons une variable pour retenir l'échelle de l'écran (jeu en 320×240 pixels). Ensuite nous aurons besoin de charger l'ensemble des vignettes, nous utiliserons un tableau de `Vignette`. Enfin nous prendrons une occurrence de chacune des deux classes `Partie` et `Mouvement`. Voici les attributs :

Code : Java

```
private int echelle;
private Vignette[] mesVignettes;
private Partie maPartie;
private Mouvement monMouvement;
```

Aucune surprise dans le constructeur si ce n'est la nécessité de réaliser une boucle pour faire appel aux constructeurs de la classe `Vignette`. Voici ce constructeur :

Code : Java

```
    public MonDessin(){
        super();
        echelle = Math.min(Display.getWidth()/320,
    Display.getHeight()/240);
        maPartie = new Partie();
        monMouvement = new Mouvement();
        mesVignettes = new Vignette[11];
        for(int i=0; i<4; i++){
            for(int j=0; j<3; j++){
                if(maPartie.getVignette(i,j)!=0){
                    mesVignettes[maPartie.getVignette(i,j)-1] = new
    Vignette(maPartie.getVignette(i,j),echelle, i, j);
                }
            }
        }
    }
```

Nous voici arrivés à l'endroit qui va donner vie à notre jeu. Vous l'aurez deviné, il s'agit bien évidemment de la gestion des évènements. Nous utiliserons les évènements suivants: `TouchEvent.DOWN`, `TouchEvent.UP` et `TouchEvent.MOVE`. Pour commencer, nous allons nous occuper du cas où l'utilisateur pose son doigt sur l'écran. Il faut alors dans un premier temps trouver quelle vignette a été touchée. Ensuite nous regardons si cette vignette peut être déplacée. Puis, il ne nous reste plus qu'à mettre à jour les attributs de la classe `Mouvement`. Voici donc le code correspondant :

Code : Java

```
    if(eventCode == TouchEvent.DOWN) {
        int i = touchX/(80*echelle);
        int j = touchY/(80*echelle);
        monMouvement.setNumero(maPartie.getVignette(i,j));
        int[][] mouv = maPartie.estJouable(i,j, echelle);
        monMouvement.setMouvements(mouv);
        monMouvement.setAnterieur(touchX, touchY);
        invalidate();
    }
```

Ensuite nous devons gérer le cas où le doigt bouge à l'écran. Tout d'abord nous devons calculer le déplacement effectué depuis le dernier évènement. Ensuite si le mouvement est possible, nous mettons à jour la position de la vignette. Pour cela, nous regardons si le déplacement proposé s'insère dans l'intervalle de déplacement autorisé. Voici comment j'ai réalisé ceci :

Code : Java

```
    if(eventCode == TouchEvent.MOVE) {
        int mouvX = touchX - monMouvement.getAnterieur()[0];
        int mouvY = touchY - monMouvement.getAnterieur()[1];
        if(monMouvement.getNumero()!=0 && monMouvement.estActif()==true){

    if((mesVignettes[monMouvement.getNumero()-1].getX()+mouvX)<monMouvement.getMouve
                mesVignettes[monMouvement.getNumero()-1].setX(monMouvement.getMouven
            }else
    if((mesVignettes[monMouvement.getNumero()-1].getX()+mouvX)>monMouvement.getMouve
                mesVignettes[monMouvement.getNumero()-1].setX(monMouvement.getMouven
            }else{
                 mesVignettes[monMouvement.getNumero()-1].addX(mouvX);
            }

    if((mesVignettes[monMouvement.getNumero()-1].getY()+mouvY)<monMouvement.getMouve
                mesVignettes[monMouvement.getNumero()-1].setY(monMouvement.getMouven
            }else
    if((mesVignettes[monMouvement.getNumero()-1].getY()+mouvY)>monMouvement.getMouve
                mesVignettes[monMouvement.getNumero()-1].setY(monMouvement.getMouven
            }else{
                mesVignettes[monMouvement.getNumero()-1].addY(mouvY);
            }
        }
```

```
            monMouvement.setAnterieur(touchX, touchY);
            invalidate();
        }
```

Enfin, une fois que l'utilisateur retire son doigt de l'écran, nous devons vérifier si le jeu a été modifié. Donc nous devons déterminer sur quelle position se trouve alors la vignette déplacée par un petit calcul. Ensuite, nous devons réactualiser le jeu, en appelant quelques accesseurs et autres méthodes :

Code : Java

```
if(eventCode == TouchEvent.UP) {
    if(monMouvement.getNumero()!=0 && monMouvement.estActif()==true){
        int i =
(mesVignettes[monMouvement.getNumero()-1].getX()+40*echelle)/(80*echelle);
        int j =
(mesVignettes[monMouvement.getNumero()-1].getY()+40*echelle)/(80*echelle);
        monMouvement.setMouvements();
        maPartie.placer(i, j, monMouvement.getNumero());
        mesVignettes[monMouvement.getNumero()-1].setPosition(i*80*echelle,
j*80*echelle);
    }
    invalidate();
    if(maPartie.estEnPlace()==true){
        Dialog.alert("Bravo, vous avez réussi!");
        System.exit(0);
        return true;
    }
}
```

La deuxième partie de ce bloc d'instructions correspond à la fin de la partie. Une fois que toutes les pièces sont en place, nous affichons une boîte de dialogue puis mettons fin à l'application.

N'hésitez pas à améliorer cette application, vous trouverez certainement de nouvelles fonctionnalités à réaliser. Vous pouvez également essayer de réorganiser les classes et d'optimiser le code.

Partie 3 : Les techniques avancées

Travailler avec plusieurs vues

Pour démarrer en douceur cette troisième partie, je vous propose que nous revenions un peu sur les vues. Dans la partie précédente, nous avons appris à réaliser divers types de vues. Maintenant nous allons voir comment travailler avec plusieurs vues, et comment passer facilement d'une à autre. Nous verrons donc très peu de théorie, mais plutôt diverses techniques de gestion des vues.
Ce chapitre va donc nous permettre de démarrer cette nouvelle partie en douceur. Je vous invite donc à vous lancer dans la suite de ce cours !

Préparation des vues

Nous allons ici préparer les différentes vues que nous utiliserons dans la suite. Ce sera pour vous l'occasion de faire quelques révisions sur ce que nous avons déjà vu !

La page d'accueil

Objectif

Pour notre première vue, nous allons utiliser principalement deux éléments, à savoir une image et un bouton. Ceux-ci seront alors centrés à l'écran avec un dégradé de couleurs en arrière-plan.
Voilà donc à quoi cela va ressembler :

Le fond et le centrage

Pour gérer l'ensemble de la vue de la page d'accueil, nous allons déclarer un conteneur nommé `accueil`. Afin que les éléments soient centrés au final, nous allons créer un conteneur de type `HorizontalFieldManager` auquel nous imposerons d'utiliser toute la place disponible verticalement.
Voilà donc comment nous commençons :

Code : Java

```
HorizontalFieldManager accueil = new
HorizontalFieldManager(Field.USE_ALL_HEIGHT);
```

Nous allons également profiter de cette déclaration pour créer le dégradé souhaité. Pour cela, nous allons redéfinir la méthode `paint()` du conteneur :

Code : Java

```
HorizontalFieldManager accueil = new
HorizontalFieldManager(Field.USE_ALL_HEIGHT){
    public void paint(Graphics g){
        g.drawGradientFilledRect(0, 0, Display.getWidth(),
Display.getHeight(), 0x009FCFFD, 0x0079B6FC);
        super.paint(g);
    }
};
```

Soyez attentifs au fait que l'appel de la méthode de la superclasse `super.paint()` se fait après la création du dégradé. En effet si vous ne réalisez pas cela dans cet ordre, vous verrez alors apparaître le dégradé par-dessus les composants enfants du conteneur.

À présent nous allons créer un nouveau conteneur vertical. Celui-ci sera centré verticalement et utilisera toute la largeur de l'écran. À l'intérieur, nous pourrons ainsi ajouter nos deux éléments et les centrer.
Voici donc le code correspondant :

Code : Java

```
VerticalFieldManager conteneur = new
VerticalFieldManager(Field.FIELD_VCENTER|Field.USE_ALL_WIDTH);
```

Ajout des composants

Maintenant occupons-nous des composants de notre vue, à savoir une image et un bouton !
Pour l'image, j'ai utilisé le logo du Site du Zéro que je vous invite à télécharger pour pouvoir travailler en même temps que moi :

logo_sdz.png

Nous allons donc nous servir de la classe `BitmapField` pour tracer l'image à l'écran. Nous en profiterons également pour centrer celle-ci horizontalement.
Voilà comment faire :

Code : Java

```
Bitmap logo = Bitmap.getBitmapResource("logo_sdz.png");
BitmapField monBitmap = new BitmapField(logo,Field.FIELD_HCENTER);
```

Ensuite, créons notre bouton :

Code : Java

```
ButtonField entrer = new ButtonField("ENTRER",Field.FIELD_HCENTER);
```

Enfin, il ne nous reste plus qu'à ajouter l'ensemble dans les différents conteneurs :

Code : Java

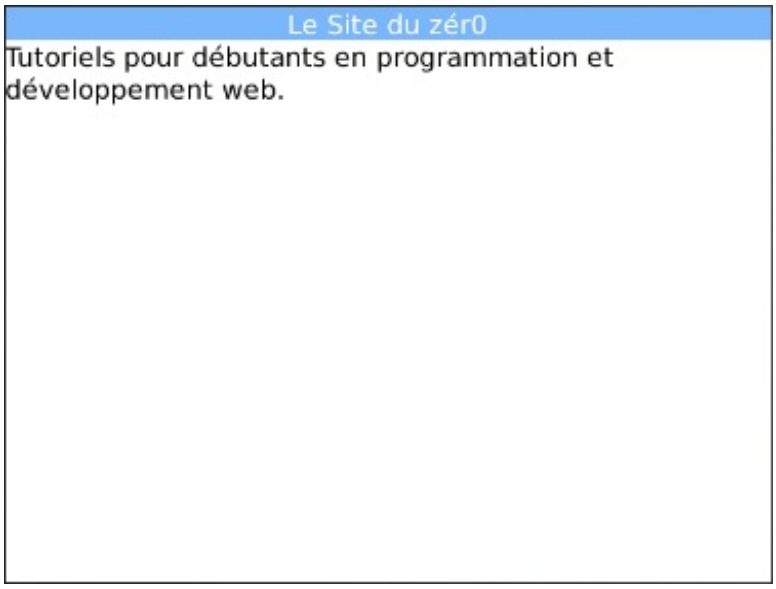
```
conteneur.add(monBitmap);
conteneur.add(entrer);
accueil.add(conteneur);
```

À ce stade, nous avons donc un conteneur nommé `accueil` qui contient l'ensemble de notre page d'accueil !

La page de description

Objectif

Maintenant nous allons créer une page de description du Site du Zéro.
Pour cela, nous utiliserons donc deux champs de texte, un pour l'intitulé et le second pour la description du site.



Page

L'entête

Comme précédemment, nous allons commencer par créer un conteneur nommé `page`, qui contiendra donc l'ensemble des éléments de la page de description :

Code : Java

```
VerticalFieldManager page = new VerticalFieldManager();
```

Ensuite, nous allons créer un second conteneur de type `VerticalFieldManager` qui nous servira à appliquer une couleur de fond à notre entête. Nous lui imposerons donc d'utiliser toute la largeur de l'écran.
Voici le code :

Code : Java

```
VerticalFieldManager entete = new
VerticalFieldManager(Field.USE_ALL_WIDTH);
Background maCouleur =
BackgroundFactory.createSolidBackground(0x0079B6FC);
entete.setBackground(maCouleur);
```

En ce qui concerne le titre de la vue, nous allons utiliser un champ de texte de type `LabelField`. En effet, il s'agit du seul champ de texte qui ne prend pas l'ensemble de la largeur lors de sa création. Ainsi nous pourrons le centrer, et aussi nous redéfinirons sa couleur d'écriture.
Voilà comment faire tout ceci :

Code : Java

```
LabelField titre = new LabelField("Le Site du
zér0",Field.FIELD_HCENTER){
```

```
        public void paint(Graphics g){
            g.setColor(0x00FFFFFF);
            super.paint(g);
        }
    };
```

La description

Pour la suite, il n'y a rien de compliqué ou de surprenant. Effectivement, nous allons simplement utiliser un `RichTextField` de la manière suivante :

Code : Java

```
RichTextField description = new RichTextField();
description.setText("Tutoriels pour débutants en programmation et développement web.");
```

Nous n'avons alors plus qu'à tout insérer dans les différents conteneurs :

Code : Java

```
entete.add(titre);
page.add(entete);
page.add(description);
```

Pour la suite, nous avons donc également un conteneur nommé `page` qui contient l'ensemble de la page de description.

Mise à jour d'un écran

Le principe

Définition des vues

La première technique consiste à utiliser un seul écran, à savoir notre `MyScreen` !
Dans ce cas, nous pouvons déclarer deux attributs `accueil` et `page` qui contiendront donc l'ensemble de nos deux vues.
Nous pouvons alors définir ces deux vues à l'intérieur du constructeur de notre classe.

Voilà donc à quoi ressemble la définition de nos deux vues :

Code : Java

```
package mypackage;

import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.Manager;
import net.rim.device.api.ui.component.BitmapField;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.HorizontalFieldManager;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;

public final class MyScreen extends MainScreen
{
```

```
        private HorizontalFieldManager accueil;
        private VerticalFieldManager page;

        public MyScreen()
        {
            super(Manager.NO_VERTICAL_SCROLL);

            // Accueil
            accueil = new HorizontalFieldManager(Field.USE_ALL_HEIGHT){
                public void paint(Graphics g){
                    g.drawGradientFilledRect(0, 0, Display.getWidth(),
    Display.getHeight(), 0x009FCFFD, 0x0079B6FC);
                    super.paint(g);
                }
            };
            VerticalFieldManager conteneur = new
    VerticalFieldManager(Field.FIELD_VCENTER|Field.USE_ALL_WIDTH);
            Bitmap logo = Bitmap.getBitmapResource("logo_sdz.png");
            BitmapField monBitmap = new
    BitmapField(logo,Field.FIELD_HCENTER);
            ButtonField entrer = new
    ButtonField("ENTRER",Field.FIELD_HCENTER);
            conteneur.add(monBitmap);
            conteneur.add(entrer);
            accueil.add(conteneur);

            // Page
            page = new VerticalFieldManager();
            VerticalFieldManager entete = new
    VerticalFieldManager(Field.USE_ALL_WIDTH);
            Background maCouleur =
    BackgroundFactory.createSolidBackground(0x0079B6FC);
            entete.setBackground(maCouleur);
            LabelField titre = new LabelField("Le Site du
    zér0",Field.FIELD_HCENTER){
                public void paint(Graphics g){
                    g.setColor(0x00FFFFFF);
                    super.paint(g);
                }
            };
            RichTextField description = new RichTextField();
            description.setText("Tutoriels pour débutants en
    programmation et développement web.");
            entete.add(titre);
            page.add(entete);
            page.add(description);
        }
    }
```

La mise à jour de l'écran

À l'ouverture de l'application, nous voulons que l'utilisateur tombe sur la vue `accueil`. C'est pourquoi pour commencer, nous n'avons qu'à ajouter cette vue au conteneur principal :

Code : Java

```
add(accueil);
```

L'objectif est alors de remplacer la vue `accueil` par `page` lors du clic du bouton. Pour cela nous allons nous servir de la méthode `deleteAll()` de la classe `Manager`. Effectivement, cette méthode permet de supprimer l'ensemble des composants enfants d'un conteneur. Ainsi nous pourrons par la suite ajouter notre nouvelle vue : `page`.
Voilà donc le code qui nous permettra de faire la transition entre les deux vues :

Code : Java

```
deleteAll();
add(page);
```

Finalement, nous n'avons plus qu'à ajouter ce code à l'intérieur de la méthode de gestion des évènements du bouton, comme ceci :

Code : Java

```
entrer.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        deleteAll();
        add(page);
    }
});
```

Le code complet

Au final pour utiliser les deux vues définies précédemment, nous avons donc le code suivant :

Code : Java

```
package mypackage;

import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.Manager;
import net.rim.device.api.ui.component.BitmapField;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.HorizontalFieldManager;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;

public final class MyScreen extends MainScreen
{
    private HorizontalFieldManager accueil;
    private VerticalFieldManager page;

    public MyScreen()
    {
        super(Manager.NO_VERTICAL_SCROLL);

        // Accueil
        accueil = new HorizontalFieldManager(Field.USE_ALL_HEIGHT){
            public void paint(Graphics g){
                g.drawGradientFilledRect(0, 0, Display.getWidth(),
Display.getHeight(), 0x009FCFFD, 0x0079B6FC);
                super.paint(g);
            }
        };
        VerticalFieldManager conteneur = new
VerticalFieldManager(Field.FIELD_VCENTER|Field.USE_ALL_WIDTH);
        Bitmap logo = Bitmap.getBitmapResource("logo_sdz.png");
        BitmapField monBitmap = new
BitmapField(logo,Field.FIELD_HCENTER);
        ButtonField entrer = new
```

```
ButtonField("ENTRER",Field.FIELD_HCENTER);
        conteneur.add(monBitmap);
        conteneur.add(entrer);
        accueil.add(conteneur);

        // Page
        page = new VerticalFieldManager();
        VerticalFieldManager entete = new
VerticalFieldManager(Field.USE_ALL_WIDTH);
        Background maCouleur =
BackgroundFactory.createSolidBackground(0x0079B6FC);
        entete.setBackground(maCouleur);
        LabelField titre = new LabelField("Le Site du
zér0",Field.FIELD_HCENTER){
            public void paint(Graphics g){
                g.setColor(0x00FFFFFF);
                super.paint(g);
            }
        };
        RichTextField description = new RichTextField();
        description.setText("Tutoriels pour débutants en
programmation et développement web.");
        entete.add(titre);
        page.add(entete);
        page.add(description);

        // Gestion de la transition entre les vues
        add(accueil);
        entrer.setChangeListener(new FieldChangeListener() {
            public void fieldChanged(Field field, int context) {
                deleteAll();
                add(page);
            }
        });
    }
}
```

Travailler par héritage
Utiliser réellement plusieurs vues

Introduction

Comme vous vous en doutez certainement, la méthode précédente n'est pas vraiment la plus appropriée pour gérer plusieurs vues !

Pourquoi, cette technique fonctionne bien ?

Effectivement la technique précédente fonctionne très bien et est dans ce cas-là relativement simple d'utilisation. Toutefois nous n'avons pas réalisé de vues extrêmement complexes et nous n'avions que deux vues différentes. Imaginez maintenant que vous vouliez faire une application qui sera divisée en une multitude de vues. Vous pourriez bien évidemment utiliser une seule classe pour décrire l'ensemble de vos vues, mais il faut avouer que ce serait un peu le bazar !

Le principe

Je vous ai déjà légèrement lâché le morceau, mais effectivement nous allons utiliser plusieurs classes. Jusqu'à présent, nous n'avons utilisé qu'une unique classe pour décrire les vues de nos applications : `MyScreen`. C'est pourquoi nous allons maintenant définir de « vraies » vues, en définissant plusieurs classes qui héritent de `MainScreen`. Nous pourrons ainsi passer d'une vue à une autre en utilisant la méthode `pushScreen()` de la classe `UiApplication`, dont je suis certain que vous aviez oublié l'existence !
Cependant, nous sommes ici confronté à un léger problème.

En effet, comment accéder à la méthode `pushScreen()` depuis l'intérieur d'une classe « enfant » ?

Dans certains langages tels que l'Actionscript, il existe un mot-clé `parent` qui permet d'accéder à la classe « parente » dont l'instance en est un enfant. Cependant, ce mot-clé n'a pas d'équivalent en Java, et nous allons donc devoir faire autrement.
En réalité, nous allons déclarer un attribut de type `UiApplication` à l'intérieur de nos `MainScreen`, afin de pouvoir passer une référence de notre application. Ne vous inquiétez pas si vous ne comprenez pas bien, nous allons tout de suite voir ce que cela donne au niveau du code.

Nos deux vues

Comme nous l'avons dit, nous allons à présent utiliser une classe pour chacune de nos vues. Je vous laisse donc découvrir les classes `Accueil` et `Page`, où nous avons séparé la définition des deux vues.

La classe `Accueil`

Code : Java - Accueil.java

```
package mypackage;

import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.Manager;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.BitmapField;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.container.HorizontalFieldManager;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;

public final class Accueil extends MainScreen
{
    private UiApplication monApp;

    public Accueil(UiApplication app)
    {
        super(Manager.NO_VERTICAL_SCROLL);
        monApp = app;
        // Accueil
        HorizontalFieldManager accueil = new
HorizontalFieldManager(Field.USE_ALL_HEIGHT){
            public void paint(Graphics g){
                g.drawGradientFilledRect(0, 0, Display.getWidth(),
Display.getHeight(), 0x009FCFFD, 0x0079B6FC);
                super.paint(g);
            }
        };
        VerticalFieldManager conteneur = new
VerticalFieldManager(Field.FIELD_VCENTER|Field.USE_ALL_WIDTH);
        Bitmap logo = Bitmap.getBitmapResource("logo_sdz.png");
        BitmapField monBitmap = new
BitmapField(logo,Field.FIELD_HCENTER);
        ButtonField entrer = new
ButtonField("ENTRER",Field.FIELD_HCENTER);
        conteneur.add(monBitmap);
        conteneur.add(entrer);
        accueil.add(conteneur);
        add(accueil);
    }
}
```

La classe `Page`

Code : Java - Page.java

```
package mypackage;

import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.Manager;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;

public class Page extends MainScreen{

    private UiApplication monApp;

    public Page(UiApplication app){
        super(Manager.NO_VERTICAL_SCROLL);
        monApp = app;
        // Page
        VerticalFieldManager page = new VerticalFieldManager();
        VerticalFieldManager entete = new
VerticalFieldManager(Field.USE_ALL_WIDTH);
        Background maCouleur =
BackgroundFactory.createSolidBackground(0x0079B6FC);
        entete.setBackground(maCouleur);
        LabelField titre = new LabelField("Le Site du
zér0",Field.FIELD_HCENTER){
            public void paint(Graphics g){
                g.setColor(0x00FFFFFF);
                super.paint(g);
            }
        };
        RichTextField description = new RichTextField();
        description.setText("Tutoriels pour débutants en
programmation et développement web.");
        entete.add(titre);
        page.add(entete);
        page.add(description);
        add(page);

    }

}
```

Afficher les vues

Au lancement de l'application

Étant donné que nous avons ajouté un attribut `UiApplication` à chacune de nos classes, et que leurs constructeurs prennent un nouveau paramètre, nous allons donc devoir modifier légèrement la classe principale. Pour cela, nous enverrons donc une référence à l'application en cours en utilisant le mot-clé **`this`**.
Voilà donc le nouveau constructeur de notre classe principale :

Code : Java

```
public MonApp()
{
    pushScreen(new Accueil(this));
}
```

Ainsi nous aurons donc accès à cette méthode `pushScreen()` depuis l'intérieur de nos classes de vues.

La transition entre les vues

Maintenant que nous avons une référence à notre application, nous pouvons donc changer de vue, simplement grâce à l'instruction suivante :

Code : Java

```
monApp.pushScreen(new Page(monApp));
```

Il nous suffit donc de l'insérer à l'intérieur de la méthode `fielChanged()` qui nous permet de gérer les évènements de notre bouton `entrer`.
Voilà donc l'action associée au bouton `entrer` :

Code : Java

```
entrer.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        monApp.pushScreen(new Page(monApp));
    }
});
```

La classe `Accueil` complète

Pour finir, je vous propose donc le code complet de la classe `Accueil`, afin que vous puissiez tester par vous-mêmes :

Code : Java - Accueil.java

```
package mypackage;

import net.rim.device.api.system.Bitmap;
import net.rim.device.api.system.Display;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.Manager;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.BitmapField;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.container.HorizontalFieldManager;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;

public final class Accueil extends MainScreen
{
    private UiApplication monApp;

    public Accueil(UiApplication app)
    {
        super(Manager.NO_VERTICAL_SCROLL);
        monApp = app;
        // Accueil
        HorizontalFieldManager accueil = new
HorizontalFieldManager(Field.USE_ALL_HEIGHT){
            public void paint(Graphics g){
                g.drawGradientFilledRect(0, 0, Display.getWidth(),
Display.getHeight(), 0x009FCFFD, 0x0079B6FC);
                super.paint(g);
            }
        };
        VerticalFieldManager conteneur = new
VerticalFieldManager(Field.FIELD_VCENTER|Field.USE_ALL_WIDTH);
```

```
            Bitmap logo = Bitmap.getBitmapResource("logo_sdz.png");
            BitmapField monBitmap = new
    BitmapField(logo,Field.FIELD_HCENTER);
            ButtonField entrer = new
    ButtonField("ENTRER",Field.FIELD_HCENTER);
            entrer.setChangeListener(new FieldChangeListener() {
                public void fieldChanged(Field field, int context) {
                    monApp.pushScreen(new Page(monApp));
                }
            });
            conteneur.add(monBitmap);
            conteneur.add(entrer);
            accueil.add(conteneur);
            add(accueil);
        }
    }
```

- Pour supprimer l'ensemble des enfants d'un conteneur et pouvoir le mettre à jour, il existe la méthode `deleteAll()`.
- Pour travailler avec différentes vues, il est préférable de créer une classe héritant de `MainScreen` pour chacune d'entre elles.
- Afin de pouvoir passer d'une vue à une autre, il est nécessaire de définir un attribut de type `UiApplication` et d'y stocker une référence de la classe principale.
- La méthode `pushScreen()` permet d'effacer la vue actuelle et de la remplacer par une nouvelle.

L'internationalisation

Plus tard, lorsque vous créerez des applications, vous serez amenés à définir l'étendue d'utilisation de celle-ci. J'entends par là qu'il vous faudra déterminer si votre application est susceptible d'être utilisée à l'étranger. Si c'est le cas, vous devrez alors l'adapter à différentes langues.
Dans ce chapitre, nous allons donc voir comment utiliser des **ressources** et comment s'en servir pour créer des applications multilingues.

Créer des ressources

Préparation du projet

Introduction

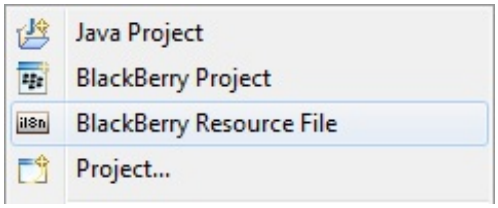
Comme je vous l'ai dit en introduction, pour gérer des applications multilingues nous allons devoir utiliser ce qu'on appelle des **ressources** !

Qu'est-ce qu'une ressource ?

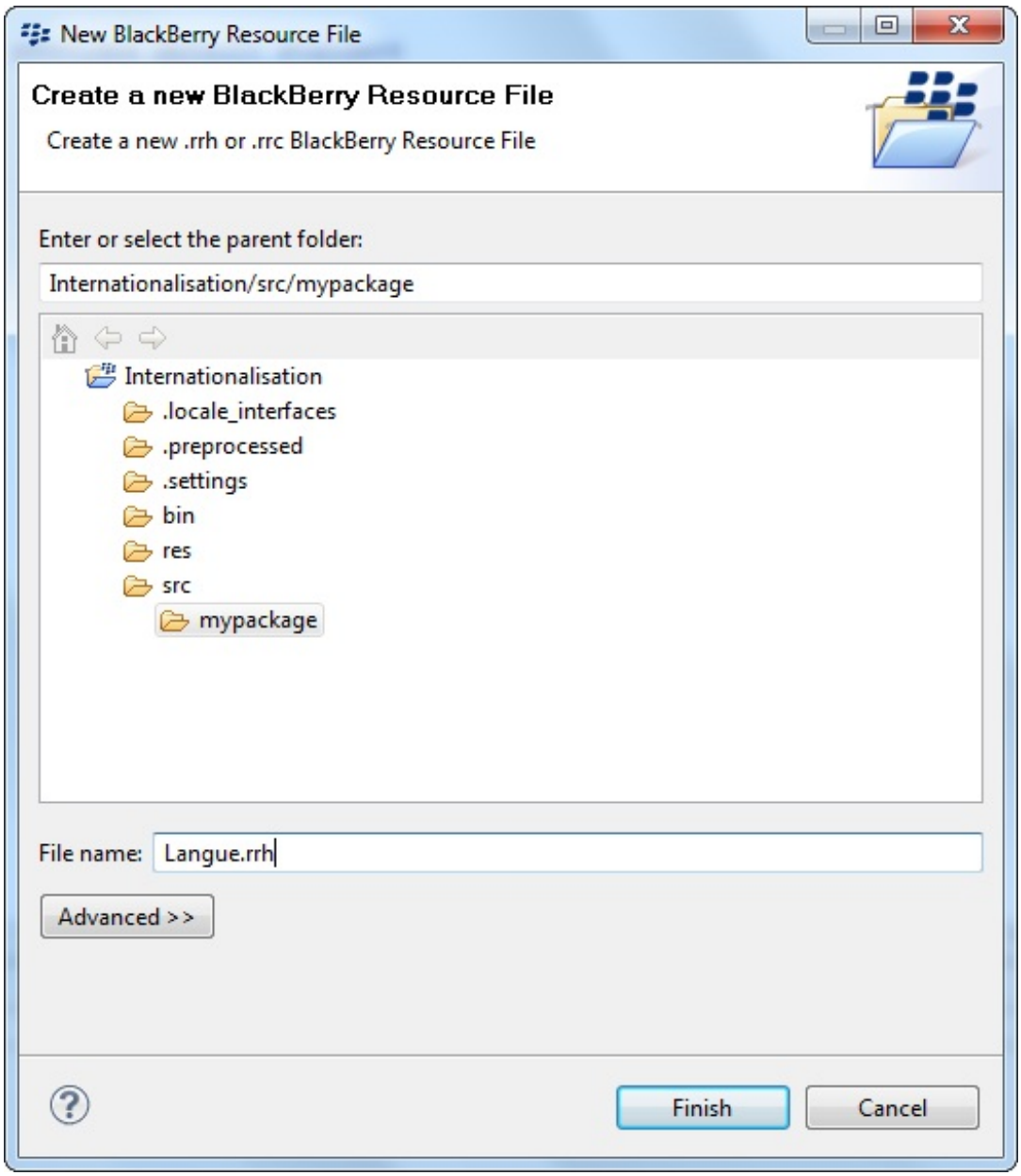
Les ressources sont en fait des fichiers dans lesquels nous allons pouvoir définir nos champs de texte en différentes langues. Le principe est alors de définir différents champs que nous nommerons des **clés**. Celles-ci sont alors associées à diverses chaînes de caractères, une pour chaque langue désirée. L'idée est ensuite non pas d'associer une chaîne de caractères à un objet graphique, mais plutôt une *clé* qui elle contiendra l'ensemble des traductions.
Nous allons donc voir dans ce chapitre comment réaliser tout ceci.

Créer une ressource

Pour créer un fichier de ressources, cliquez sur `File > New > BlackBerry Resource File` :

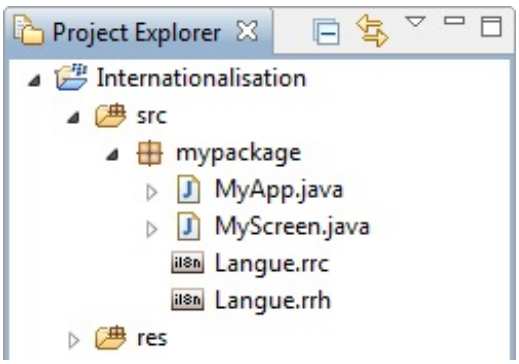


Nous allons donc créer un nouveau fichier de ressources nommé « `Langue.rrh` » à l'intérieur du dossier `mypackage` :



Pour ce chapitre, nous insèrerons les fichiers de ressources dans le même dossier que les sources en elles-mêmes. En revanche lorsque vous développerez des applications plus conséquentes, je vous recommande de les séparer des fichiers sources et de tout bien organiser dans différents dossiers.

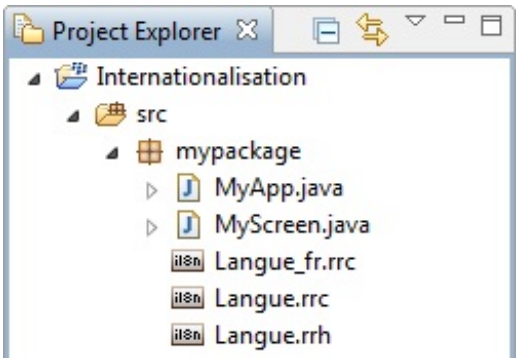
Vous devriez alors voir apparaître deux fichiers dans l'arborescence de votre projet nommés `Langue.rrh` et `Langue.rrc`, comme ci-dessous :



Ajouter une nouvelle ressource

Précédemment, nous n'avons en réalité créé une ressource uniquement pour pour la langue par défaut. C'est pourquoi pour

ajouter une nouvelle langue à notre application, nous allons devoir ajouter un nouveau fichier « `.rrc` ».
Je vous propose donc d'ajouter un fichier pour la langue française nommé `Langue_fr.rrc` :



Contrairement à ce que vous pourriez croire, l'extension « `_fr` » est imposée pour la langue française. Pour savoir quelle extension choisir, je vous invite à jeter un œil à la classe `Locale` du package `net.rim.device.api.i18n` de l'API Reference.

Implémentation des ressources

L'utilisation des ressources est assez spéciale, puisqu'elle consiste à implémenter celles-ci. Le nom de la « pseudo-interface » à implémenter alors est le nom du fichier « `.rrh` » suivi du terme `Resource`.
Ainsi dans notre cas, le nom sera `LangueResource` :

Code : Java

```
public final class MyScreen extends MainScreen implements
LangueResource
```

Une fois implémentée, la ressource doit également être déclarée en tant qu'attribut. Pour faire cela, utilisez la syntaxe suivante :

Code : Java

```
private static ResourceBundle langue =
ResourceBundle.getBundle(BUNDLE_ID, BUNDLE_NAME);
```

Dans la suite, nous accèderons donc aux différents champs de la ressource grâce à cet attribut `langue`.

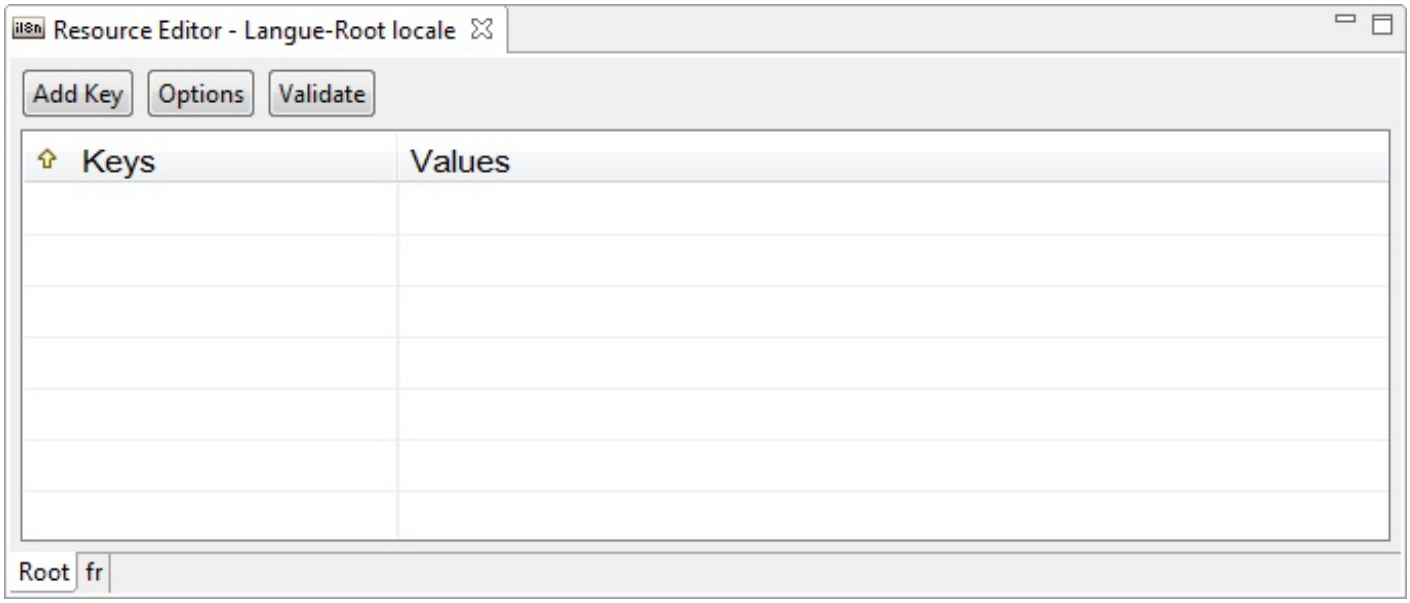
Utiliser les ressources

Travailler les ressources

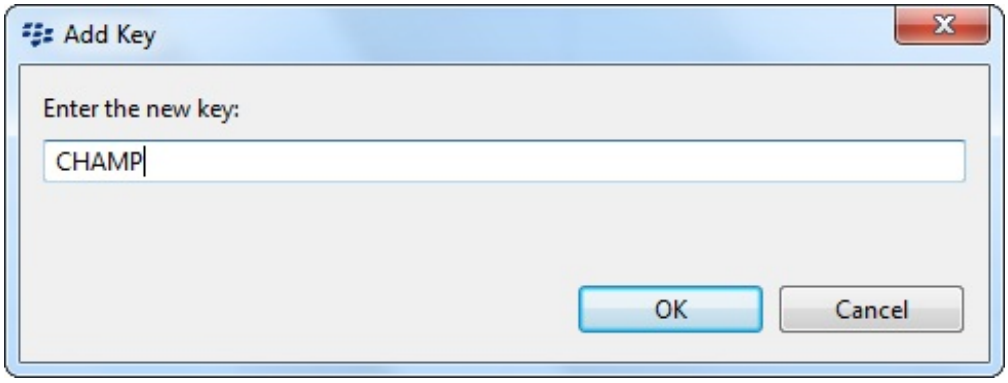
À ce stade, vous devriez normalement avoir les trois fichiers suivants : `Langue.rrh`, `Langue.rrc` et `Langue_fr.rrc`.

Ajouter une clé

Avant toute chose, commencez par ouvrir le fichier `Langue.rrh`. Vous verrez alors apparaître une sorte de tableau comme ci-dessous :

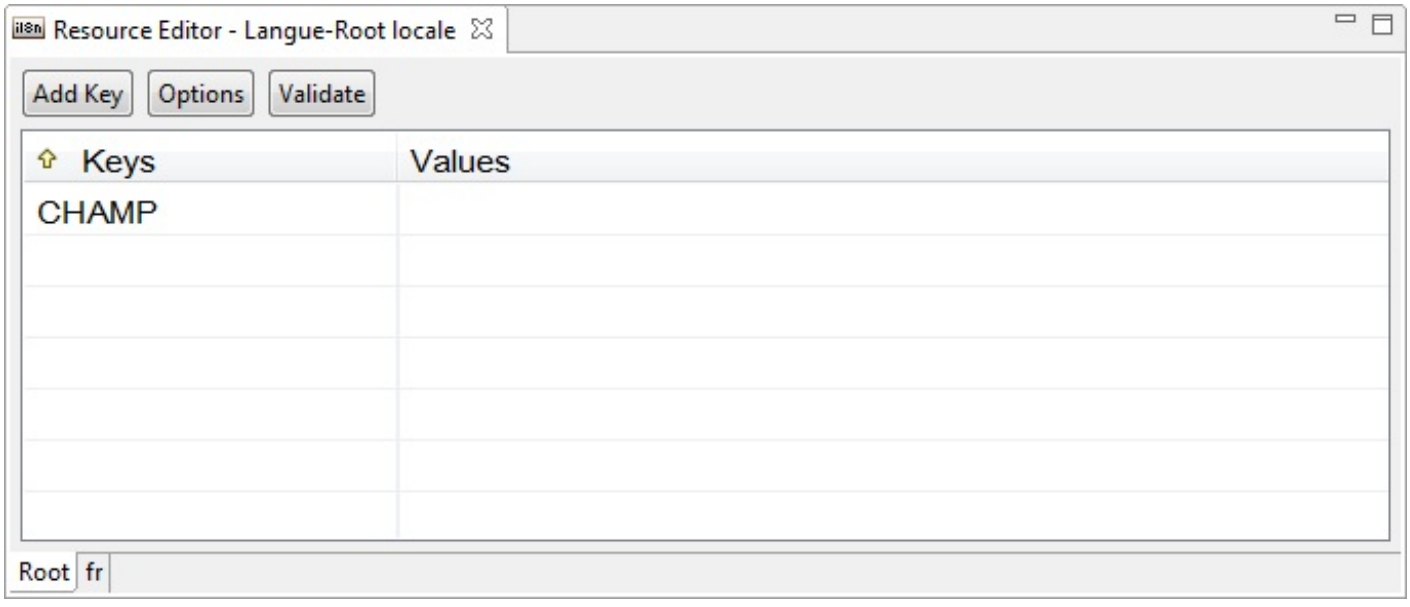


Comme vous pouvez le voir, nous allons ici pouvoir associer des valeurs à des clés. Il nous faut donc commencer par ajouter une nouvelle clé, en cliquant sur `Add Key`. Renseignez alors le nom du champ :



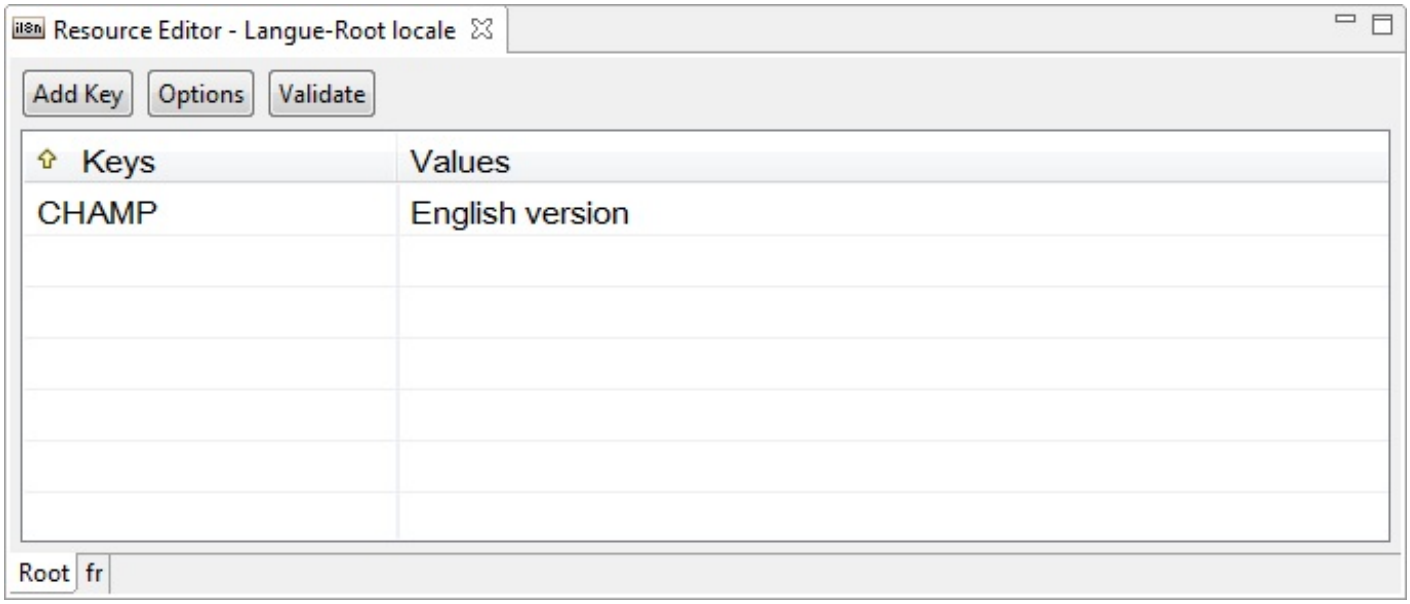
Il est de coutume d'utiliser les ressources comme des constantes de classes. C'est pourquoi je vous recommande d'utiliser les noms en majuscules pour définir vos champs. Cette technique vous permettra également de faire plus facilement la distinction entre les clés et leurs valeurs.

Nous venons donc de créer notre première clé que vous pouvez maintenant voir :



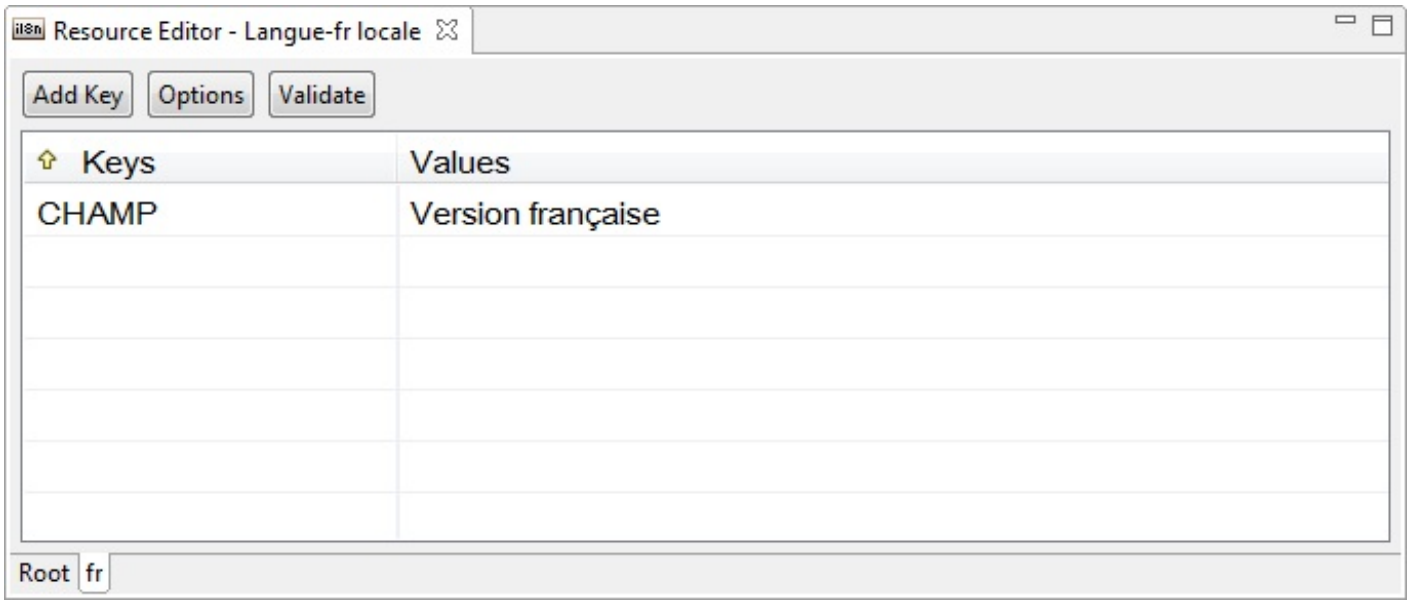
Remplir les champs

À l'ouverture du fichier de ressources, vous tombez normalement sur les ressources de la langue par défaut, c'est-à-dire l'anglais. Vous pouvez donc commencer à remplir les champs en version anglaise dans la section `values`.
Voilà donc ce que nous avons à présent :



Pour passer d'une langue à une autre, il suffit de naviguer dans les différents onglets visibles au bas de la fenêtre d'édition :

Si vous passez dans l'onglet `fr`, vous verrez alors que le champ `values` de `CHAMP` est vide. C'est normal puisqu'il s'agit de la valeur pour la version française, vous pouvez alors renseigner une chaîne de caractères différente de la version anglaise :



Nos fichiers de ressources sont à présent prêts à être utilisés. Nous n'avons ici défini qu'une seule clé, mais vous pouvez en ajouter autant que vous le souhaitez.

Les clés à l'intérieur du code

La partie intéressante maintenant est de pouvoir récupérer les valeurs de nos clés à l'intérieur du code. Pour cela nous allons utiliser la méthode `getString()` de notre objet `langue` de type `ResourceBundle`.
Voici donc comment récupérer la valeur de notre clé `CHAMP` :

Code : Java

```
String monChamp = langue.getString(CHAMP);
```

La valeur sélectionnée lors de l'exécution dépend alors de la langue par défaut du smartphone. Toutefois, il est possible de changer la langue en utilisant l'instruction suivante :

Code : Java

```
Locale.setDefault (Locale.get(Locale.LOCALE_fr, null));
```

Il est alors possible à tout moment de revenir à n'importe quelle langue :

Code : Java

```
Locale.setDefault (Locale.get(Locale.LOCALE_en, null));
```

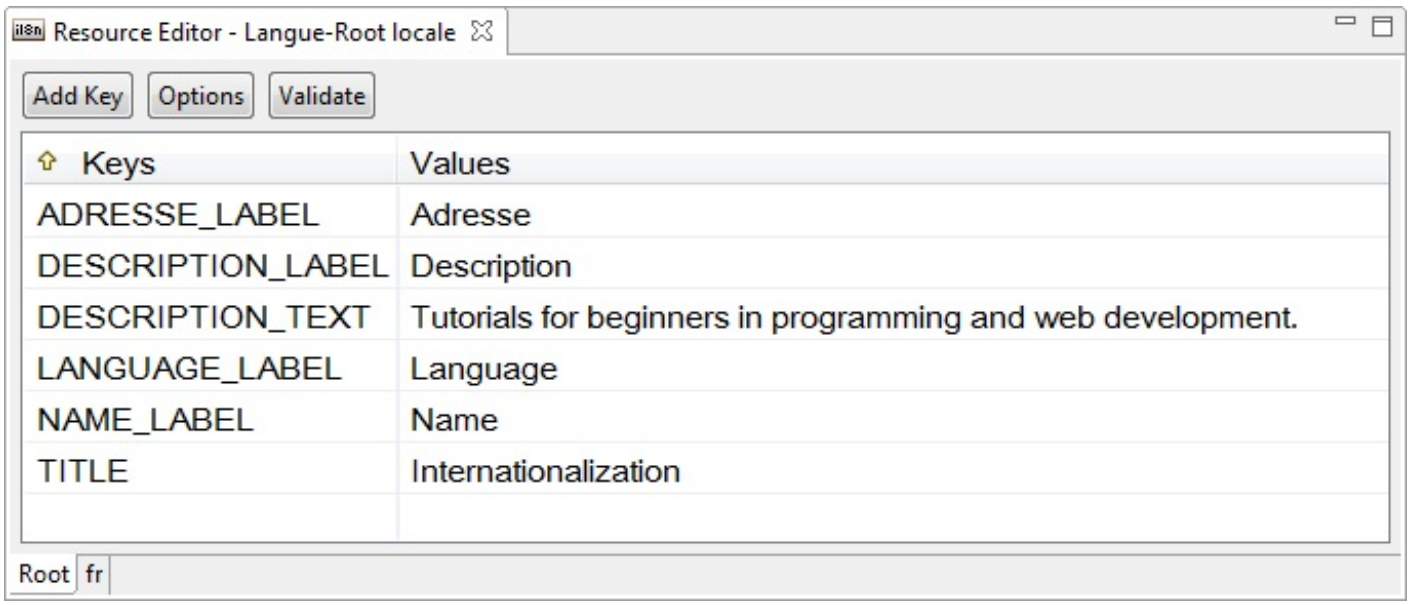
Lorsque vous changez la langue de votre application, il est préférable de mettre à jour les champs manuellement en redéfinissant chacun d'entre eux. Pour vous familiariser avec tout ceci, je vous propose de découvrir un exemple d'application ci-dessous.

Une application multilingue

Préparation des ressources

Les chaînes de caractères anglaises

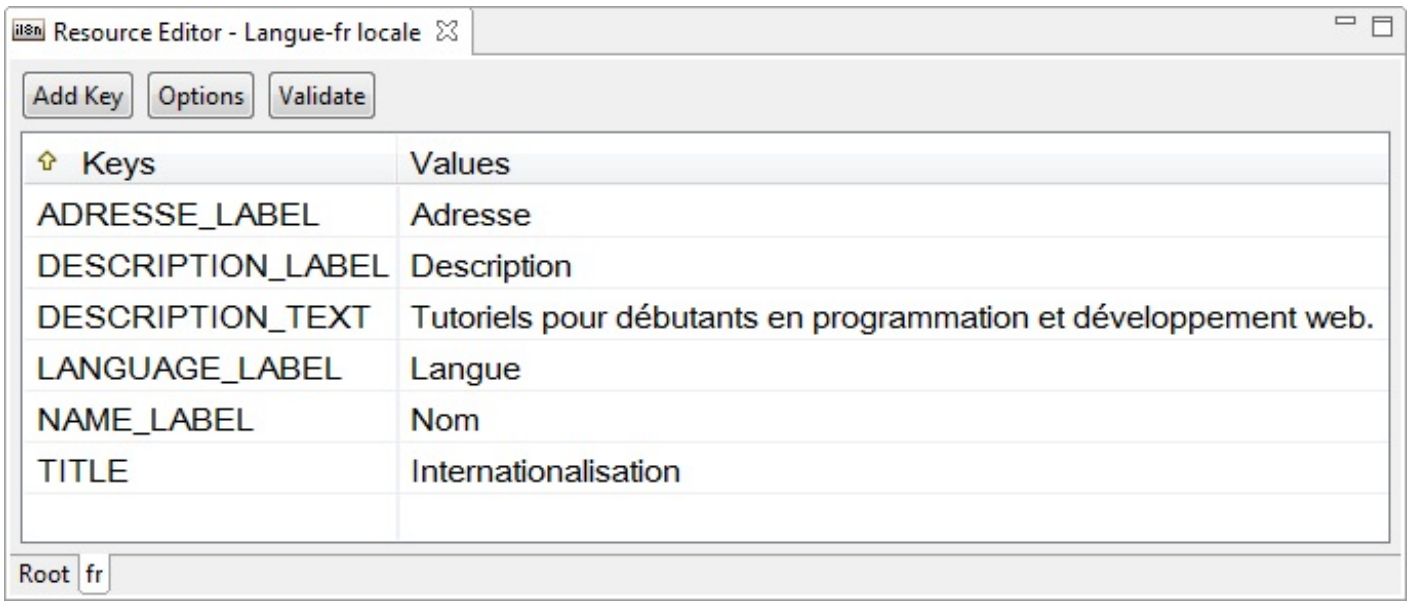
Dans l'exemple d'application que nous allons réaliser, nous réutiliserons les ressources définies plus haut. Nous aurons donc deux langues à savoir : l'anglais et le français.
Au lieu de faire un long discours, je vous laisse découvrir les différentes clés ajoutées ainsi que leurs valeurs anglaises associées :



Ressources anglaises

Les chaînes de caractères françaises

Pour la langue française, nous retrouverons donc les mêmes clés. Nous allons donc uniquement changer leur valeur comme suit :



Ressources françaises

Les sources

Le code utilisé dans cette application n'est pas très complexe, c'est pourquoi je vous propose directement l'intégralité de celui-ci. Les points qui méritent votre attention sont l'*affectation des ressources* et la *gestion des évènements liés à la liste déroulante*. Voici donc le code de cette application :

Code : Java

```
package mypackage;

import net.rim.device.api.i18n.Locale;
import net.rim.device.api.i18n.ResourceBundle;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.component.ObjectChoiceField;
import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.component.SeparatorField;
import net.rim.device.api.ui.container.HorizontalFieldManager;
import net.rim.device.api.ui.container.MainScreen;

public final class MyScreen extends MainScreen implements
LangueResource
{
    private static ResourceBundle langue =
ResourceBundle.getBundle(BUNDLE_ID, BUNDLE_NAME);
    private ObjectChoiceField maListe;
    private LabelField nomLabel;
    private RichTextField nomTexte;
    private LabelField descriptionLabel;
    private RichTextField descriptionTexte;
    private LabelField adresseLabel;
    private RichTextField adresseTexte;

    public MyScreen()
    {
        // Affectation des ressources
        setTitle(langue.getString(TITLE));
        nomLabel = new LabelField(langue.getString(NAME_LABEL) + " :
");
        nomTexte = new RichTextField("Site du Zéro");
        descriptionLabel = new
LabelField(langue.getString(DESCRIPTION_LABEL) + " : ");
        descriptionTexte = new
```

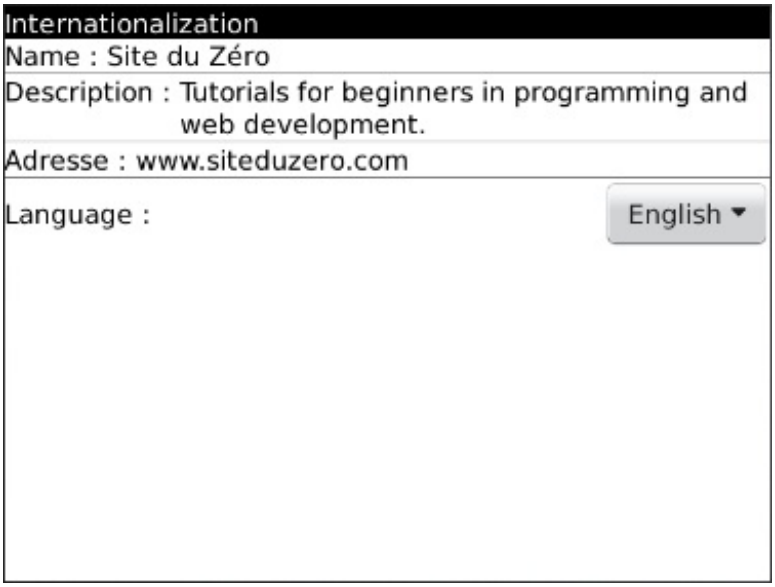
```
RichTextField(langue.getString(DESCRIPTION_TEXT));
        adresseLabel = new
LabelField(langue.getString(ADRESSE_LABEL) + " : ");
        adresseTexte = new RichTextField("www.siteduzero.com");

        // Création de la liste déroulante
        String mesLangues[] = {"English","Français"};
        maListe = new
ObjectChoiceField(langue.getString(LANGUAGE_LABEL) + " :
",mesLangues,0);

        // Création des conteneurs
        HorizontalFieldManager nomManager = new
HorizontalFieldManager();
        HorizontalFieldManager descriptionManager = new
HorizontalFieldManager();
        HorizontalFieldManager adresseManager = new
HorizontalFieldManager();

        // Mise en place des composants
        nomManager.add(nomLabel);
        nomManager.add(nomTexte);
        descriptionManager.add(descriptionLabel);
        descriptionManager.add(descriptionTexte);
        adresseManager.add(adresseLabel);
        adresseManager.add(adresseTexte);

        // Gestion de l'affichage final
        add(nomManager);
        add(new SeparatorField());
        add(descriptionManager);
        add(new SeparatorField());
        add(adresseManager);
        add(new SeparatorField());
        add(maListe);

        // Gestion des évènements liés à la liste déroulante
        maListe.setChangeListener(new FieldChangeListener() {
            public void fieldChanged(Field field, int context) {
                if(maListe.getChoice(maListe.getSelectedIndex()) ==
"English"){
                    Locale.setDefault (Locale.get(Locale.LOCALE_en,
null));
                }else{
                    Locale.setDefault (Locale.get(Locale.LOCALE_fr,
null));
                }
                // Mise à jour des champs
                setTitle(langue.getString(TITLE));
                nomLabel.setText((langue.getString(NAME_LABEL) + " :
"));
                nomTexte.setText(("Site du Zéro"));

descriptionLabel.setText((langue.getString(DESCRIPTION_LABEL) + " :
"));

descriptionTexte.setText((langue.getString(DESCRIPTION_TEXT)));

adresseLabel.setText((langue.getString(ADRESSE_LABEL) + " : "));
                adresseTexte.setText("www.siteduzero.com");
                maListe.setLabel(langue.getString(LANGUAGE_LABEL) +
" :");
            }
        });
    }
}
```

Résultat final

Nous allons enfin pouvoir admirer le résultat de ces manipulations de ressources. Lancez donc l'application réalisée !

Version anglaise

Étant donné que par défaut votre simulateur est en anglais, vous devriez donc tomber sur l'écran suivant :



Version anglaise

Version française

Si vous avez analysé le code précédent, vous devez savoir que pour changer la langue il suffit de mettre à jour la liste déroulante.
Je vous invite donc à passer sur la valeur « Fançais ».
Vous verrez alors l'ensemble des champs se mettre à jour :

Version française

- Pour internationaliser son application, il est nécessaire d'utiliser des **ressources**.
- Les ressources sont composées d'un fichier `.rrh` et de fichiers `.rrc`, dont le nombre dépend des langues que vous souhaitez intégrer.
- À l'intérieur des ressources, nous définissons des **clés** associées à une **valeur** pour chacune des langues.
- L'utilisation des ressources se fait par *implémentation* de celles-ci.
- Les classes `ResourceBundle` et `Locale` permettent de se servir des ressources à l'intérieur du code.

Le stockage de données

À présent nous allons apprendre à *stocker des données* à l'intérieur du smartphone. Bien sûr, il est possible de stocker tout type de données. Toutefois dans ce cours, nous nous contenterons de gérer des fichiers de type texte avec l'extension « `.txt` ».
Nous verrons donc comment créer un fichier, puis comment y écrire dedans et lire son contenu.
Soyez attentif au cours de ce chapitre, car les notions vues ici vous seront grandement utiles dans le chapitre suivant qui est un TP !

Gérer des données

Introduction au stockage

Comme dit en introduction, nous allons dans ce chapitre nous intéresser au stockage de données. Contrairement aux variables qui sont stockées dans la **mémoire vive**, les données seront ici stockées sur le disque dur du smartphone ou éventuellement sur une **carte mémoire externe**. Dans le second cas, il s'agit d'une carte SD que vous pouvez insérer dans l'appareil.

Afin d'éviter de s'embêter plus tard avec les chemins absolus correspondant au disque dur et à la carte externe, je suggère que nous créions deux *constantes* : `DEVICE` et `SD_CARD`. Ainsi nous n'aurons plus à nous préoccuper des chemins depuis la « racine », et surtout cela nous évitera de faire des erreurs.
Voici donc ces constantes :

Code : Java

```
public static final String SD_CARD = "file:///SDCard/";
public static final String DEVICE =
"file:///store/home/user/documents/";
```

Ainsi ces constantes seront accessibles par les expressions `MyScreen.SD_CARD` et `MyScreen.DEVICE`.

L'interface `FileConnection`

Une histoire de connexion

Quand nous parlons de stockage de données, nous faisons généralement allusion à l'écriture ou la lecture d'un fichier. Pour réaliser ces actions, nous faisons en réalité une **connexion** à ce fichier. Une fois la connexion établie, il est ensuite possible de réaliser toutes sortes de modifications sur ce fichier. Une fois celles-ci terminées, il est alors nécessaire de fermer cette connexion afin de « casser » le lien avec le fichier.

En ce qui nous concerne, la connexion à un fichier, ou à un dossier, se réalise à l'aide de l'interface `FileConnection`. Une fois la connexion réalisée, cette interface permet également de manipuler le fichier ou dossier correspondant grâce à ses méthodes.
Avant d'aller plus loin, je vous propose de découvrir l'instruction permettant de réaliser cette liaison :

Code : Java

```
FileConnection fc = (FileConnection)Connector.open(/* Chemin */);
```

Étant donné que la connexion à un fichier ou à un dossier peut échouer et donc générer des erreurs, nous devrons réaliser l'ensemble des opérations à l'intérieur d'un bloc `try{....} catch{...}`.
Voici donc la structure de base pour utiliser des données externes au programme :

Code : Java

```
try
{
    FileConnection fc = (FileConnection)Connector.open(/* Chemin
*/);
    // Manipulation de la donnée
    fc.close();
}
catch (IOException ioe)
{
    System.out.println(ioe.getMessage());
```

```
}
```

Vous remarquerez que j'ai utilisé la méthode `close()` de l'interface `FileConnection` pour fermer la connexion à la donnée. Cette instruction est *très importante* et *ne doit pas* être oubliée lorsque vous échangez avec l'extérieur d'une application.

Manipulation d'une donnée

Maintenant que vous savez comment réaliser une connexion à un fichier ou dossier, nous allons pouvoir commencer à travailler avec les différentes méthodes de l'interface `FileConnection`.
Pour démarrer, je vais vous introduire la méthode `exists()` que nous utiliserons maintenant à chaque fois. En effet, il est probable que la donnée que vous voulez atteindre n'existe tout simplement pas. C'est pourquoi à partir de maintenant, nous allons devoir vérifier à *chaque fois* si la donnée en question existe. Dans le cas contraire, nous la créerons !
Sachant que la méthode `exists()` renvoie une valeur booléenne, nous pouvons directement l'insérer dans une condition :

Code : Java

```
try
{
    FileConnection fc = (FileConnection)Connector.open(systeme +
dossier + "/");
    if (!fc.exists())
    {
        // Créer la donnée
    }
    fc.close();
}
catch (IOException ioe)
{
    System.out.println(ioe.getMessage());
}
```

À l'intérieur des accolades du **`if`**, nous allons donc créer notre donnée. Il peut donc s'agir d'un dossier ou d'un fichier. Comme vous l'imaginez, l'instruction n'est pas la même dans les deux cas.
Tout d'abord, voici comment créer un dossier à l'aide de la méthode `mkdir()` :

Code : Java

```
fc.mkdir();
```

Pour créer un fichier, l'instruction est tout aussi simple grâce à la méthode `create()`. Voici comment faire :

Code : Java

```
fc.create();
```

Une autre méthode qui pourrait vous être utile est `delete()`, qui permet de supprimer un fichier comme un dossier. Néanmoins celle-ci suppose que vous ayez la donnée correspondante. C'est pourquoi elle doit s'utiliser avec la condition suivante :

Code : Java

```
if (fc.exists())
{
    fc.delete();
}
```

Notez bien la différence avec la condition précédente. Ici, l'expression est `fc.exists()` sans « ! », car le fichier doit *obligatoirement* exister.

Création de données

Maintenant que vous possédez toutes les bases pour gérer des données, je vais simplement vous présenter des méthodes toutes faites.

Créer un dossier

Pour commencer, voici une méthode que j'ai nommée `creerDossier()` et qui bien évidemment permet de créer un dossier :

Code : Java

```
public void creerDossier(String dossier, String systeme){
    try
    {
        FileConnection fc = (FileConnection)Connector.open(systeme +
dossier);
        if (!fc.exists())
        {
            fc.mkdir();
        }
        fc.close();
    }
    catch (IOException ioe)
    {
        System.out.println(ioe.getMessage() );
    }
}
```

Rien de nouveau ici, si ce n'est la construction par « segment » du chemin absolu du dossier.
Voici donc un exemple d'utilisation de cette méthode :

Code : Java

```
creerDossier("SdZ/", MyScreen.SD_CARD);
```

Pour pouvoir se connecter à un dossier, le caractère « / » de séparation des dossiers doit être présent à la fin du nom du dossier. Pensez donc bien à le mettre lors de la saisie du chemin de votre dossier.

Créer un fichier

Cette méthode `creerFichier()` est identique à la précédente mise à part la création à l'aide de `create()` :

Code : Java

```
public void creerFichier(String fichier, String systeme){
    try
    {
        FileConnection fc = (FileConnection)Connector.open(systeme +
fichier);
        if (!fc.exists())
        {
            fc.create();
        }
        fc.close();
    }
    catch (IOException ioe)
```

```
        {
            System.out.println(ioe.getMessage() );
        }
    }
```

Voici comment l'utiliser :

Code : Java

```
creerFichier("fichier.txt", MyScreen.DEVICE);
```

Supprimer une donnée

Comme nous l'avons dit, l'opération pour supprimer un fichier ou un dossier est strictement la même. C'est pourquoi nous ne créerons qu'une méthode permettant de supprimer tout type de données.
Voici cette méthode nommée `supprimerDonnee()` :

Code : Java

```
public void supprimerDonnee(String donnee, String systeme){
    try
    {
        FileConnection fc = (FileConnection)Connector.open(systeme +
donnee);
        if (fc.exists())
        {
            fc.delete();
        }
        fc.close();
    }
    catch (IOException ioe)
    {
        System.out.println(ioe.getMessage() );
    }
}
```

Grâce à cette méthode nous pouvons donc supprimer notre fichier `fichier.txt`, comme cela :

Code : Java

```
supprimerDonnee("fichier.txt", MyScreen.DEVICE);
```

Puis voici comment supprimer notre dossier `SdZ` créé plus haut :

Code : Java

```
supprimerDonnee("SdZ/", MyScreen.SD_CARD);
```

Accéder « physiquement » aux données

Précédemment nous avons appris à créer et supprimer des fichiers comme des dossiers. Toutefois, la seule manière de bien visualiser ces manipulations est d'aller « fouiller » directement dans les fichiers du smartphone. Nous allons donc maintenant voir comment récupérer ces données depuis l'extérieur de notre application.

Stockage à l'intérieur du smartphone

Dans un premier temps, nous allons créer un nouveau dossier. Pour cela, nous utiliserons donc notre méthode

`creerDossier()` comme ceci :

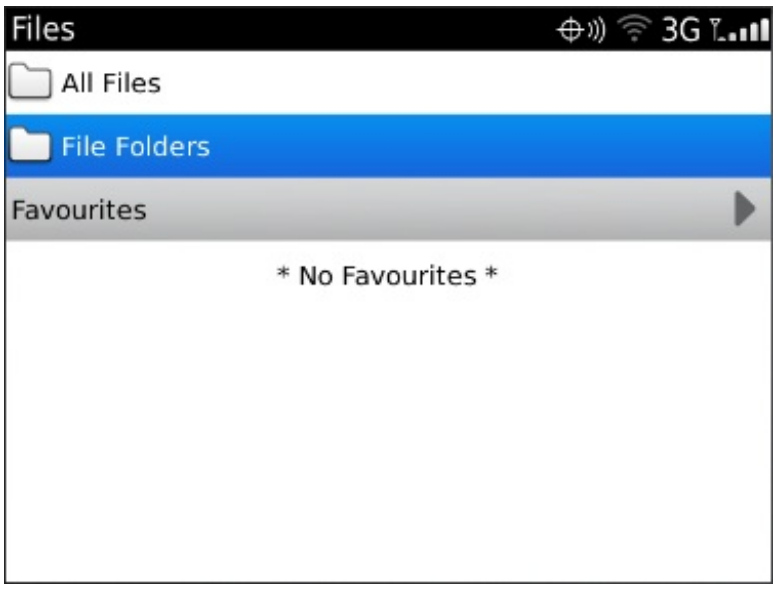
Code : Java

```
creerDossier("SdZ/", MyScreen.DEVICE);
```

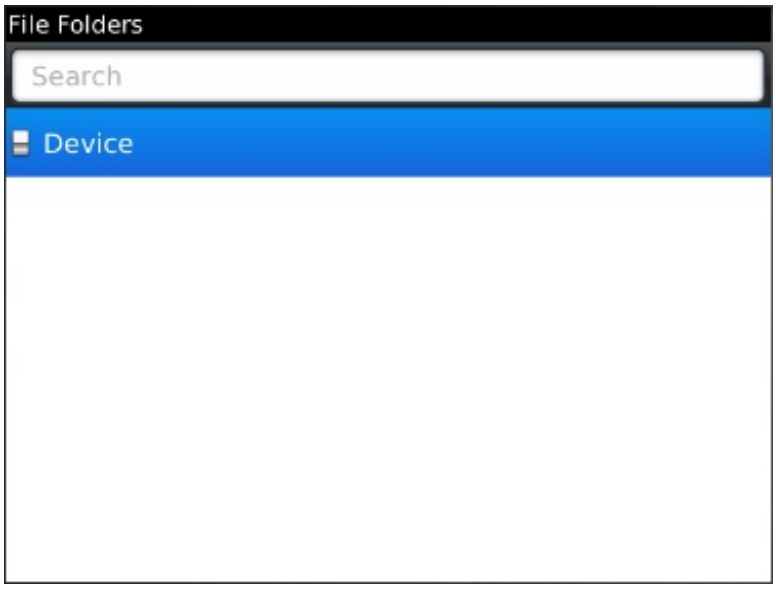
Après exécution, le dossier `SdZ` est normalement créé. Pour vérifier, nous allons entrer dans les dossiers du disque dur du smartphone. Pour faire ça, commencez par cliquer sur l'icône `Applications` comme ci-dessous :

Ensuite, entrez dans l'explorateur de fichiers représenté par l'icône sélectionnée ci-dessous :

Puis rendez-vous dans l'explorateur de dossiers par hiérarchie en cliquant sur `File Folders` comme présenté sur l'image suivante :



N'ayant pas encore inséré de carte mémoire, nous n'avons pour l'instant accès uniquement au smartphone. N'ayant pas le choix, cliquez donc sur `Device` :

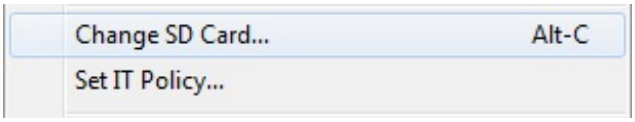


Maintenant suivez le cheminement que nous avons réalisé, c'est-à-dire `home/user/documents/`. Arrivé à l'intérieur du répertoire `documents`, vous devriez alors trouver notre dossier `SdZ`.
Voilà ce que j'obtiens :

Stockage sur une carte externe

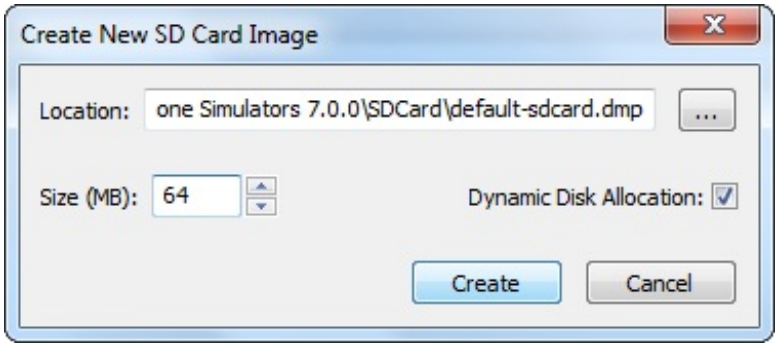
Créer une carte mémoire

Comme nous avons vu plus haut, il est possible de stocker des données à l'intérieur d'une carte mémoire externe. En revanche pour que cela soit réalisable, il faut bien entendu que le smartphone dispose d'une carte mémoire. Nous allons donc commencer par ajouter une carte mémoire à notre simulateur.
Pour cela, lancez votre simulateur, puis allez dans `Change SD Card...` du menu `Simulate` :

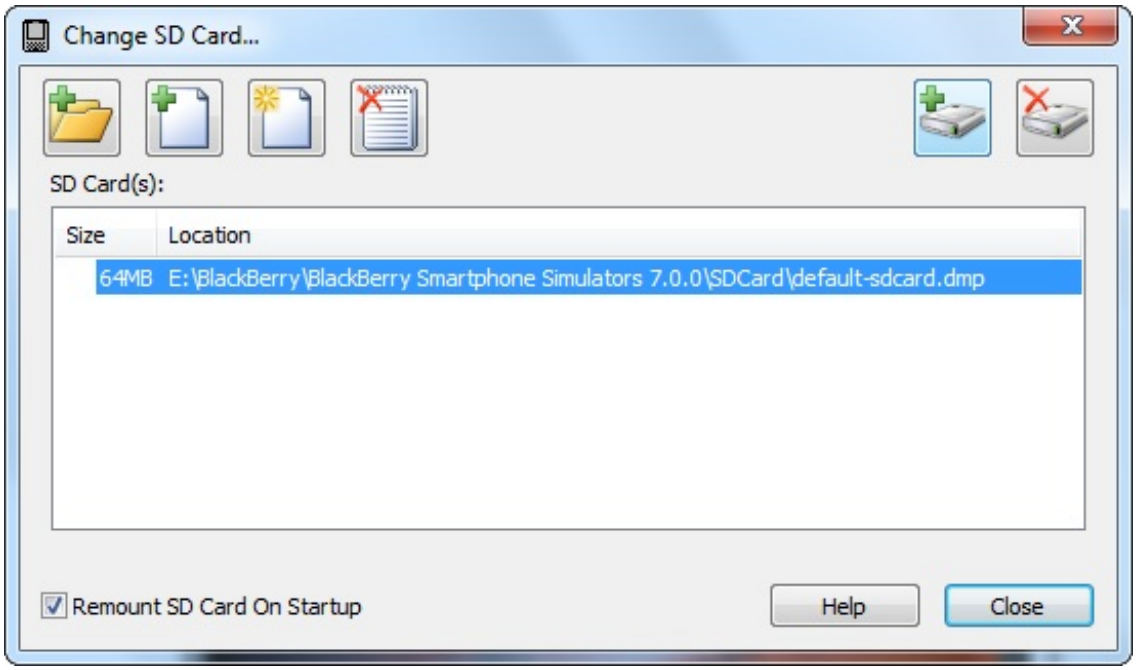


Vous arrivez alors sur le gestionnaire de cartes mémoires. Pour en créer une nouvelle, cliquez sur le bouton suivant : 

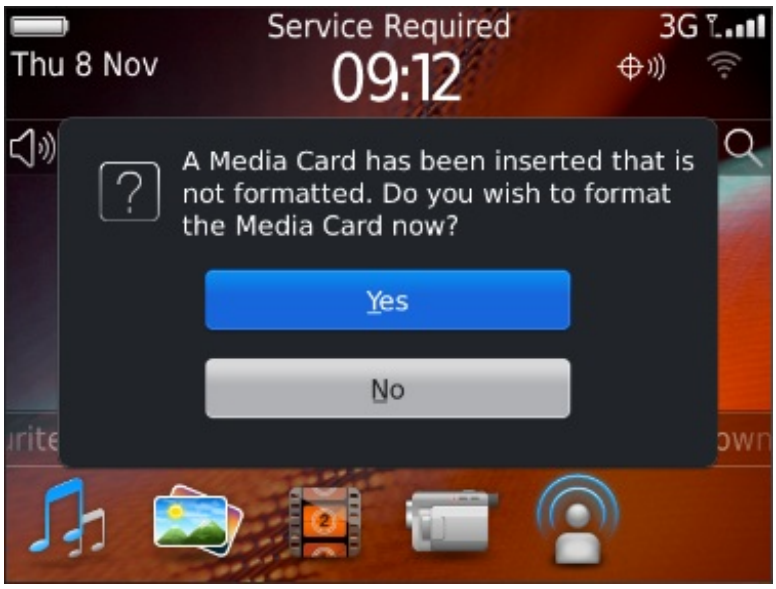
Maintenant nous allons pouvoir paramétrer notre carte. Ainsi, renseignez un emplacement où stocker votre carte sur votre ordinateur puis spécifiez la taille de la mémoire :



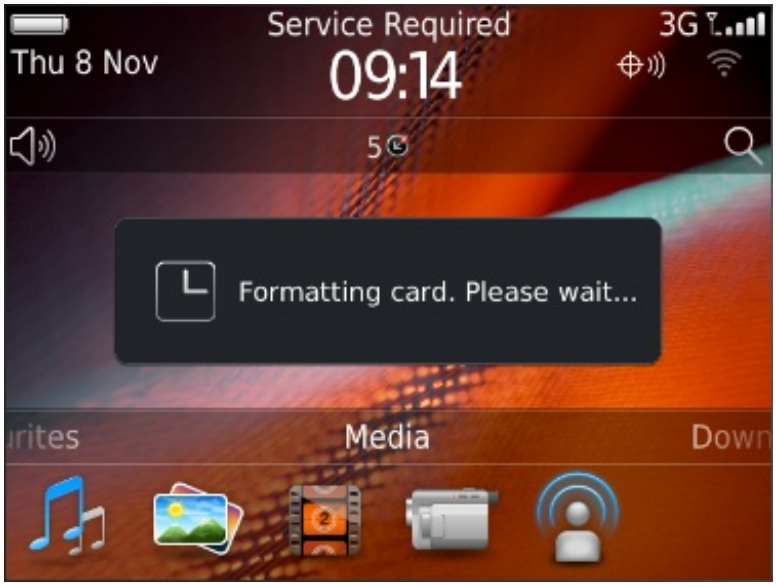
Une fois le bouton `Create` cliqué, vous verrez apparaître votre carte dans le gestionnaire. Cependant celle-ci n'est pas encore insérée dans le smartphone. C'est pourquoi vous allez cocher la case « `Remount SD Card On Startup` » et cliquer sur le bouton `Mount selected` en haut à droite de la fenêtre :



Une fois la fenêtre fermée, il est probable qu'on vous demande de formater votre carte mémoire. Acceptez en cliquant sur `Yes` :



Attendez durant le formatage de celle-ci. Une fois terminé, votre carte est prête à être utilisée :



Accéder au dossier de la carte mémoire

Maintenant que notre carte est insérée dans l'appareil, nous allons pouvoir y stocker des données. Créons donc un dossier grâce à l'instruction suivante :

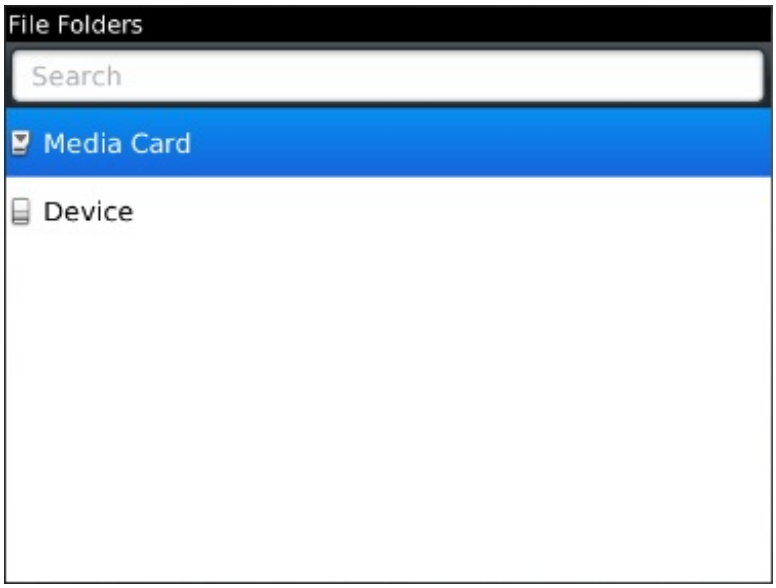
Code : Java

```
creerDossier("SdZ", MyScreen.SD_CARD);
```

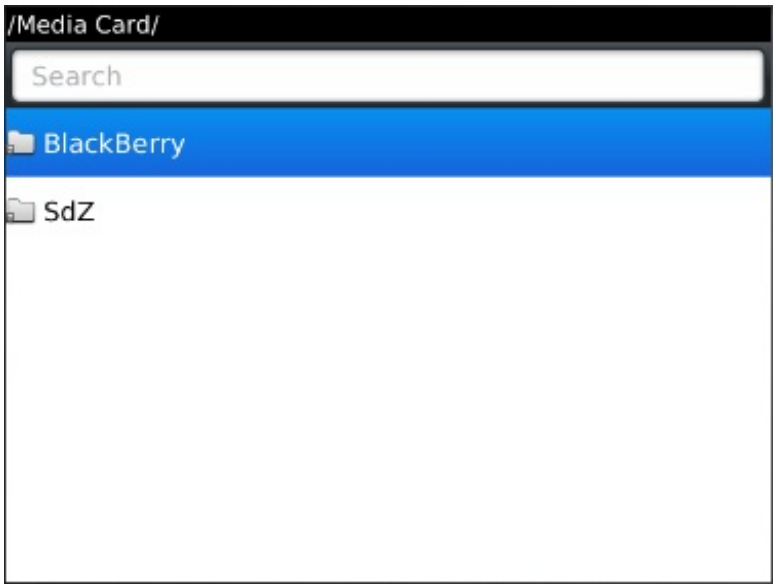
Pour visualiser notre nouveau dossier, nous devons retourner dans l'explorateur de fichiers. Une autre manière d'y accéder est de se placer sur la catégorie `Media`, puis d'appuyer sur la touche `menu` de votre smartphone : 

Dans le menu qui apparaît, cliquez sur `Explore` :

Dans `File Folders`, vous devriez normalement voir une nouvelle entrée : `Media Card`. En cliquant dessus, vous entrez donc dans la carte mémoire externe :



Comme nous nous y attendions, un dossier `SdZ` est présent :



Un fichier de test

Juste par curiosité, nous allons créer cette fois un fichier et non pas un dossier. Nous allons donc réutiliser la méthode `creerFichier()` que nous avions auparavant réalisée :

Code : Java

```
creerFichier("fichier.txt", MyScreen.DEVICE);
```

Sans grande surprise, nous trouvons notre fichier nommé `fichier.txt` à l'emplacement prévu :

Lire et écrire dans un fichier

Un système binaire

Un peu de théorie

Avant de nous lancer dans la suite, revenons un peu sur de la théorie, en particulier sur le **langage binaire**.
Pour illustrer ce qui va être dit, nous allons prendre comme comparaison notre belle langue natale, le français !
La base de notre langage, c'est la lettre. Nous avons donc un alphabet de 26 lettres, grâce auquel nous pouvons créer des mots et faire des phrases. Ainsi, à l'aide de seulement 26 lettres (ou symboles), nous sommes capables de réaliser des millions et des millions de phrases et dire tout ce qu'on veut. Un appareil électronique fonctionne de la même manière.

0001100100111000111000101101001110001110 00

Ci-dessus, vous pouvez voir un signal électrique tel qu'on peut en trouver dans tout appareil électronique. On définit alors un 1 « logique » et un 0 « logique » pour représenter les états haut et bas du signal. On obtient donc notre base du *langage binaire* (binaire pour deux symboles). Et de la même manière que les phrases, on peut réaliser des instructions grâce à une succession de 0 et de 1. Voilà le seul langage que les appareils électroniques comprennent !

Pour en revenir à notre problème, le stockage de données se fait également en binaire. Ainsi les données sont enregistrées sous la forme de *mots binaires* appelés **bytes**. C'est pourquoi lorsque nous voudrons enregistrer ou lire du texte, nous allons devoir faire la conversion en bytes.

La classe `String`

En fait la conversion d'une chaîne de caractères en bytes est appelée l'**encodage** !
Il existe plusieurs manières d'encoder du texte, et le système d'exploitation BlackBerry OS en prend en charge plusieurs, à savoir :

- ISO-8859-1
- UTF-8
- UTF-16BE
- US-ASCII.

Ainsi la classe `String` dispose de méthodes qui permettent de faire l'encodage suivant ces différentes normes. La méthode `getBytes()` par exemple sert à encoder un texte.
Voici donc comment convertir une chaîne de caractères nommée `monTexte` de type `String` en bytes :

Code : Java

```
monTexte.getBytes("UTF-8");
```

En réalité, en Java il existe un type `byte` représentant un mot binaire. Ainsi l'instruction précédente renvoie en fait un tableau de type `byte[]`.
L'opération inverse est bien évidemment réalisable. Pour cela, nous pouvons utiliser directement le constructeur de la classe `String`. Voici par exemple comment retrouver une chaîne de caractères à partir d'une variable `donnees` de type `byte[]` :

Code : Java

```
monTexte = new String(donnees, "UTF-8");
```

Sachant que le système d'exploitation dispose déjà d'un encodage par défaut (à savoir ISO-8859-1), il peut être facultatif de renseigner l'encodage lors de l'appel des méthodes `getBytes()` et `String()`.

Écriture dans un fichier

La classe `OutputStream`

L'interface `FileConnection` ne permet pas de lire ou d'écrire dans un fichier. C'est pourquoi nous sommes obligés de passer par des classes « annexes » pour le faire. En ce qui concerne l'écriture, nous disposons de la classe abstraite `OutputStream`.
Pour déclarer et initialiser une variable de type `OutputStream`, nous devons alors passer par la méthode `openOutputStream()` de l'interface `FileConnection` :

Code : Java

```
OutputStream os = fc.openOutputStream();
```

Une fois réalisé, nous pouvons enfin écrire à l'intérieur de ce fichier grâce à la méthode `write()`. Celle-ci prend alors en paramètre les données à stocker sous la forme d'un tableau `byte[]`. Nous aurons ainsi besoin ici d'utiliser la méthode `getBytes()` présentée plus haut.
Voici donc comment insérer une chaîne de caractères nommée `monTexte` dans le fichier :

Code : Java

```
os.write(monTexte.getBytes());
```

Enfin pour finir, il est nécessaire de fermer la connexion comme pour l'interface `FileConnection` :

Code : Java

```
os.close();
```

Une méthode complète

Pour vous simplifier les choses dans l'avenir, je vous propose donc une méthode `ecrireFichier()` qui contient tout le code pour pouvoir écrire un texte nommé `donnees` dans un fichier.
Voici le code de cette méthode :

Code : Java

```
public void ecrireFichier(String fichier, String systeme, String
donnees){
    try
    {
        FileConnection fc = (FileConnection)Connector.open(systeme +
fichier);
        if (fc.exists())
        {
            fc.delete();
        }
        fc.create();
        OutputStream os = fc.openOutputStream();
        os.write(donnees.getBytes());
        os.close();
        fc.close();
    }
    catch (IOException ioe)
    {
        System.out.println(ioe.getMessage() );
    }
}
```

Vous remarquerez qu'à l'intérieur de cette méthode, nous effaçons le fichier avant d'y écrire dedans. Ceci s'explique par le fait que l'enregistrement d'un fichier n'affecte que les bytes concernés par le nouveau fichier. Ainsi si votre fichier était auparavant plus long, vous aurez des « résidus » de caractères de votre ancienne version à la fin du fichier.

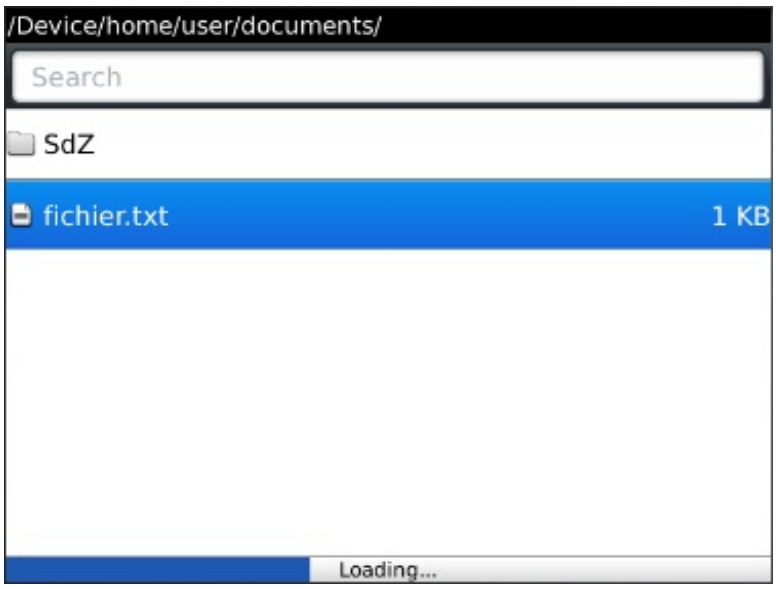
Voici également comment utiliser cette méthode pour écrire à l'intérieur du fichier `fichier.txt` créé tout à l'heure :

Code : Java

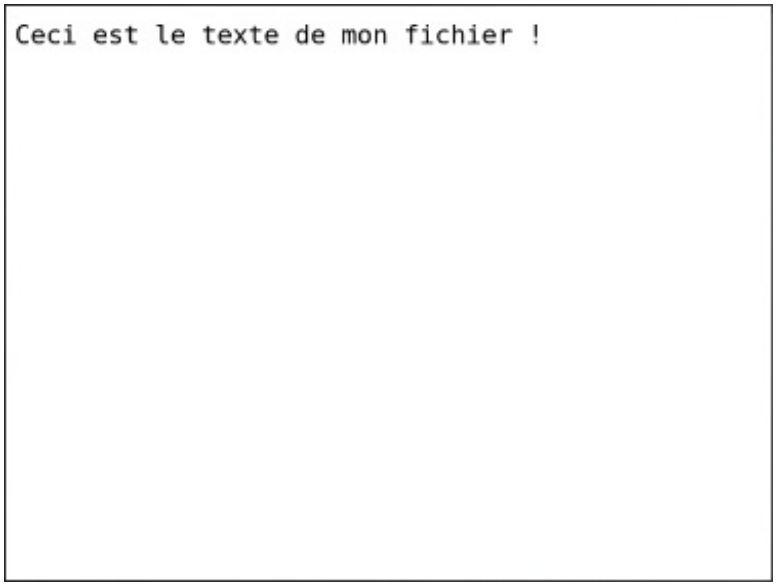
```
String monTexte = "Ceci est le texte de mon fichier !";
ecrireFichier("fichier.txt", MyScreen.DEVICE, monTexte);
```

Ouverture du fichier

Le meilleur moyen de vérifier si le code précédent marche est encore d'ouvrir le fichier en question. Pour faire ça, rendez-vous dans l'explorateur de fichier, puis cliquez sur `fichier.txt` pour l'ouvrir :



Une fois le fichier chargé, vous devriez normalement retrouver la chaîne de caractères écrite plus tôt :



Lecture d'un fichier

La classe `InputStream`

Identiquement à `OutputStream`, nous disposons de la classe abstraite `InputStream` pour lire le contenu d'un fichier.
La déclaration d'une variable de ce type est donc très similaire :

Code : Java

```
InputStream is = fc.openInputStream();
```

La récupération du contenu du fichier se fait de manière assez spéciale. En effet pour cela il faut utiliser la méthode `streamToBytes()` de la classe `IOUtilities`. Celle-ci renvoie alors une variable de type `byte[]` que nous pourrons décrypter par la suite.
Voici comment ça se passe :

Code : Java

```
byte[] data = IOUtilities.streamToBytes(is);
```

Enfin, il faut bien évidemment fermer ceci :

Code : Java

```
is.close();
```

Une méthode complète

Comme d'habitude, je vais maintenant vous fournir une méthode tout prête à l'emploi.
Voici donc le code de cette méthode que j'ai appelée `lireFichier()` :

Code : Java

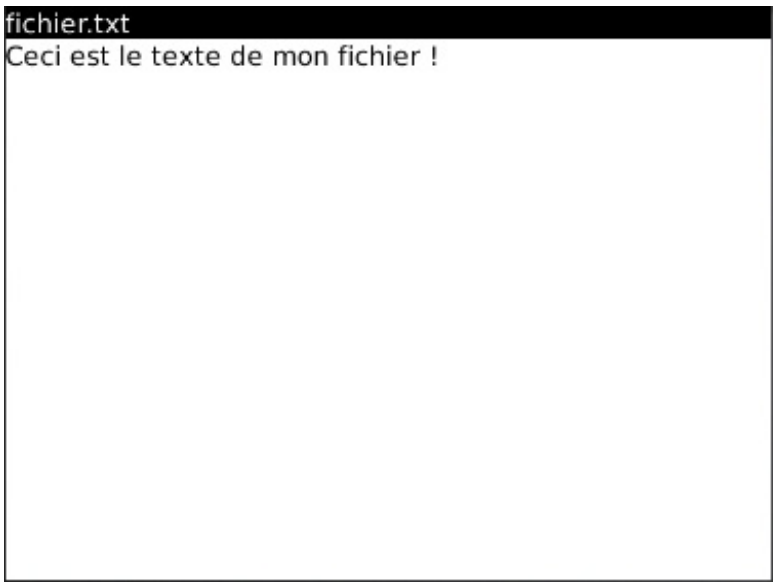
```
public String lireFichier(String fichier, String systeme){
    String donnees = new String();
    try
    {
        FileConnection fc = (FileConnection)Connector.open(systeme +
fichier);
        if (!fc.exists())
        {
            fc.create();
        }
        InputStream is = fc.openInputStream();
        byte[] data = IOUtilities.streamToBytes(is);
        is.close();
        fc.close();
        donnees = new String(data);
    }
    catch (IOException ioe)
    {
        System.out.println(ioe.getMessage() );
    }
    return(donnees);
}
```

La méthode `lireFichier()` renvoie une variable de type `String`. Nous pouvons ainsi afficher son contenu à l'aide d'un composant de type `RichTextField` par exemple.
Voilà donc ce que je vous propose :

Code : Java

```
setTitle("fichier.txt");
add(new RichTextField(lireFichier("fichier.txt", MyScreen.DEVICE)));
```

En lançant l'application, vous verrez apparaître le contenu de votre fichier à l'intérieur du champ de texte, comme vous pouvez le voir ci-dessous :



- Pour travailler avec un fichier, il est nécessaire d'établir une **connexion** avec celui-ci.
- L'interface `FileConnection` permet de se connecter à un fichier ou un dossier et d'effectuer diverses opérations dessus.
- Les méthodes `mkdir()`, `create()` et `delete()` servent respectivement à créer un dossier, créer un fichier et supprimer tout type de données.
- L'encodage d'un texte en **bytes** est réalisé à l'aide des méthodes `String()` et `getBytes()` de la classe `String`.
- L'écriture dans un fichier se fait via la classe `OutputStream`, et plus particulièrement la méthode `write()`.
- La lecture du contenu d'un fichier nécessite l'utilisation des classes `InputStream` et `IOUtilities`.

TP : un éditeur de texte

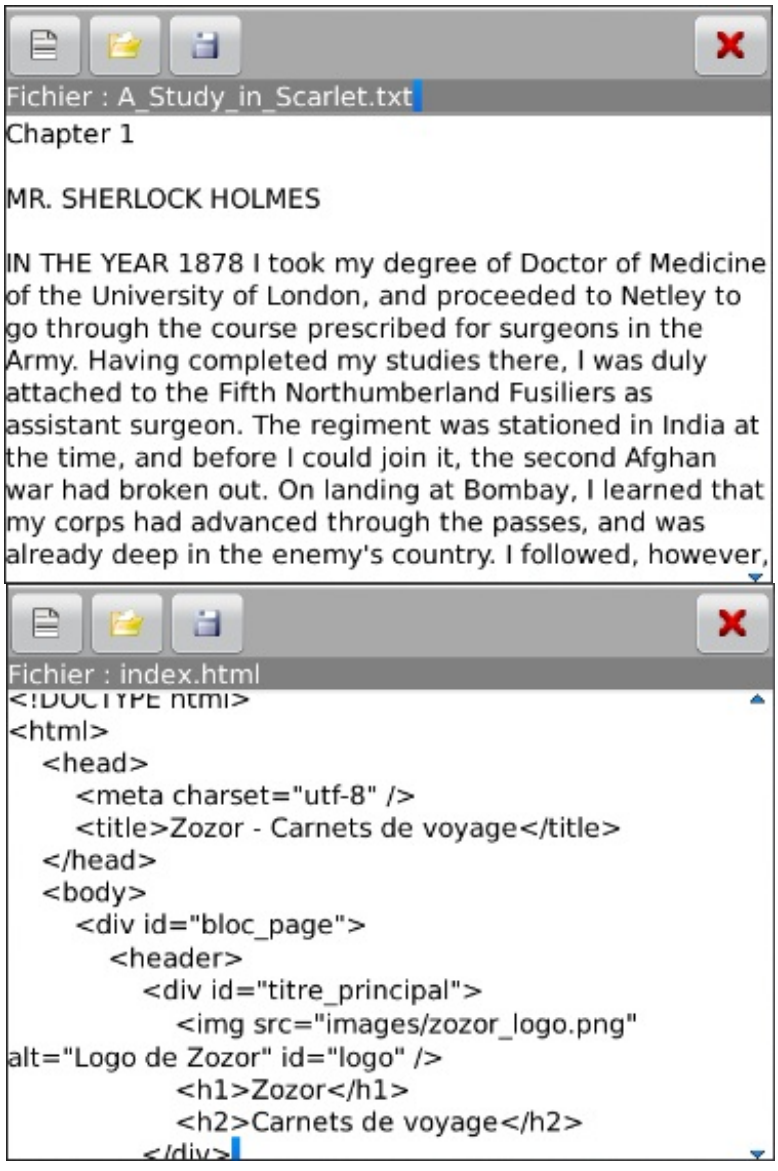
Nous revoici dans un chapitre un peu spécial, puisqu'il s'agit d'un TP !
Ce sera donc l'occasion pour vous de pratiquer à nouveau un peu avant de passer à d'autres notions plus théoriques. Au cours de ce TP nous allons, ou plutôt vous allez, réaliser un éditeur de texte qui vous permettra d'éditer différents types de fichiers. Bien entendu je vous fournirai la correction de celui-ci ainsi que des explications, qui vous permettront de finaliser ou de comparer votre solution avec la mienne.

Le cahier des charges
Spécification du projet

Au cours de ce TP, nous allons donc réaliser un éditeur de texte simplifié. Évidemment vous pourrez adopter le « design » que vous voulez pour cette application, néanmoins essayez de respecter au maximum mon interface graphique si vous souhaitez pouvoir comparer votre code avec le mien.
Sans plus attendre, je vous propose donc de faire un tour des éléments que vous aurez besoin de réaliser :

- un champ de saisie de texte pour pouvoir éditer le contenu du fichier
- un second champ de saisie de texte pour pouvoir spécifier le fichier à éditer
- un bouton « Nouveau » qui permettra de réinitialiser les deux champs de texte précédents
- un bouton « Ouvrir » pour pouvoir charger le contenu d'un fichier
- un bouton « Sauvegarder » pour enregistrer le contenu du champ de texte à l'intérieur du fichier
- un bouton « Quitter » qui servira à sortir de l'application.

Puis pour finir, je vous laisse découvrir ce que j'obtiens en utilisant le code de la correction :



Notez que suivant la taille du contenu du fichier, il se peut que votre application ait besoin d'une barre de défilement

verticale. Afin de conserver les divers boutons de contrôle de l'application, vous devrez alors être vigilants à définir le défilement sur le texte et non sur l'ensemble de la vue.

Enfin, pour fluidifier et améliorer l'utilisation des boutons, j'ai ajouté une boîte de dialogue pour confirmer l'enregistrement du fichier. D'autre part, j'interdis également l'enregistrement ou l'ouverture d'un fichier si aucun nom de fichier n'est renseigné. Pour cela j'utilise aussi des boîtes de dialogue.

Avant de commencer

Le champ de texte d'édition

Dans la partie précédente, je vous ai introduit les champs de texte éditables uniquement grâce à la classe `EditField`.
Néanmoins il existe d'autres manières d'y parvenir, notamment à l'aide de la classe `TextField`.
Ainsi dans le cas de l'éditeur de texte, j'utilise ce type de champ de la façon suivante :

Code : Java

```
monTexte = new TextField(Field.EDITABLE);
```

Toutefois, vous pouvez tout à fait utiliser des champs de texte de type `EditField` ou encore `BasicEditField`. À vous de choisir la classe que vous souhaitez, tant que vous parvenez au bout du TP.

Le défilement du contenu

Les défilements sont en réalité liés aux conteneurs et non aux composants en eux-mêmes. Ainsi en ce qui concerne le défilement vertical dans notre application, vous serez certainement amenés à utiliser les constantes de la classe `Manager`. En particulier retenez votre attention sur les constantes suivantes : `Manager.NO_VERTICAL_SCROLL`, `Manager.VERTICAL_SCROLL`, `Manager.VERTICAL_SCROLLBAR`.
Je ne vous en dis pas plus, à vous de trouver comment utiliser des constantes pour obtenir le rendu désiré !

Les icônes des boutons

Enfin avant de finir, je vous propose de télécharger les différentes icônes de boutons afin que vous puissiez au mieux vous rapprocher des aperçus présentés plus haut.
Voici donc les quatre images en question :

 nouveau.png  ouvrir.png  sauvegarder.png  quitter.png

Allez, c'est parti !

La correction

Comme dans le dernier TP, je vous propose dans un premier temps uniquement l'ensemble du code source sans explications. Je vous présenterai donc les trois classes que j'ai utilisées pour réaliser cette application.
N'hésitez donc pas à essayer cette application en copiant-collant le code source !

Encore une fois le code est proposé sans commentaire, et les explications seront alors données dans la suite.

La classe MonApp

Aucune surprise pour cette classe maintenant bien connue :

Code : Java - MonApp.java

```
package mypackage;

import net.rim.device.api.ui.UiApplication;

public class MonApp extends UiApplication
{
```

```
    public static void main(String[] args)
    {
        MonApp theApp = new MonApp();
        theApp.enterEventDispatcher();
    }

    public MonApp()
    {
        pushScreen(new Editeur());
    }
}
```

La classe ConnexionFichier

La classe `ConnexionFichier` permet exclusivement de faire les connexions vers le fichier en cours d'édition. Nous retrouverons ainsi à l'intérieur les méthodes `lireFichier()` et `ecrireFichier()` définies dans le chapitre précédent. Voici donc le code de cette classe :

Code : Java - ConnexionFichier.java

```
package mypackage;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.file.FileConnection;
import net.rim.device.api.io.IOUtilities;

public class ConnexionFichier {

    private static final String DEVICE =
"file:///store/home/user/documents/";

    public static void ecrireFichier(String fichier, String
donnees){
        try
        {
            FileConnection fc =
(FileConnection)Connector.open(DEVICE + fichier);
            if (fc.exists())
            {
                fc.delete();
            }
            fc.create();
            OutputStream os = fc.openOutputStream();
            os.write(donnees.getBytes());
            os.close();
            fc.close();
        }
        catch (IOException ioe)
        {
            System.out.println(ioe.getMessage() );
        }
    }

    public static String lireFichier(String fichier){
        String donnees = new String("");
        try
        {
            FileConnection fc =
(FileConnection)Connector.open(DEVICE + fichier);
            if (!fc.exists())
            {
                fc.create();
            }
```

```
                InputStream is = fc.openInputStream();
                byte[] data = IOUtilities.streamToBytes(is);
                is.close();
                fc.close();
                donnees = new String(data);
            }
            catch (IOException ioe)
            {
                System.out.println(ioe.getMessage() );
            }
            return(donnees);
        }

    }
```

Vous remarquerez que j'ai utilisé le mot-clé **static** pour les méthodes de cette classe. Ainsi il n'est pas nécessaire d'instancier celle-ci pour pouvoir utiliser ses méthodes. Cette technique est souvent utile pour simplifier le code lorsque nous utilisons des fonctions d'utilité commune.

La classe Editeur

Enfin, la classe `Editeur` contient l'ensemble de la logique de l'application. C'est donc cette classe que nous détaillerons dans la suite pour expliquer le fonctionnement de l'application.
En attendant, voici son code :

Code : Java - Editeur.java

```
package mypackage;

import net.rim.device.api.system.EncodedImage;
import net.rim.device.api.ui.Color;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.Manager;
import net.rim.device.api.ui.component.ButtonField;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.component.EditField;
import net.rim.device.api.ui.component.SeparatorField;
import net.rim.device.api.ui.component.TextField;
import net.rim.device.api.ui.container.HorizontalFieldManager;
import net.rim.device.api.ui.container.MainScreen;
import net.rim.device.api.ui.container.VerticalFieldManager;
import net.rim.device.api.ui.decor.Background;
import net.rim.device.api.ui.decor.BackgroundFactory;
import net.rim.device.api.ui.image.ImageFactory;

public final class Editeur extends MainScreen
{
    private EditField fichier;
    private HorizontalFieldManager mesBoutons;
    private TextField monTexte;

    public Editeur()
    {
        super(Manager.NO_VERTICAL_SCROLL);

        mesBoutons = new HorizontalFieldManager(Field.FIELD_HCENTER|Field.USE_A
        Background maCouleur = BackgroundFactory.createSolidBackground(Color.DA
        mesBoutons.setBackground(maCouleur);

        ButtonField nouveau = new ButtonField("");
        nouveau.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageR
        nouveau.setChangeListener(new FieldChangeListener() {
            public void fieldChanged(Field field, int context) {
```

```
                    monTexte.setText("");
                    fichier.setText("");
                }
            });

            ButtonField ouvrir = new ButtonField("");
            ouvrir.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageRe
            ouvrir.setChangeListener(new FieldChangeListener() {
                public void fieldChanged(Field field, int context) {
                    if(fichier.getText().equals("")){
                        Dialog.alert("Veuillez renseigner un nom de fichier.");
                    }else{
                        monTexte.setText(ConnexionFichier.lireFichier(fichier.getTe
                    }
                }
            });

            ButtonField sauvegarder = new ButtonField("");

    sauvegarder.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResou
            sauvegarder.setChangeListener(new FieldChangeListener() {
                public void fieldChanged(Field field, int context) {
                    if(fichier.getText().equals("")){
                        Dialog.alert("Veuillez renseigner un nom de fichier.");
                    }else{
                        ConnexionFichier.ecrireFichier(fichier.getText(), monTexte.
                        Dialog.alert("Le fichier a été enregistré.");
                    }
                }
            });

            VerticalFieldManager ajustement = new VerticalFieldManager(Field.USE_AI

            ButtonField quitter = new ButtonField("",Field.FIELD_RIGHT);
            quitter.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageR
            quitter.setChangeListener(new FieldChangeListener() {
                public void fieldChanged(Field field, int context) {
                    System.exit(0);
                }
            });
            ajustement.add(quitter);

            mesBoutons.add(nouveau);
            mesBoutons.add(ouvrir);
            mesBoutons.add(sauvegarder);
            mesBoutons.add(ajustement);

            fichier = new EditField("Fichier : ",""){
                public void paint(Graphics g){
                    g.setColor(0x00FFFFFF);
                    super.paint(g);
                }
            };
            maCouleur = BackgroundFactory.createSolidBackground(Color.GRAY);
            fichier.setBackground(maCouleur);

            VerticalFieldManager scrollTexte = new
    VerticalFieldManager(Manager.VERTICAL_SCROLLBAR|Manager.VERTICAL_SCROLL);
            monTexte = new TextField(Field.EDITABLE);
            monTexte.select(true);
            monTexte.setText("");
            scrollTexte.add(monTexte);

            add(new SeparatorField());
            add(mesBoutons);
            add(fichier);
            add(new SeparatorField());
            add(scrollTexte);
        }
    }
```

Les explications

Les champs de texte

Le champ de texte d'édition

Tout d'abord, nous allons nous occuper du champ de texte d'édition, c'est-à-dire de celui dans lequel nous éditerons le contenu du fichier. Cependant avant de nous lancer sur la création de ce champ, nous allons dans un premier temps régler cette histoire de défilement. Comme nous l'avons dit, nous voulons un défilement uniquement sur ce champ de texte. C'est pourquoi nous allons commencer par bloquer le défilement du conteneur principal qui hérite de `MainScreen`.
Pour cela, nous allons donc appeler le constructeur de la superclasse comme ceci :

Code : Java

```
super(Manager.NO_VERTICAL_SCROLL);
```

Maintenant que le défilement est neutralisé sur le conteneur principal, nous allons pouvoir en créer un spécifique à notre champ de texte, qui lui en revanche autorisera le défilement.
Voici donc le conteneur en question :

Code : Java

```
VerticalFieldManager scrollTexte = new
VerticalFieldManager(Manager.VERTICAL_SCROLLBAR|Manager.VERTICAL_SCROLL);
```

À présent, nous pouvons enfin nous préoccuper de notre champ de texte éditable. J'ai donc utilisé un champ de texte de type `TextField` que j'ai rendu éditable, sélectionnable et que j'ai initialisé.
Voici le code correspondant :

Code : Java

```
monTexte = new TextField(Field.EDITABLE);
monTexte.select(true);
monTexte.setText("");
```

Enfin, il suffit d'ajouter le champ de texte au conteneur pour que celui-ci puisse gérer le défilement suivant la taille du champ de texte :

Code : Java

```
scrollTexte.add(monTexte);
```

Vous remarquerez que d'origine le champ de texte éditable n'est en réalité défini que sur une seule ligne. Ainsi c'est en l'éditant que les lignes suivantes apparaîtront au fur et à mesure.

La saisie du nom de fichier

Pour renseigner le nom du fichier qui doit être édité, nous utiliserons cette fois la classe `EditField` dont nous utiliserons l'étiquette ainsi que le champ de saisie de texte. Étant donné que nous redéfinirons la couleur d'arrière-plan de ce champ, j'ai opté pour un texte de couleur blanche.
Voici donc l'initialisation de ce champ, ainsi que la redéfinition de la couleur d'écriture :

Code : Java

```
        fichier = new EditField("Fichier : ",""){
            public void paint(Graphics g){
                g.setColor(0x00FFFFFF);
                super.paint(g);
            }
        };
```

Comme promis, modifions la couleur d'arrière-plan :

Code : Java

```
        maCouleur = BackgroundFactory.createSolidBackground(Color.GRAY);
        fichier.setBackground(maCouleur);
```

Les différents boutons

À présent nous allons nous occuper des différents boutons qui permettent de gérer toute l'interaction de l'application. Ceux-ci seront alors stockés à l'intérieur d'un conteneur horizontal dont nous allons changer la couleur de fond.
Voici donc la définition de ce conteneur :

Code : Java

```
        mesBoutons = new
        HorizontalFieldManager(Field.FIELD_HCENTER|Field.USE_ALL_WIDTH);
        Background maCouleur =
        BackgroundFactory.createSolidBackground(Color.DARKGRAY);
        mesBoutons.setBackground(maCouleur);
```

Nouveau

Commençons par le plus à gauche !
Le bouton « Nouveau » va nous servir à réinitialiser les deux champs de texte définis plus haut. Comme pour l'ensemble de nos boutons, nous utiliserons une icône plutôt qu'un texte pour définir celui-ci.
Déclarons donc ce bouton :

Code : Java

```
    ButtonField nouveau = new ButtonField("");
    nouveau.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResource("r
```

Comme nous l'avons dit, ce bouton va nous servir à réinitialiser nos champs de texte. L'évènement associé au bouton n'est pas très compliqué, puisqu'il se contente d'appeler les méthodes `setText()` des deux champs de texte.
Voilà donc comment procéder :

Code : Java

```
        nouveau.setChangeListener(new FieldChangeListener() {
            public void fieldChanged(Field field, int context) {
                monTexte.setText("");
                fichier.setText("");
            }
        });
```

Ouvrir

Comme pour le bouton précédent, déclarons-le :

Code : Java

```
ButtonField ouvrir = new ButtonField("");
ouvrir.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResource("ou
```

Ici nous allons charger le contenu du fichier à l'intérieur du champ de texte d'édition. Pour cela, nous allons donc nous servir de la méthode `lireFichier()` de notre classe `ConnexionFichier`. Sachant que la difficulté résidait principalement dans le code de la méthode `lireFichier()`, nous n'avons à présent plus qu'à l'utiliser à l'intérieur d'une condition. Celle-ci nous permettra également d'avertir l'utilisateur si aucun nom de fichier n'est précisé, grâce à une boîte de dialogue.
Voici le résumé des opérations :

Code : Java

```
ouvrir.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        if(fichier.getText().equals("")){
            Dialog.alert("Veuillez renseigner un nom de fichier.");
        }else{

monTexte.setText(ConnexionFichier.lireFichier(fichier.getText()));
        }
    }
});
```

Sauvegarder

Nous avons, comme d'habitude :

Code : Java

```
ButtonField sauvegarder = new ButtonField("");
sauvegarder.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResourc
```

La gestion de l'évènement du bouton « Sauvegarder » est en fait très proche de celle du bouton « Ouvrir ». Nous nous servirons cette fois la méthode `ecrireFichier()`, et nous utiliserons une boîte de dialogue sans quoi l'utilisateur ne verra absolument rien se passer à l'écran.
Voici le code de cet évènement :

Code : Java

```
sauvegarder.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        if(fichier.getText().equals("")){
            Dialog.alert("Veuillez renseigner un nom de fichier.");
        }else{
            ConnexionFichier.ecrireFichier(fichier.getText(),
monTexte.getText());
            Dialog.alert("Le fichier a été enregistré.");
        }
    }
});
```

Quitter

Si vous avez bien visualisé les aperçus de l'application en début de chapitre, vous aurez noté que le bouton « Quitter » est aligné à droite, contrairement à tous les autres qui sont à gauche. C'est pourquoi nous allons donc être obligés d'utiliser un nouveau conteneur. Afin de gérer correctement la mise en place des boutons, nous allons définir un conteneur qui prendra toute la place disponible en largeur, afin de « pousser » les autres boutons à gauche.
Voici donc comment faire :

Code : Java

```
VerticalFieldManager ajustement = new
VerticalFieldManager(Field.USE_ALL_WIDTH);
```

Puis nous allons créer notre bouton, et l'aligner à droite à l'intérieur de ce conteneur :

Code : Java

```
ButtonField quitter = new ButtonField("",Field.FIELD_RIGHT);
quitter.setImage(ImageFactory.createImage(EncodedImage.getEncodedImageResource("c
```

L'évènement de ce bouton est très simple puisqu'il s'agit uniquement de sortir de l'application.
Voilà donc le code correspondant :

Code : Java

```
quitter.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        System.exit(0);
    }
});
```

Enfin, n'oubliez pas d'ajouter cet élément au conteneur que nous venons de définir :

Code : Java

```
ajustement.add(quitter);
```

La mise en place des boutons

Maintenant que nous avons créé tous les boutons, nous allons pouvoir les disposer à l'intérieur du conteneur `mesBoutons` que nous avions déclaré auparavant. La seule chose à prendre en compte est l'ordre dans lequel ceux-ci doivent être ajoutés.
Voici ce que j'ai fait :

Code : Java

```
mesBoutons.add(nouveau);
mesBoutons.add(ouvrir);
mesBoutons.add(sauvegarder);
mesBoutons.add(ajustement);
```

La mise en page finale

Enfin pour clore ce TP, il ne nous reste plus qu'à ajouter l'ensemble de nos composants à l'intérieur du conteneur principal.
Voilà donc le code qui termine ce chapitre :

Code : Java

```
add(new SeparatorField());
add(mesBoutons);
add(fichier);
add(new SeparatorField());
add(scrollTexte);
```

Le multimédia

Dans ce chapitre, nous allons voir comment utiliser des fichiers multimédias à l'intérieur d'une application. Nous aborderons donc les fichiers de types audio et vidéo. Dans un premier temps, nous verrons comment transférer des fichiers depuis notre ordinateur vers le simulateur. Puis nous apprendrons à manipuler et gérer la lecture de ces fichiers.
Pour vous faciliter les choses à l'avenir, je vous proposerai également des classes toutes prêtes, que vous pourrez directement utiliser et qui vous feront gagner un temps précieux !

Transférer des fichiers

Créer un dossier de partage

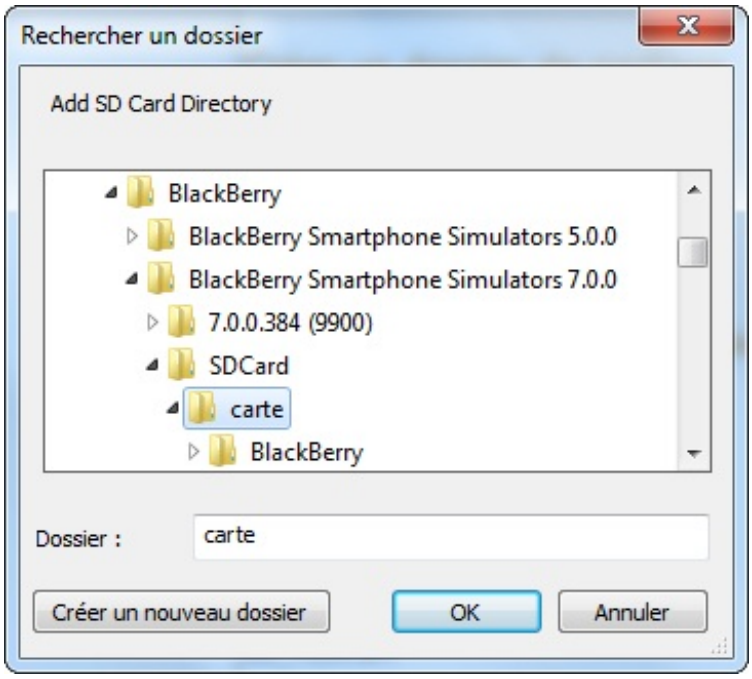
Introduction

Sûrement vous êtes-vous déjà demandé comment transférer divers fichiers depuis votre ordinateur jusqu'au simulateur ?

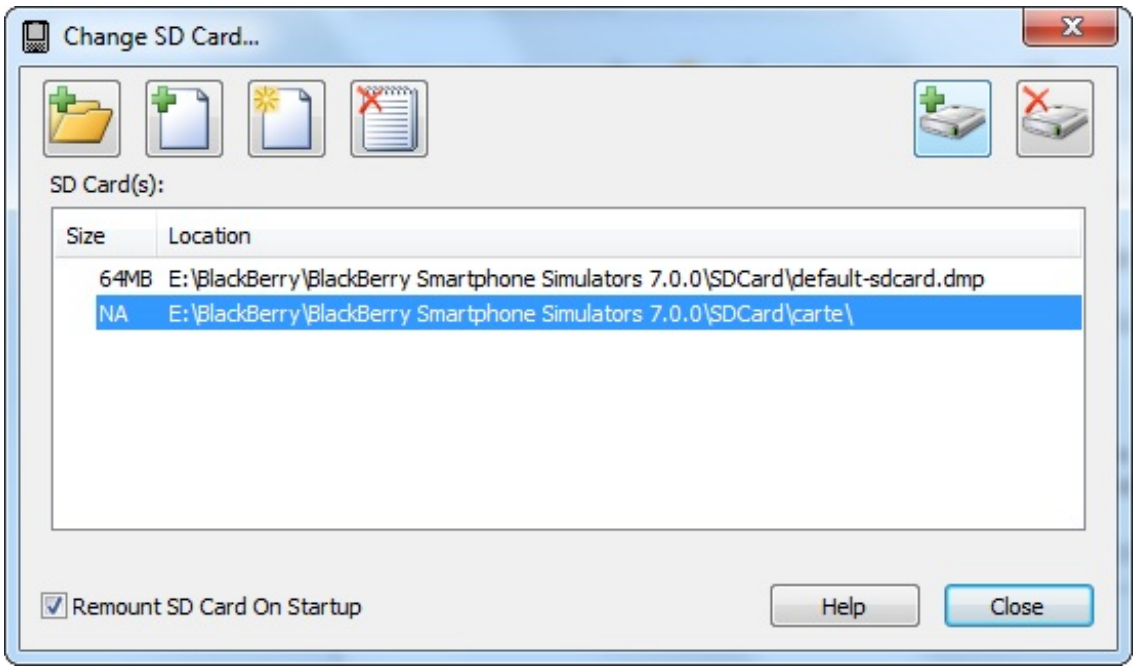
Étant donné que nous allons apprendre à lire des fichiers multimédias dans ce chapitre, nous allons également devoir voir comment transférer facilement des fichiers vers notre simulateur. Pour réaliser ceci, il existe diverses techniques. Néanmoins, la solution la plus facile consiste à réutiliser le système de gestion de cartes mémoires externes.
La dernière fois, nous avons ajouté une carte en créant un fichier de type « `.dmp` », qui contenait alors l'intégralité des fichiers stockés à l'intérieur de notre carte mémoire. Sachez qu'il existe également une autre manière de créer une carte mémoire externe pour notre simulateur : utiliser un **dossier de partage** !
Le principe consiste à utiliser un « vrai » dossier de notre disque dur pour simuler la carte mémoire. Nous pourrons ainsi ajouter aisément des fichiers à l'intérieur de ce dossier, qui seront alors visibles depuis le simulateur et ses applications.

Mise en place

Tout d'abord, pour créer un dossier de partage, commencez par lancer votre simulateur. Ensuite, rendez-vous dans le gestionnaire de cartes mémoires par le menu `Simulate > Change SD Card...`, comme la dernière fois.
Puis cliquez sur le bouton `Add Directory` pour associer un dossier à une carte mémoire.
Je vous laisse ensuite choisir un dossier de votre disque dur à associer à votre nouvelle carte (voir la figure suivante).



Une fois créée, votre carte devrait apparaître dans le gestionnaire avec une taille non spécifiée « `NA` », ce qui veut dire que vous pouvez y insérer autant de fichiers que vous le souhaitez.
Il ne vous reste alors plus qu'à l'insérer dans le simulateur, en la sélectionnant et en cliquant sur le bouton `Mount Selected`.



Après quelques messages d'avertissement à l'intérieur de votre simulateur, votre carte est enfin prête à être utilisée.

Transférer des fichiers multimédias

Trouver des fichiers compatibles

Dans un premier temps, nous allons voir comment lire des fichiers sonores. C'est pourquoi nous aurons besoin d'une musique. Je vous propose donc de prendre la musique de votre choix au format MP3, et de la renommer simplement en « `musique.mp3` ». Placez-la alors dans le dossier de partage créé juste avant.

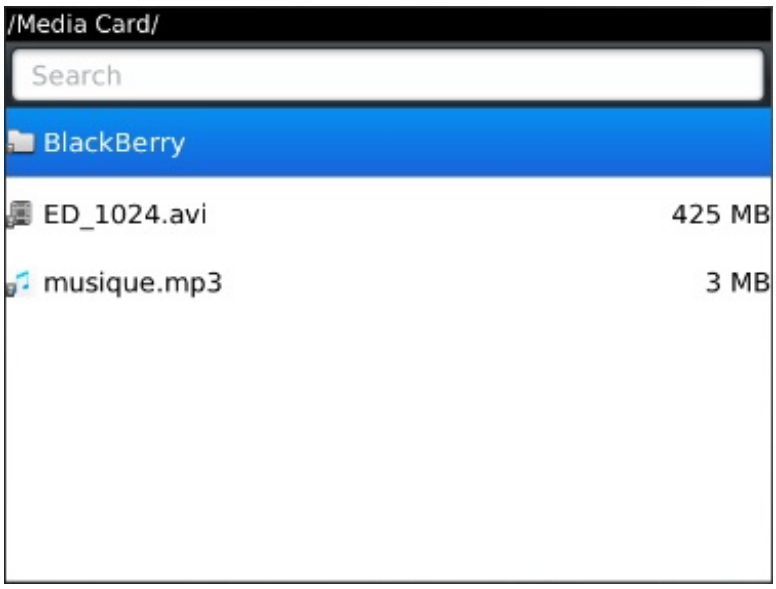
Si les formats audio sont plutôt bien supportés dans l'ensemble, il n'en est pas de même en ce qui concerne les vidéos. Je vous avouerai qu'il a été plutôt difficile de trouver une vidéo qui soit directement compatible. Peut-être avez-vous déjà entendu parler du projet Orange nommé « Elephants Dream ». Si je vous en parle aujourd'hui, c'est parce que je vous propose de télécharger ce film : ED_1024.avi. Une fois le téléchargement terminé, placez également cette vidéo dans le dossier de partage afin que nous y ayons accès depuis notre simulateur.

Pour vos futurs projets, je vous conseille de jeter un coup d'œil à la liste des formats supportés par les différentes versions de BlackBerry OS. Si les diverses vidéos que vous désirez utiliser ne sont pas compatibles, je vous invite à les encoder par vous-mêmes.

Tester la compatibilité des fichiers

Enfin, avant de vous lancer tête baissée dans votre application et de passer des heures à essayer de la débugger, pensez au moins à vérifier la compatibilité de vos fichiers. Ainsi vous serez certains que les erreurs proviennent de votre code et non des fichiers que vous tentez de lire.

Pour commencer, rendez-vous dans le gestionnaire des fichiers afin de vérifier que les fichiers y sont bien présents.



Ensuite, utilisez les lecteurs par défaut du simulateur pour vérifier le bon fonctionnement de l'ensemble de vos fichiers. Pour cela, il vous suffit de cliquer sur chacun d'eux.
Voici par exemple sur la figure suivante la vidéo « Elephants Dream ».

Comme vous pouvez le constater, nous avons bien l'image mais le son ne semble pas fonctionner sur cette vidéo. Dans notre cas, ceci n'est pas bien grave. Toutefois nous savons maintenant que cela provient de la compatibilité du fichier et non de notre future application.

Voyons maintenant comment utiliser ces divers fichiers multimédias au sein d'une application !

Jouer de l'audio

Lire un fichier audio

Jouer une musique

L'objectif du chapitre est de vous permettre de charger et manipuler différents fichiers multimédias. Ainsi la manipulation de ces données se fait via la classe Java nommée `Player`, elle-même associée à un fichier multimédia.
Pour démarrer, il nous faut donc déclarer un nouveau `Player` :

Code : Java

```
Player p;
```

Le chargement du fichier est alors une étape relativement simple, puisqu'il suffit d'utiliser la méthode `createPlayer()` de la classe `Manager`.

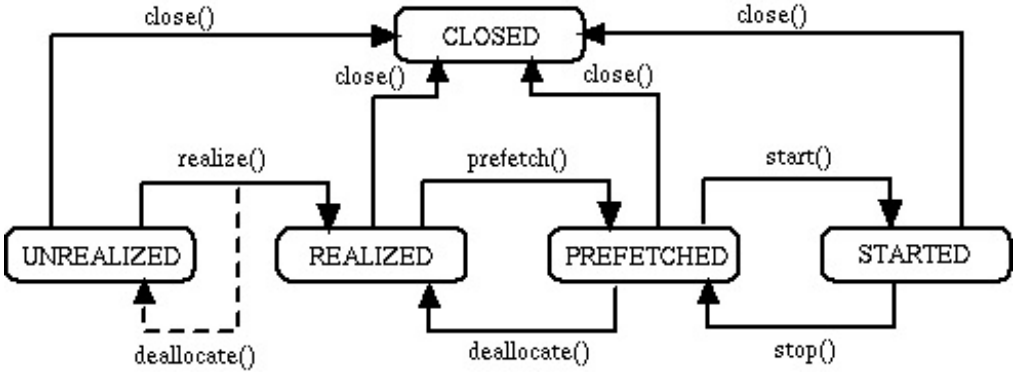
Soyer attentif ici car nous n'utilisons pas la classe `Manager` spécifique à l'API BlackBerry, mais celle définie en Java dans le package `javax.microedition.media`. Veillez donc à importer la bonne classe à l'intérieur de votre code, sinon vous aurez sans nul doute des erreurs à la compilation.

Voici donc comment instancier notre `Player` :

Code : Java

```
p = Manager.createPlayer("file:///SDCard/musique.mp3");
```

Si vous jetez un œil à la documentation de la classe Player, vous verrez que celle-ci passe par différentes étapes avant de pouvoir lancer la lecture du média définie par la constante `STARTED` (voir la figure suivante).



Ainsi il nous est donc nécessaire d'utiliser les trois méthodes `realize()`, `prefetch()` et `start()` dans cet ordre précis :

Code : Java

```
p.realize();
p.prefetch();
p.start();
```

Une fois la méthode `start()` appelée, la lecture du fichier audio est lancée. Vous pouvez alors, à tout moment, arrêter la lecture grâce à la méthode `stop()`. Nous verrons un peu plus loin comment combiner ces méthodes `start()` et `stop()` pour créer des boutons « Lecture » et « Pause ».
Également, vous pouvez régler le volume de sortie en utilisant la classe `VolumeControl`, comme ceci :

Code : Java

```
VolumeControl volume = (VolumeControl)p.getControl("VolumeControl");
volume.setLevel(30);
```

Vous devez aussi savoir que l'ensemble de ces opérations pourrait générer des erreurs lors de l'exécution du code. Je vous invite donc à encercler le tout d'un bloc `try{....} catch{...}`.
Voici donc un résumé de l'intégralité des opérations :

Code : Java

```
String SD_CARD = "file:///SDCard/";
```

```
Player p;
try
{
    p = Manager.createPlayer("file:///SDCard/musique.mp3");
    p.realize();
    VolumeControl volume =
(VolumeControl)p.getControl("VolumeControl");
    volume.setLevel(30);
    p.prefetch();
    p.start();
}
catch (MediaException me)
{
    Dialog.alert(me.toString());
}
catch (IOException ioe)
{
    Dialog.alert(ioe.toString());
}
```

Une classe prête à l'emploi

Sachant que j'aime bien ça et que cela vous facilite la vie, je vous propose donc une classe dont vous pouvez vous servir directement :

Code : Java - LecteurAudio.java

```
package mypackage;

import java.io.IOException;
import javax.microedition.media.Manager;
import javax.microedition.media.MediaException;
import javax.microedition.media.Player;
import javax.microedition.media.control.VolumeControl;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.container.VerticalFieldManager;

public class LecteurAudio extends VerticalFieldManager{

    private static final String SD_CARD = "file:///SDCard/";
    private Player p;

    public LecteurAudio(String fichier)
    {
        try
        {
            p = Manager.createPlayer(SD_CARD + fichier);
            p.realize();
            VolumeControl volume =
(VolumeControl)p.getControl("VolumeControl");
            volume.setLevel(30);
            p.prefetch();
            p.start();
        }
        catch (MediaException me)
        {
            Dialog.alert(me.toString());
        }
        catch (IOException ioe)
        {
            Dialog.alert(ioe.toString());
        }
    }
}
```

Vous remarquerez que j'ai créé une classe qui hérite de `VerticalFieldManager`. Ceci n'a pas de grande utilité pour l'instant, mais vous verrez que cela sera utile lorsqu'il s'agira par exemple d'ajouter des boutons de contrôle à celle-ci.
Voici alors comment utiliser cette classe `LecteurAudio` :

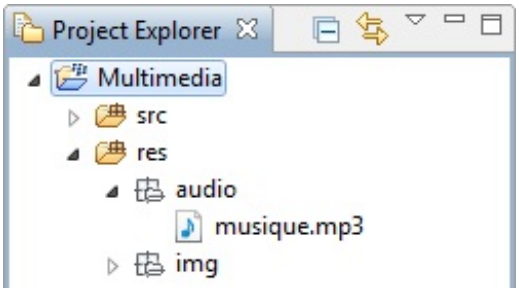
Code : Java

```
LecteurAudio monLecteur = new LecteurAudio("musique.mp3");
add(monLecteur);
```

Insérer du son à l'application

Jouer un son

Étant donné que nous parlons d'audio, je pense que c'est le bon moment pour vous montrer comment insérer du son à l'intérieur de votre application. En général, cette technique n'est pas vraiment faite pour insérer des musiques mais plutôt divers sons et bruitages utilisés principalement dans les jeux vidéo. Toutefois dans notre cas, nous réutiliserons notre fichier `musique.mp3`. Je vous invite donc à ajouter ce fichier aux ressources de votre application. De mon côté, j'ai créé un nouveau dossier `audio` pour y stocker l'ensemble des fichiers de type audio.



Soyez vigilants quant à la taille maximale des ressources, à savoir : 7802880 octets. À cause de cela, vous serez limités et je vous conseille donc d'éviter d'y insérer des fichiers lourds, tels que des vidéos par exemple.

Comme vous vous en doutez, la façon de créer un `Player` va être différente de ce que nous avons déjà vu. En revanche une fois celui-ci créé, nous le manipulerons identiquement.
Une manière de réaliser ceci est d'ouvrir le fichier en utilisant la classe principale de notre application. Pour cela, nous allons utiliser une classe un peu particulière : `Class`. Nous allons en fait stocker une référence à notre classe principale en nous servant de la méthode `forName()`, comme ceci :

Code : Java

```
Class cl = null;
try
{
    cl = Class.forName("mypackage.MyApp");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

Ensuite nous allons accéder au fichier audio via la méthode `getResourceAsStream()` et la classe `InputStream`, que vous connaissez déjà.
Voilà comment réaliser ceci :

Code : Java

```
if (cl!=null) {
    InputStream is = cl.getResourceAsStream("/musique.mp3");
}
```

Ainsi nous pouvons enfin créer notre `Player` grâce à l'instruction suivante :

Code : Java

```
p = Manager.createPlayer(is, "audio/mpeg");
```

Comme je l'ai déjà dit, une fois ouvert, notre `Player` s'utilise exactement de la même manière que précédemment.

Ajouter un bouton de contrôle

Pour aller un peu plus loin ici, nous allons réaliser un bouton de contrôle qui permettra d'arrêter et relancer la lecture du fichier audio.
Pour commencer, créons d'abord notre bouton, puis ajoutons-lui un écouteur « vide » :

Code : Java

```
ButtonField monBouton = new
ButtonField("Pause",Field.FIELD_HCENTER);
monBouton.setChangeListener(new FieldChangeListener() {
    public void fieldChanged(Field field, int context) {
        // Instructions
    }
});
```

Étant donné que nous n'utiliserons qu'un seul bouton, nous allons devoir tester l'état de notre `Player`. Si vous avez bien suivi, vous devriez donc savoir qu'une fois en lecture, le `Player` est dans l'état `STARTED`.
Ainsi nous pouvons réaliser le contenu de la méthode `fieldChanged()` à l'aide d'une condition **`if`** :

Code : Java

```
if(p.getState() == Player.STARTED){
    p.stop();
    monBouton.setLabel("Lecture");
}else{
    p.start();
    monBouton.setLabel("Pause");
}
```

En réalité, l'intégralité de cette condition doit également être placée à l'intérieur d'un bloc **`try`**`{....}` **`catch`**`{...}` comme nous l'avons fait avant pour l'instanciation du `Player`.

Une classe toute prête

Comme précédemment, je vous propose une classe intégrale, prête à l'emploi :

Code : Java - LecteurSon.java

```
package mypackage;

import java.io.IOException;
import java.io.InputStream;
import javax.microedition.media.Manager;
import javax.microedition.media.MediaException;
import javax.microedition.media.Player;
import javax.microedition.media.control.VolumeControl;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.FieldChangeListener;
import net.rim.device.api.ui.component.ButtonField;
```

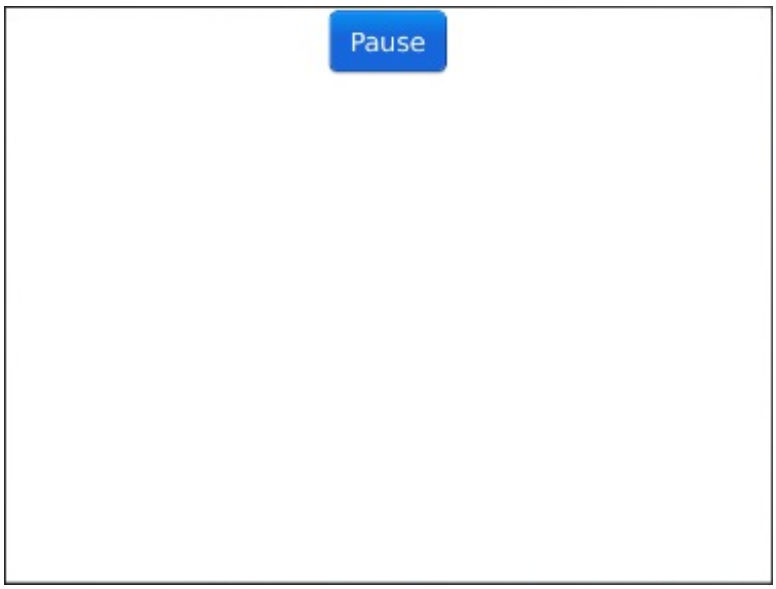
```
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.container.VerticalFieldManager;

public class LecteurSon extends VerticalFieldManager{

    private Player p;
    private ButtonField monBouton;

    public LecteurSon(String fichier)
    {
        super(Field.USE_ALL_WIDTH);
        Class cl = null;
        try
        {
            cl = Class.forName("mypackage.MyApp");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        if (cl!=null) {
            InputStream is = cl.getResourceAsStream("/" + fichier);
            try
            {
                p = Manager.createPlayer(is, "audio/mpeg");
                p.realize();
                VolumeControl volume =
(VolumeControl)p.getControl("VolumeControl");
                volume.setLevel(30);
                p.prefetch();
                p.start();
            }
            catch(MediaException me)
            {
                Dialog.alert(me.toString());
            }
            catch(IOException ioe)
            {
                Dialog.alert(ioe.toString());
            }
        }
        monBouton = new ButtonField("Pause",Field.FIELD_HCENTER);
        monBouton.setChangeListener(new FieldChangeListener() {
            public void fieldChanged(Field field, int context) {
                try
                {
                    if(p.getState() == Player.STARTED){
                        p.stop();
                        monBouton.setLabel("Lecture");
                    }else{
                        p.start();
                        monBouton.setLabel("Pause");
                    }

                }
                catch(MediaException me)
                {
                    Dialog.alert(me.toString());
                }
            }
        });
        add(monBouton);
    }
}
```

Évidemment cette classe s'utilise de la même façon que `LecteurAudio` que nous avions créée auparavant :

Code : Java

```
LecteurSon monLecteur = new LecteurSon("musique.mp3");
add(monLecteur);
```

Avec l'arrivée du nouveau bouton, vous devriez maintenant voir l'intérêt de créer une classe qui hérite de `VerticalFieldManager` :



Lire une vidéo

Chargement du fichier

Lorsqu'il s'agit d'un fichier vidéo, la création du `Player` se fait de la même manière que pour un fichier audio.
Le début du code est donc strictement identique :

Code : Java

```
p = Manager.createPlayer(SD_CARD + "ED_1024.avi");
p.realize();
```

La grande différence par rapport à un fichier audio, c'est qu'ici nous allons être obligés d'utiliser un composant graphique pour afficher le contenu visuel du fichier. Pour cela, nous utiliserons donc le composant de base `Field` ainsi que la classe `AdvancedVideoControl` qui permettra de créer ce composant basique.
La technique utilisée ici est légèrement complexe, c'est pourquoi je vous propose directement le code permettant de créer l'objet de type `Field` :

Code : Java

```
Field field;
AdvancedVideoControl vc;
if ((vc = (AdvancedVideoControl)
p.getControl("net.rim.device.api.media.control.AdvancedVideoControl")) != null)
{
    field =
(Field)vc.initDisplayMode(AdvancedVideoControl.USE_GUI_ADVANCED,"net.rim.device.a
    vc.setVisible(true);
}
```

Une fois ce travail réalisé, nous pouvons après contrôler notre `Player` comme avant, notamment avec les méthodes `start()` et `stop()` :

Code : Java

```
p.start()
```

Enfin, n'oubliez pas dans ce cas-là d'ajouter le composant `field` au conteneur :

Code : Java

```
add(field);
```

Une classe prête à l'emploi

Encore une fois, vous trouverez dans la suite une classe toute prête que vous pouvez utiliser à votre guise. Notez que j'y ai inséré des contrôles de lectures de types « tactiles ».
Voici cette classe nommée `LecteurVideo` :

Code : Java - LecteurVideo.java

```
package mypackage;

import java.io.IOException;

import javax.microedition.media.Manager;
import javax.microedition.media.MediaException;
import javax.microedition.media.Player;
import net.rim.device.api.media.control.AdvancedVideoControl;
import net.rim.device.api.ui.Field;
import net.rim.device.api.ui.TouchEvent;
import net.rim.device.api.ui.component.Dialog;
import net.rim.device.api.ui.container.VerticalFieldManager;

public class LecteurVideo extends VerticalFieldManager{

    private static final String SD_CARD = "file:///SDCard/";
    private Field field;
    private Player p;

    public LecteurVideo(String fichier)
    {
        try
        {
            p = Manager.createPlayer(SD_CARD + fichier);
            p.realize();
            AdvancedVideoControl vc;
            if ((vc = (AdvancedVideoControl)
p.getControl("net.rim.device.api.media.control.AdvancedVideoControl")) != null)
            {
                field =
(Field)vc.initDisplayMode(AdvancedVideoControl.USE_GUI_ADVANCED,"net.rim.device.
                vc.setVisible(true);
            }
            p.start();
        }
        catch (MediaException me)
        {
            Dialog.alert(me.toString());
        }
        catch (IOException ioe)
```

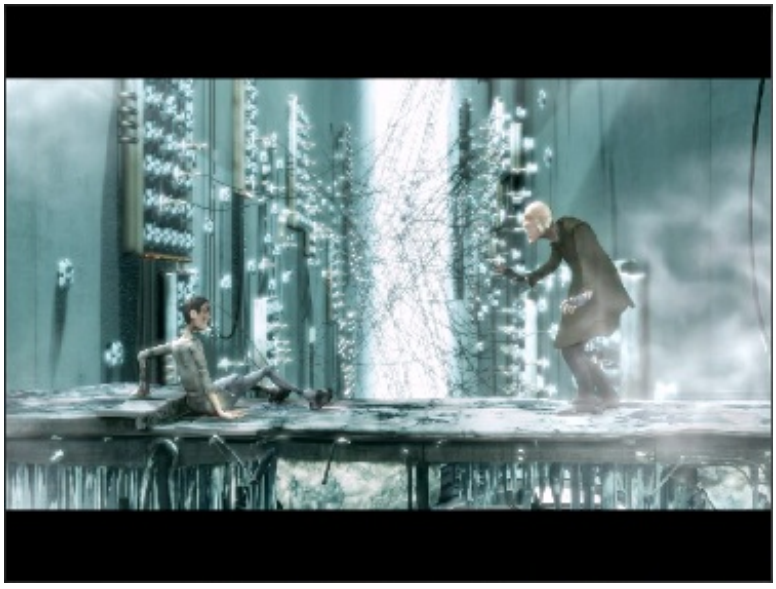
```
            {
                Dialog.alert(ioe.toString());
            }
            add(field);
        }

        protected boolean touchEvent(TouchEvent message)
        {
            int eventCode = message.getEvent();
            if(eventCode == TouchEvent.DOWN) {
                try
                {
                    if(p.getState() == Player.STARTED){
                        p.stop();
                    }else{
                        p.start();
                    }
                }
                catch(MediaException me)
                {
                    Dialog.alert(me.toString());
                }
            }
            return true;
        }
    }
```

Voici comment déclarer et instancier cette classe :

Code : Java

```
LecteurVideo monLecteur = new LecteurVideo("ED_1024.avi");
add(monLecteur);
```

Au lancement de l'application, vous verrez alors la vidéo se lire à l'intérieur de celle-ci.



- Il est possible de transférer des fichiers vers le simulateur en définissant une carte mémoire comme **dossier de partage**.
- Tous les formats de médias ne sont pas pris en compte, il est donc nécessaire de vérifier les formats supportés.
- Pour contrôler un fichier multimédia, nous utilisons la classe `Player`.
- La méthode `createPlayer()` sert à associer une variable de type `Player` à un fichier multimédia.
- La gestion de la lecture du fichier se fait via les méthodes `start()` et `stop()` de la classe `Player`.

L'accéléromètre

Dans ce chapitre, nous allons apprendre à utiliser l'**accéléromètre**. Il s'agit en fait d'un *capteur* fixé au smartphone, qui permet notamment de récupérer des données liées à l'orientation de ce dernier. Nous verrons donc comment l'utiliser concernant d'une part l'orientation « grossière », puis l'orientation « précise » du smartphone. Ce chapitre sera aussi l'occasion de s'intéresser à la gestion du temps à l'intérieur d'une application. Cela nous permettra de pouvoir récupérer des données provenant de l'accéléromètre à intervalles réguliers.

L'orientation « grossière »

Introduction

Présentation du concept d'orientation

Lorsque vous créerez des applications, vous aurez peut-être besoin de connaître l'orientation « grossière » du smartphone afin de pouvoir exécuter des instructions différentes suivant chaque cas. Dans un premier temps, nous allons donc voir comment utiliser cette orientation « grossière » à l'intérieur de notre code. Nous verrons comment récupérer l'orientation « précise » du smartphone plus tard dans ce chapitre.

Pour connaître l'orientation du smartphone, ou du simulateur en ce qui nous concerne, nous allons utiliser les constantes de la classe `AccelerometerSensor` :

- `ORIENTATION_BACK_UP`
- `ORIENTATION_BOTTOM_UP`
- `ORIENTATION_FRONT_UP`
- `ORIENTATION_LEFT_UP`
- `ORIENTATION_RIGHT_UP`
- `ORIENTATION_TOP_UP`
- `ORIENTATION_UNKNOWN`.

Ainsi grâce à ces constantes nous connaîtrons l'orientation « grossière » du simulateur, que nous afficherons directement à l'écran pour mieux en comprendre le fonctionnement.

Préparation de la classe `MyScreen`

Pour gérer cette orientation, nous allons dans la suite créer une classe nommée `Orientation`. Toutefois nous allons préparer en amont la classe `MyScreen` que voici :

Code : Java - MyScreen.java

```
package mypackage;

import net.rim.device.api.ui.component.RichTextField;
import net.rim.device.api.ui.container.MainScreen;

public class MyScreen extends MainScreen{

    private RichTextField monTexte;
    private Orientation orientation;

    public MyScreen(){
        super();
        monTexte = new RichTextField();
        orientation = new Orientation(this);
        add(monTexte);
    }

    public void setTexte(String texte){
        monTexte.setLabel(texte);
    }
}
```

Ce qu'il est important de noter dans cette classe, c'est l'apparition d'une méthode `setTexte()` pour modifier le contenu du champ de texte, ainsi que le paramètre **`this`** à l'intérieur du constructeur de notre future classe `Orientation`. En réalité, ce mot-clé **`this`** nous permet uniquement de passer une référence de notre `MyScreen`, afin que nous puissions appeler la méthode `setTexte()` depuis l'intérieur de notre classe `Orientation`.

Création de la classe `Orientation`

Implémenter l'interface `AccelerometerListener`

Pour créer notre classe `Orientation`, nous allons devoir implémenter l'interface `AccelerometerListener`, qui est un écouteur lié à l'accéléromètre. À l'intérieur, nous devrons alors redéfinir la méthode `onData()` de l'interface qui sera appelée à chaque mouvement du smartphone.
Voici donc la structure de base de notre classe `Orientation` :

Code : Java

```
package mypackage;

import net.rim.device.api.system.AccelerometerData;
import net.rim.device.api.system.AccelerometerListener;
import net.rim.device.api.system.AccelerometerSensor;
import net.rim.device.api.system.Application;
import net.rim.device.api.system.AccelerometerSensor.Channel;

public class Orientation implements AccelerometerListener{

    public Orientation(){
        Channel orientationChannel =
AccelerometerSensor.openOrientationDataChannel(Application.getApplication());
        orientationChannel.addAccelerometerListener(this);
    }

    public void onData(AccelerometerData donnees)
    {
        // Instructions lorsqu'un évènement est généré
    }

}
```

Notez que j'ai déjà inséré les lignes suivantes, afin de créer et d'ajouter un écouteur à l'accéléromètre :

Code : Java

```
Channel orientationChannel =
AccelerometerSensor.openOrientationDataChannel(Application.getApplication());
orientationChannel.addAccelerometerListener(this);
```

Enfin pour finir voici le plus intéressant, nous allons récupérer l'orientation du simulateur sous forme d'entier grâce à la méthode `getOrientation()` de notre objet `donnees`. Cet entier pourra ainsi être comparé aux différentes constantes de la classe `AccelerometerSensor` que nous avons vues plus haut.
Voici l'instruction de récupération de l'orientation :

Code : Java

```
int orientation = donnees.getOrientation();
```

La classe `Orientation` complète

Le reste de la classe `Orientation` est uniquement du traitement de données. C'est pour cela que je vous propose directement le code final de cette classe :

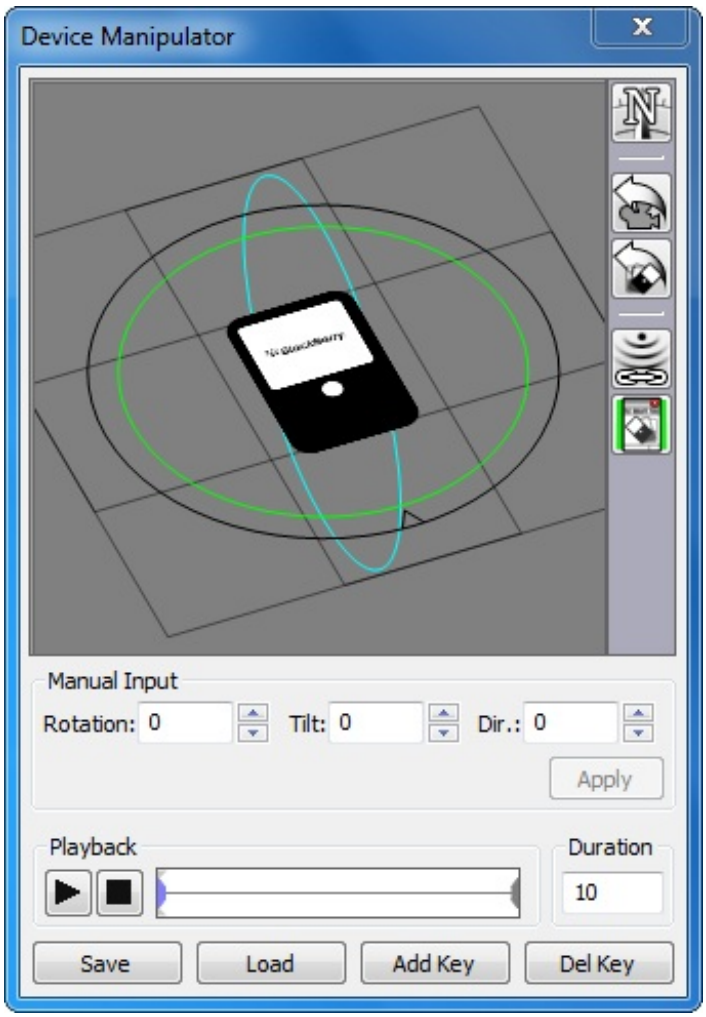
Code : Java - Orientation.java

```
package mypackage;

import net.rim.device.api.system.AccelerometerData;
import net.rim.device.api.system.AccelerometerListener;
import net.rim.device.api.system.AccelerometerSensor;
import net.rim.device.api.system.Application;
import net.rim.device.api.system.AccelerometerSensor.Channel;

public class Orientation implements AccelerometerListener{

    private MyScreen maVue;
    private int orientation;

    public Orientation(MyScreen nouvelleVue){
        super();
        maVue = nouvelleVue;
        orientation = 0;
        Channel orientationChannel =
AccelerometerSensor.openOrientationDataChannel(Application.getApplication());
        orientationChannel.addAccelerometerListener(this);
    }

    public void onData(AccelerometerData donnees)
    {
        orientation = donnees.getOrientation();
        String texte = new String();
        switch (orientation){
            case AccelerometerSensor.ORIENTATION_BACK_UP:
                texte = "ORIENTATION_BACK_UP";
                break;
            case AccelerometerSensor.ORIENTATION_BOTTOM_UP:
                texte = "ORIENTATION_BOTTOM_UP";
                break;
            case AccelerometerSensor.ORIENTATION_FRONT_UP:
                texte = "ORIENTATION_FRONT_UP";
                break;
            case AccelerometerSensor.ORIENTATION_LEFT_UP:
                texte = "ORIENTATION_LEFT_UP";
                break;
            case AccelerometerSensor.ORIENTATION_RIGHT_UP:
                texte = "ORIENTATION_RIGHT_UP";
                break;
            case AccelerometerSensor.ORIENTATION_TOP_UP:
                texte = "ORIENTATION_TOP_UP";
                break;
            case AccelerometerSensor.ORIENTATION_UNKNOWN:
                texte = "ORIENTATION_UNKNOWN";
                break;
            default:
                texte = "inconnue";
        }
        maVue.setTexte("orientation : " + texte);
    }

}
```

Visualisation de l'application

Maintenant, lancez l'application afin de voir le résultat de ce code. À l'intérieur de la fenêtre du simulateur, vous pouvez manipuler celui-ci, et notamment l'orienter dans toutes les directions. Pour réaliser cela, déplacer votre souris vers un angle de la fenêtre.
Vous verrez alors apparaître le curseur suivant : 
En cliquant dessus, vous pouvez ainsi faire tourner le simulateur comme vous le souhaitez.

Une autre manière de manipuler le simulateur est d'utiliser le menu `Simulate > Manipulate Device...`, qui est plus pratique. Il permet notamment de mieux gérer les rotations dans les diverses directions, et de contrôler précisément leur valeur. Voici sur la figure suivante à quoi ressemble cet outil.



Ce contrôleur nous sera particulièrement utile dans la suite, lorsque nous apprendrons à utiliser l'orientation « précise » du smartphone.

Gérer le temps

Présentation

La classe `Timer`

Dans la suite du chapitre, nous allons avoir besoin de gérer le temps afin de rafraîchir l'écran. J'entends par là que nous allons devoir répéter une fonction toutes les x secondes. Ceci est bien entendu réalisable, grâce à la classe `Timer`.
Commençons donc par instancier un objet `Timer` :

Code : Java

```
Timer timer = new Timer();
```

Cette classe peut nous permettre de faire différentes choses liées au temps. La première est de retarder l'exécution d'une **tâche**.
Dans la suite, nous définirons cette tâche par héritage de la classe `TimerTask` ; nous nommerons cette nouvelle classe `MonTimer`.
Ainsi pour retarder cette tâche de 10 secondes, vous pouvez utiliser l'instruction suivante :

Code : Java

```
timer.schedule(new MonTimer(), 10000);
```

Ici le second paramètre de type `long` représente le délai de retard d'exécution de la tâche en *millisecondes*.

L'autre intérêt de la classe `Timer` est de pouvoir répéter une tâche régulièrement.
Ainsi la ligne de code suivante permet d'exécuter la tâche toutes les secondes :

Code : Java

```
timer.scheduleAtFixedRate(new MonTimer(), 1000, 1000);
```

Dans cette expression, le deuxième paramètre correspond au retard de la première exécution de la tâche. Le troisième paramètre, quant à lui, indique l'intervalle de temps entre deux exécutions de cette même tâche.

La classe `TimerTask`

Comme nous l'avons vu, nous allons définir notre tâche par une classe. Celle-ci devra alors hériter de la classe `TimerTask`.
Voici donc la base de notre classe :

Code : Java

```
package mypackage;

import java.util.TimerTask;

public class MonTimer extends TimerTask{

    public MonTimer(){
        super();
        // initialisation de la tâche
    }

    public void run() {
        // Instructions de la tâche
    }

}
```

Dans cette classe, nous devons redéfinir la méthode `run()` qui est alors appelée à chaque exécution de la tâche.
Pour mieux comprendre tout ceci, je vous propose de concevoir un mini-chronomètre dès maintenant.

Création d'un chronomètre

Créer la tâche

Dans cet exercice, nous allons réutiliser la classe `MyScreen` telle que nous l'avions définie auparavant.
Nous allons commencer par créer notre tâche, qui contiendra une référence de l'objet `MyScreen` ainsi qu'un entier qui servira de compteur :

Code : Java

```
    private MyScreen maVue;
    private int compteur;
```

Je vous laisse vous occuper de l'initialisation de ces attributs, pour nous concentrer plutôt sur la méthode `run()`. Comme il s'agit d'un compteur, cette méthode se contentera uniquement d'incrémenter notre entier, et de l'afficher à l'écran.
Voilà donc comment faire :

Code : Java

```
public void run() {
    compteur++;
    maVue.setTexte(compteur + " secondes écoulées...");
}
```

Enfin pour finir, je vous ai réécrit l'intégralité de cette classe `MonTimer` :

Code : Java - MonTimer.java

```
package mypackage;

import java.util.TimerTask;

public class MonTimer extends TimerTask{

    private MyScreen maVue;
    private int compteur;

    public MonTimer(MyScreen vue){
        super();
        maVue = vue;
        compteur = 0;
    }

    public void run() {
        compteur++;
        maVue.setTexte(compteur + " secondes écoulées...");
    }

}
```

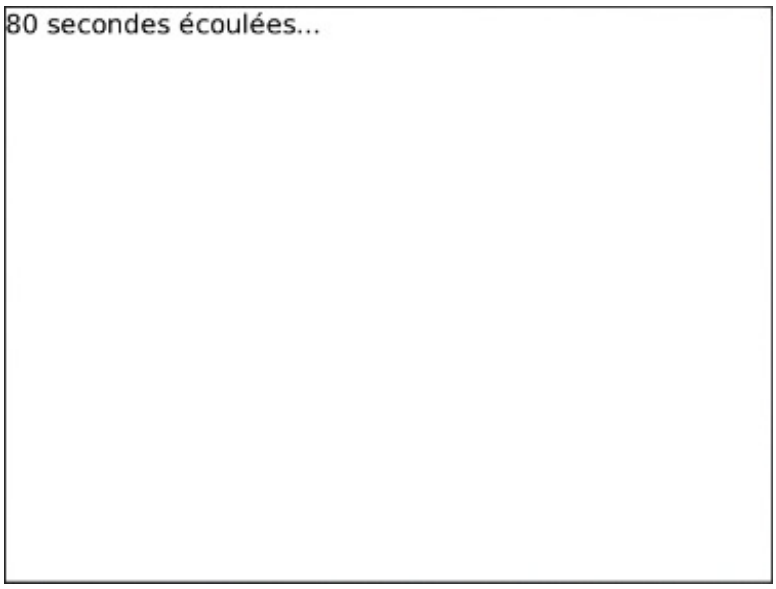
Gérer l'exécution de la tâche

En réalité, la gestion du temps ne se fait réellement qu'avec la classe `Timer`. C'est donc maintenant que nous allons définir l'exécution du compteur toutes les secondes.
Voilà comment j'ai fait :

Code : Java

```
Timer timer = new Timer();
timer.scheduleAtFixedRate(new MonTimer(this), 1000, 1000);
```

Si vous lancez l'application, vous verrez alors apparaître un compteur qui compte toutes les secondes, c'est-à-dire un chronomètre !



L'orientation « précise »
Récupérer l'orientation

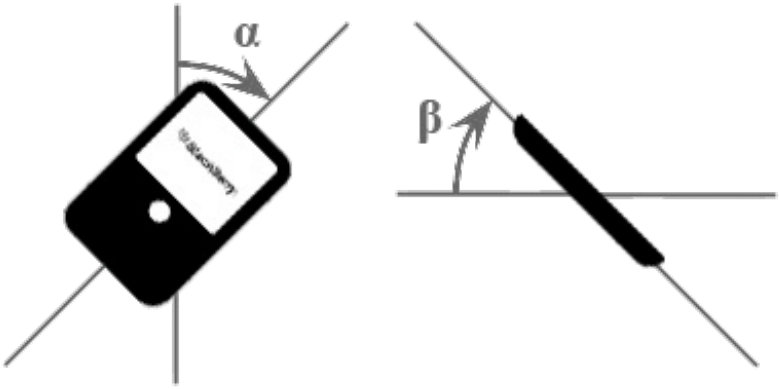
Introduction

Maintenant, nous allons nous concentrer sur la récupération de l'orientation « précise » du smartphone. J'entends par là que nous allons récupérer des données numériques représentant l'orientation de celui-ci. Ces données seront alors composées d'un tableau `xyz` composé de trois valeurs x, y et z.

Pour éviter les explications incompréhensibles, je vous indique directement la valeur de ces données :

$$\begin{cases} x &= 1000 \times \sin(\alpha) \times \sin(\beta) \\ y &= 1000 \times \cos(\alpha) \times \sin(\beta) \\ z &= 1000 \times \cos(\beta) \end{cases}$$

Dans les expressions précédentes, les angles α et β sont définis suivant les figures ci-après.



Grâce à ces valeurs, nous connaissons ainsi l'orientation exacte du smartphone. En utilisant correctement celles-ci, nous sommes alors en mesure de réaliser absolument tout ce que nous souhaitons.
Je vous propose donc de voir comment récupérer ces données, puis nous réaliserons une sorte de « niveau » dans la fin de ce chapitre.

La classe `AccelerometerSensor.Channel`

Comme en début de chapitre, nous allons commencer par déclarer une variable de type `Channel` grâce à la classe `AccelerometerSensor`.
Voici comment faire :

Code : Java

```
Channel rawDataChannel =
AccelerometerSensor.openRawDataChannel(Application.getApplication());
```

Pour récupérer les données décrites précédemment, nous allons devoir déclarer un tableau de type `short[]` que nous nommerons donc `xyz` :

Code : Java

```
short[] xyz = new short[3];
```

Une fois ce tableau déclaré, nous pouvons par la suite lui attribuer les données de l'accéléromètre à l'aide de la méthode `getLastAccelerationData()`. Celle-ci mettra alors notre tableau à jour pour un *instant donné*. C'est donc pour cet appel de méthode que nous aurons besoin de gérer le temps et d'utiliser les classes `Timer` et `TimerTask` que nous venons juste de voir.

Avant de s'occuper de la gestion du temps, je vous laisse découvrir comment mettre à jour notre tableau `xyz` :

Code : Java

```
rawDataChannel.getLastAccelerationData(xyz);
```

Récupérer les données à intervalles réguliers

À présent, nous allons insérer tout ce que nous venons de voir à l'intérieur de notre `TimerTask`, nommée `MonTimer` je vous le rappelle.
Voici donc la classe `MonTimer` que je vous conseille de tester :

Code : Java

```
package mypackage;

import java.util.TimerTask;
import net.rim.device.api.system.AccelerometerSensor;
import net.rim.device.api.system.Application;
import net.rim.device.api.system.AccelerometerSensor.Channel;

public class MonTimer extends TimerTask{

    private MyScreen maVue;
    private Channel rawDataChannel;
    private short[] xyz;

    public MonTimer(MyScreen vue){
        super();
        maVue = vue;
        xyz = new short[3];
        rawDataChannel =
AccelerometerSensor.openRawDataChannel(Application.getApplication());
    }

    public void run() {
        rawDataChannel.getLastAccelerationData(xyz);
        maVue.setXYZ(xyz);
    }

}
```

Comme vous pouvez le voir, cette classe se contente de récupérer les données de l'accéléromètre et les transfère à notre

`MyScreen` via une nouvelle méthode que nous appellerons `setXYZ()`.

Afficher ces données

Revenons à l'intérieur de la classe `MyScreen`, et démarrons en rajoutant un nouvel attribut `xyz` :

Code : Java

```
private short[] xyz;
```

Puis dans un premier temps, je vous propose uniquement d'afficher le contenu de ce tableau dans notre champ de texte :

Code : Java

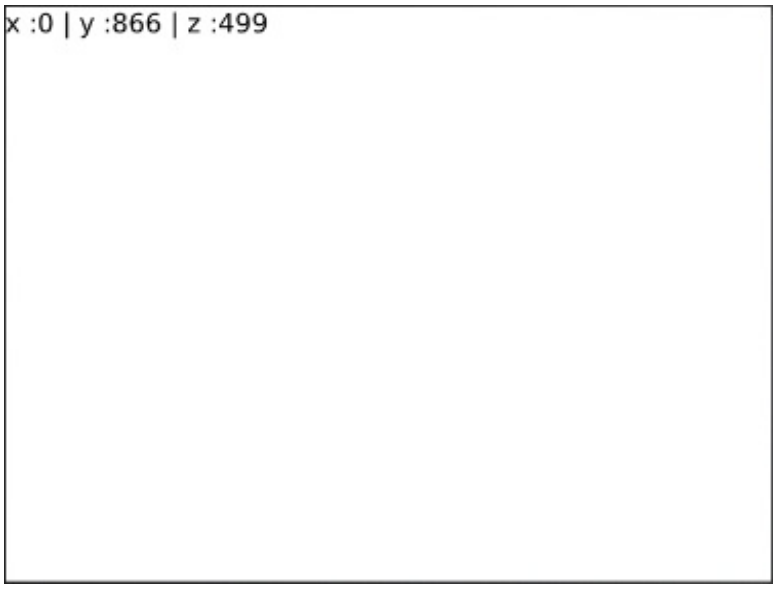
```
public void setXYZ(short[] value){
    setTexte("x :" + xyz[0] + " | y :" + xyz[1] + " | z :" +
xyz[2]);
}
```

Également, nous allons accélérer le rythme d'exécution de notre tâche :

Code : Java

```
timer.scheduleAtFixedRate(new MonTimer(this), 0, 40);
```

Lancez cette application et voyez le résultat en manipulant votre simulateur.



Un niveau

Dessiner le niveau

Maintenant, nous allons rester à l'intérieur de la classe `MyScreen`. Nous commencerons par modifier la méthode `setXYZ()`, puis nous redéfinirons la méthode `paint()` de cette même classe.
Voilà donc notre nouvelle méthode `setXYZ()`, qui se contente de mettre à jour le tableau `xyz` et de rafraîchir l'écran :

Code : Java

```
public void setXYZ(short[] valeur){
```

```
        xyz = valeur;
        invalidate();
    }
```

Ensuite, redéfinissons la méthode `paint()` de la classe `MyScreen` :

Code : Java

```
public void paint(Graphics g){
    super.paint(g);
    // Tracé d'une ligne de niveau
}
```

L'idée ici est de tracer une ligne de « niveau », c'est-à-dire qui soit toujours à l'horizontale quelle que soit l'orientation du smartphone. Le calcul des coordonnées des extrémités de cette ligne se fait donc à partir du centre de l'écran auquel on retranche ou on ajoute des valeurs dépendantes des entiers x et y donnés par l'accéléromètre.
Sans entrer dans les détails de calculs qui sont uniquement géométriques, je vous laisse le soin de recopier ces lignes de code :

Code : Java

```
int x1 = Display.getWidth()/2 - 2*Display.getWidth()*xyz[1]/1000;
int y1 = Display.getHeight()/2 - 2*Display.getWidth()*xyz[0]/1000;
int x2 = Display.getWidth()/2 + 2*Display.getWidth()*xyz[1]/1000;
int y2 = Display.getHeight()/2 + 2*Display.getWidth()*xyz[0]/1000;
```

Nous pouvons à présent tracer le niveau que j'ai en fait réalisé en superposant trois lignes pour lui donner plus d'épaisseur. J'en ai également profité pour changer la couleur de cette ligne :

Code : Java

```
g.setColor(0x000088FF);
g.drawLine(x1, y1, x2, y2);
g.drawLine(x1, y1-1, x2, y2-1);
g.drawLine(x1, y1+1, x2, y2+1);
```

La classe `MyScreen` complète

Comme d'habitude, je ne délaisse pas ceux qui auraient plus de difficultés en vous donnant le code intégral de la classe `MyScreen` :

Code : Java

```
package mypackage;

import java.util.Timer;
import net.rim.device.api.system.Display;
import net.rim.device.api.ui.Graphics;
import net.rim.device.api.ui.container.MainScreen;

public class MyScreen extends MainScreen{

    private Timer timer;
    private short[] xyz;

    public MyScreen(){
        super();
        xyz = new short[3];
        timer = new Timer();
```

```
            timer.scheduleAtFixedRate(new MonTimer(this), 0, 40);
        }

        public void paint(Graphics g){
            super.paint(g);
            g.setColor(0x000088FF);         g.setDrawingStyle(0, true);
            int x1 = Display.getWidth()/2 -
    2*Display.getWidth()*xyz[1]/1000;
            int y1 = Display.getHeight()/2 -
    2*Display.getWidth()*xyz[0]/1000;
            int x2 = Display.getWidth()/2 +
    2*Display.getWidth()*xyz[1]/1000;
            int y2 = Display.getHeight()/2 +
    2*Display.getWidth()*xyz[0]/1000;
            g.drawLine(x1, y1, x2, y2);
            g.drawLine(x1, y1-1, x2, y2-1);
            g.drawLine(x1, y1+1, x2, y2+1);
        }

        public void setXYZ(short[] value){
            xyz = value;
            invalidate();
        }
    }
```

Résultat de l'animation

Une fois l'application lancée, voyez le résultat de cette animation en manipulant votre simulateur.

- L'**accéléromètre** est un capteur qui nous permet de récupérer des données concernant l'orientation du smartphone.
- Une orientation « grossière » peut être établie en utilisant les constantes de la classe `AccelerometerSensor`.
- La gestion du temps se fait à l'aide de la classe `Timer` qui sert à contrôler l'exécution d'une **tâche**.
- Une tâche est décrite à l'aide d'une classe devant hériter de la classe `TimerTask`.
- L'orientation « précise » du smartphone peut être connue via la méthode `getLastAccelerationData()` de la classe `AccelerometerSensor.Channel`.

Les pages HTML

Pour le dernier chapitre de cette partie, nous allons nous intéresser aux pages HTML. Je suis sûr que certains d'entre-vous attendez depuis longtemps ce moment ! Nous allons donc voir comment afficher du HTML à l'intérieur d'une application ; cela peut être du code que vous aurez vous-mêmes créé ou bien une page d'un site web.
Vous devez tout de même savoir que pour charger du contenu web depuis l'intérieur d'une application, il est nécessaire d'installer une fonctionnalité supplémentaire. Dans un premier temps, nous nous contenterons donc d'afficher du code HTML créé par nous-mêmes. Puis après installation de l'outil en question, nous pourrons charger du contenu depuis le web !

Afficher des pages HTML

Créer du texte HTML

Écriture du code

Comme je vous l'ai dit en introduction, pour l'instant nous allons utiliser du code créé par nos soins. Sachant qu'il ne s'agit pas d'un cours sur l'HTML, je vous propose directement un code que nous testerons juste après.
Voici donc le code HTML en question :

Code : HTML

```
<html>
    <body>
        <h1>Bonjour à tous les Zéros !</h1>
    </body>
</html>
```

Pour ceux qui n'ont aucune notion en HTML, sachez que nous avons ici simplement défini un titre de « niveau 1 » à l'intérieur d'une page HTML. Si vous souhaitez en apprendre davantage sur ce langage, je vous conseille de suivre le cours Apprenez à créer votre site web avec HTML5 et CSS3 de Mathieu Nebra disponible sur le Site du Zéro !

Dans notre cas, le code HTML doit être de type `String` :

Code : Java

```
String texteHTML = "<html><body><h1>Bonjour à tous les Zéros
!</h1></body></html>";
```

L'objet `BrowserField`

Comme vous pouvez l'imaginer en lisant le titre, la classe `BrowserField` permet d'afficher des pages ou du code HTML à l'intérieur d'une application.
Voici donc comment créer un composant de ce type :

Code : Java

```
BrowserField page = new BrowserField();
add(page);
```

Enfin, il ne nous reste plus qu'à afficher le texte HTML défini plus haut. Pour cela, nous utiliserons alors la méthode `displayContent()` de la classe `BrowserField`.
Voici le code correspondant :

Code : Java

```
page.displayContent(texteHTML, "http://localhost");
```

Pour afficher du code HTML directement entré en tant que texte, il est nécessaire de définir le second paramètre avec "http://localhost".

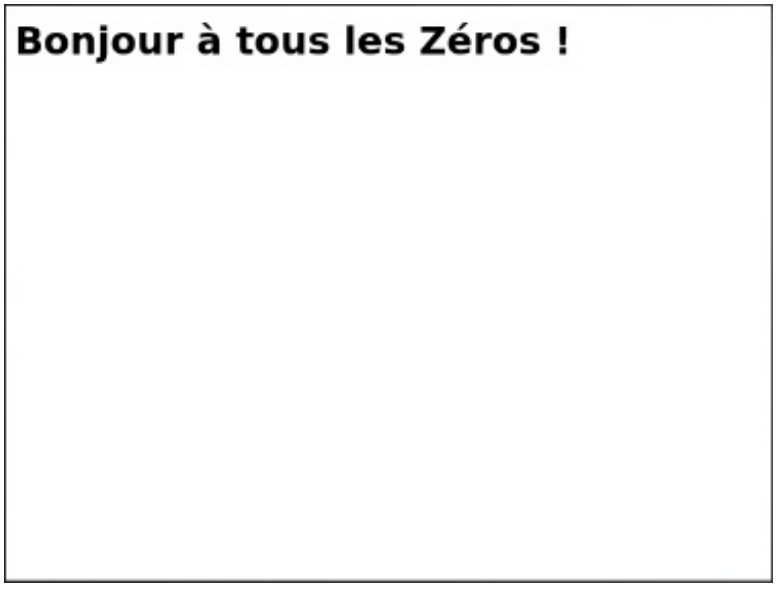
Résultat du code

Le code n'est ici pas très compliqué, puisqu'il se résume à quatre instructions que voilà :

Code : Java

```
BrowserField page = new BrowserField();
add(page);
String texteHTML = "<html><body><h1>Bonjour à tous les Zéros
!</h1></body></html>";
page.displayContent(texteHTML, "http://localhost");
```

Une fois l'application lancée, voici sur la figure suivante ce que nous obtenons à l'écran.



Charger une page locale

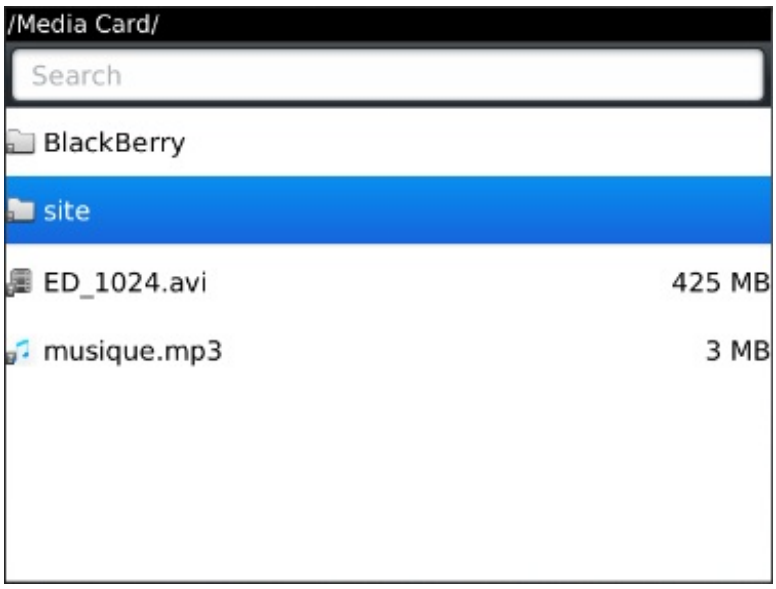
Téléchargement des sources

À présent, nous allons tester un code un peu plus volumineux. C'est pourquoi nous allons prendre le code d'un site déjà créé, c'est-à-dire tout prêt. Rien de mieux que le Site du Zéro, je vous propose de télécharger les sources du TP du cours Apprenez à créer votre site web avec HTML5 et CSS3 de Mathieu Nebra.
Voici un aperçu de la page d'accueil du site en question :

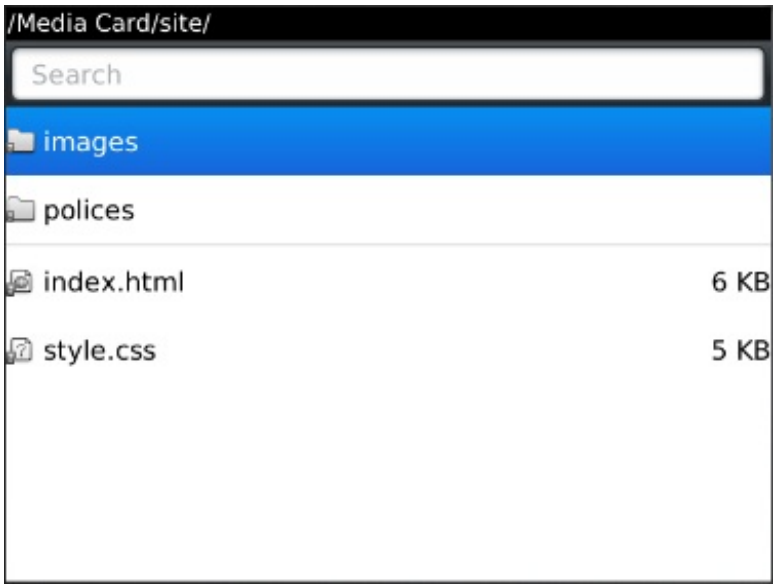
Je vous invite ainsi à télécharger les sources directement sur le Site du Zéro.

Mise en place du site

L'objectif maintenant est de charger une page HTML en local sur le smartphone depuis notre application. Il va donc nous falloir placer ces sources à l'intérieur de la mémoire du smartphone. Vous l'aurez deviné, nous allons utiliser la carte mémoire externe. Ajoutons ainsi un dossier `site` dans notre dossier de partage, dans lequel nous allons placer l'ensemble des fichiers du site :



Si vous n'avez pas fait de bêtises, voici ce que contient notre dossier `site` :



Afficher la page

Maintenant nous allons voir comment charger une page externe à l'application.
Pour commencer, créons notre objet `BrowserField` :

Code : Java

```
BrowserField page = new BrowserField();
add(page);
```

Cette fois, nous allons utiliser la méthode `requestContent()` à laquelle nous renseignerons l'URL de la page. Pour l'instant nous chargeons une page locale depuis la carte mémoire. C'est pourquoi l'URL démarrera par `"file:///SDCard/"` comme nous avons vu dans un chapitre précédent.
Voilà donc l'instruction de chargement de la page :

Code : Java

```
page.requestContent("file:///SDCard/site/index.html");
```

Comme vous pouvez le voir ci-dessous, le site s'affiche bien dans l'application :

En réalité, le site est plus large que l'écran. Dans l'image précédente, vous constaterez alors qu'il manque une partie de la page sur la droite. D'ailleurs vous ne pouvez, pour l'instant, pas y accéder. C'est normal, notre conteneur dispose uniquement d'un défilement vertical.
Nous allons donc remédier au problème en ajoutant un défilement horizontal à notre conteneur principal :

Code : Java

```
super(Manager.HORIZONTAL_SCROLL);
```

Voilà, nous avons à présent l'intégralité de la page disponible dans notre application, comme le montre la figure ci-dessous :

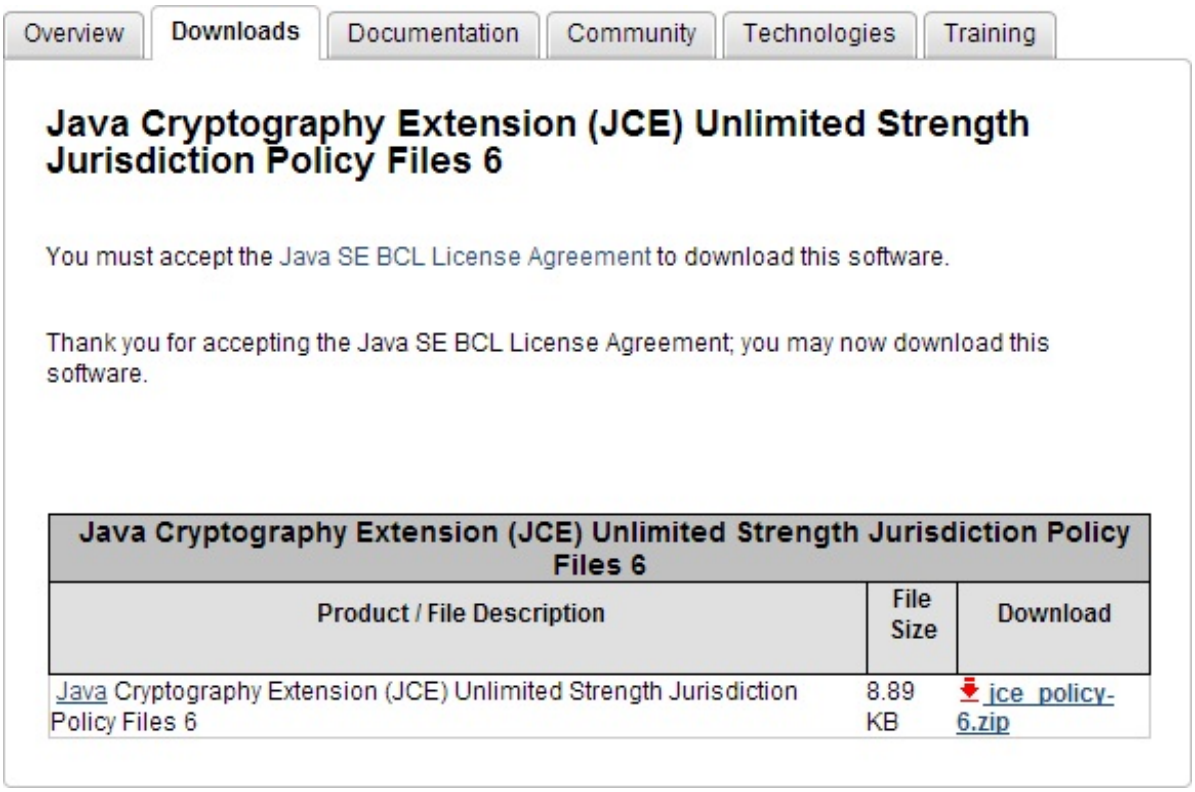
Comme je vous l'ai dit en introduction, nous ne pouvons pour l'instant pas charger des pages web. Je vous propose donc maintenant d'installer les outils nécessaires pour contourner ce problème !

Installer le BlackBerry MDS Simulator

À présent, nous allons installer l'outil nécessaire pour accéder aux pages web : le **BlackBerry MDS Simulator** !

Mise en place du JCE

Avant d'installer le MDS Simulator, il est tout d'abord nécessaire de mettre en place une API de chiffrement nommée JCE (ou Java Cryptogaphy Extension) Unlimited Strength. Celle-ci est disponible sur le site d'oracle à cette adresse.

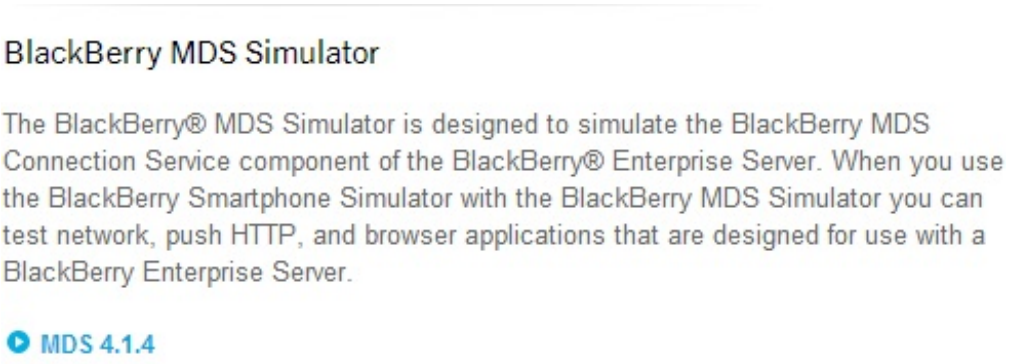


Téléchargez donc l'archive proposée, avant d'y extraire les fichiers.
Maintenant, il faut alors copier les fichiers `local_policy.jar` et `US_export_policy.jar`, pour remplacer ceux déjà présents dans votre dossier d'installation de Java à l'endroit suivant : `\Java\jdk1.7.0_02\jre\lib\security`.

Le BlackBerry MDS Simulator

Téléchargement

Nous allons maintenant enfin pouvoir nous occuper du MDS Simulator. Pour cela, rendez-vous sur la page suivante, où vous trouverez un lien de téléchargement en *bas de la page*, tel qu'il est présenté ci-dessous :

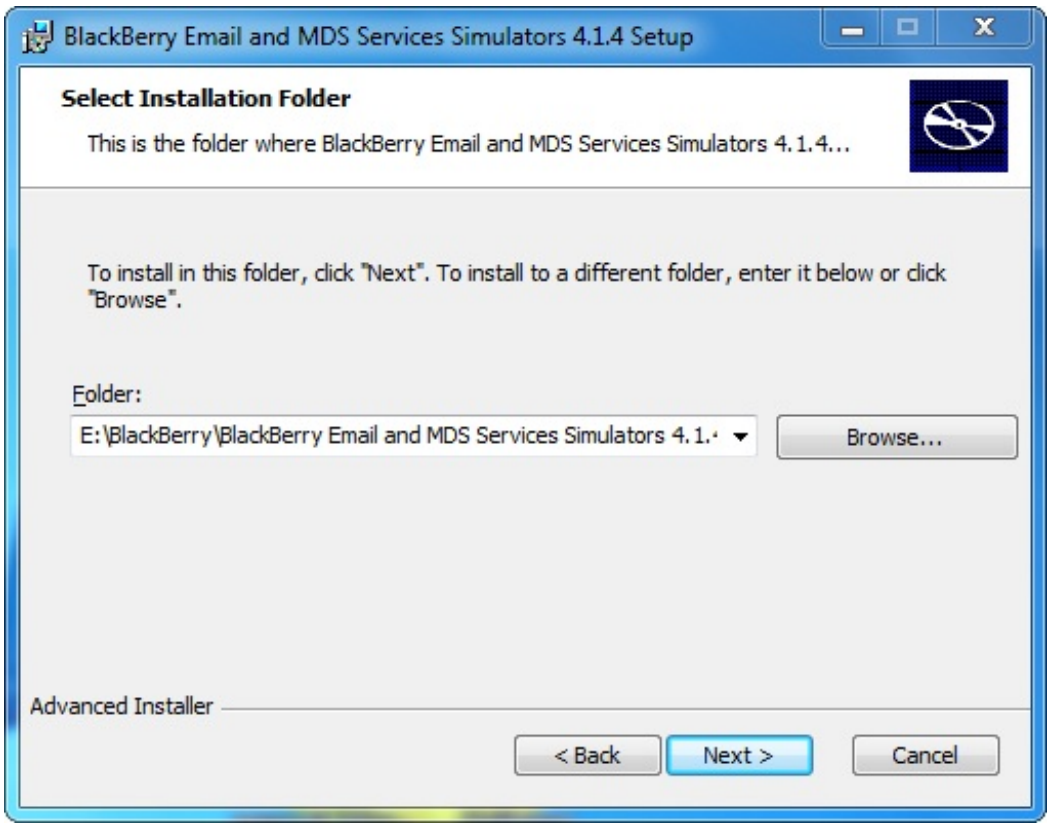


Dans mon cas je n'avais pas le choix, mais téléchargez la dernière version proposée. Vous serez alors redirigés vers une autre page semblable à la suivante :

Téléchargez donc le BlackBerry MDS Simulator, puis passez à l'installation.

Installation

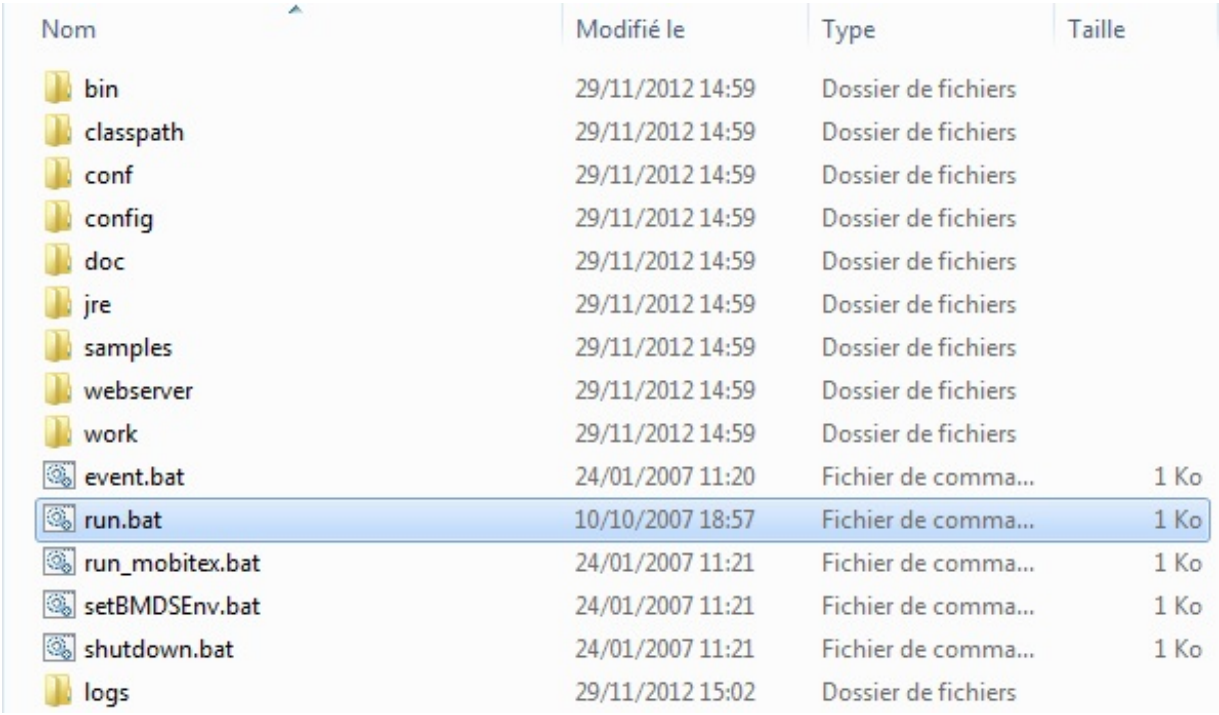
Aucune difficulté n'est présente durant l'installation. Toutefois, veuillez bien noter l'emplacement où celle-ci est réalisée :



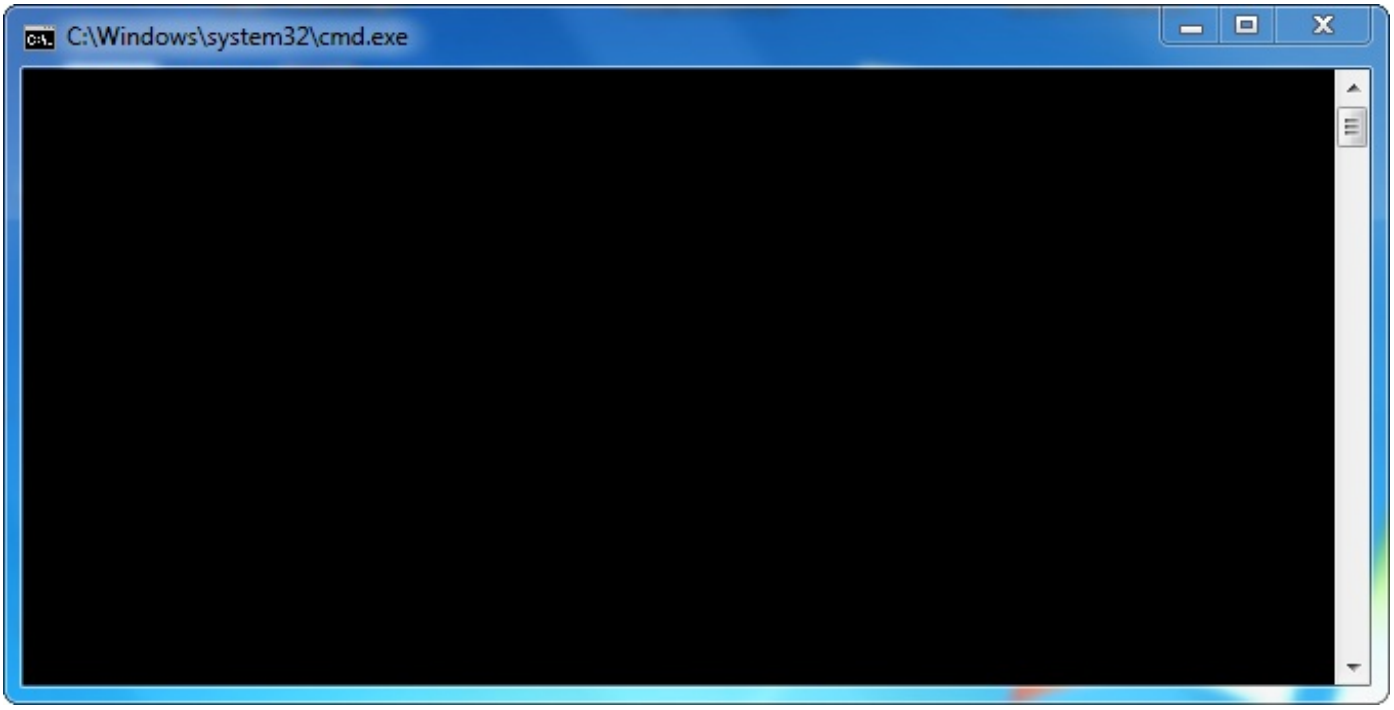
Exécution

Une fois l'installation terminée, nous allons devoir lancer l'exécution du MDS Simulator nous-mêmes. C'est pourquoi nous allons donc nous rendre dans le dossier d'installation de ce dernier, et plus précisément à l'emplacement suivant :

`BlackBerry Email and MDS Services Simulators 4.1.4\MDS\`.
Vous trouverez alors tout un tas de fichiers, et parmi eux un certain fichier nommé `run.bat` :



Double-cliquez sur ce fichier, et vous verrez une fenêtre de console s'ouvrir :



Le MDS Simulator est actif uniquement lorsque la console est ouverte. Ainsi vous devrez donc la rouvrir à chaque fois que vous en aurez besoin !

Charger des pages web

Charger une page web

Maintenant que nous avons installé le MDS Simulator, nous allons pouvoir charger des pages externes au smartphone, c'est-à-dire depuis Internet.
Pour cela rien de plus simple, il nous suffit de réutiliser la méthode `requestContent()` en utilisant cette fois une adresse avec un protocole HTTP :

Code : Java

```
page.requestContent("http://www.google.com");
```

Voilà donc le peu d'instructions nécessaires à l'affichage d'une page web à l'intérieur d'une application :

Code : Java

```
BrowserField page = new BrowserField();
add(page);
page.requestContent("http://www.google.com");
```

En utilisant le code précédent, nous voyons alors le célèbre moteur de recherche s'afficher :

Lorsque vous chargez du contenu web à l'intérieur de votre application, le temps de chargement peut être plus ou moins long. Pour ce type d'applications, n'hésitez pas à faire des essais sur un terminal physique.

Modifier les configurations

Avant de terminer ce chapitre, nous allons voir comment nous servir de la classe `BrowserFieldConfig` pour configurer nos pages. Grâce aux constantes de cette classe, nous pourrons par exemple autoriser l'exécution de code Javascript à l'aide de `JAVASCRIPT_ENABLED`, ou encore autoriser les cookies avec `ENABLE_COOKIES`.
Comme exemple, nous allons utiliser les constantes `NAVIGATION_MODE` et `NAVIGATION_MODE_POINTER` pour afficher un curseur à l'intérieur de notre composant `BrowserField`.
Voilà donc comment faire ceci au niveau du code :

Code : Java

```
BrowserFieldConfig config = new BrowserFieldConfig();
config.setProperty(BrowserFieldConfig.NAVIGATION_MODE,
BrowserFieldConfig.NAVIGATION_MODE_POINTER);
BrowserField page = new BrowserField(config);
add(page);
page.requestContent("http://www.google.com");
```

Nous obtenons ainsi la même page que précédemment, avec toutefois un curseur en plus, comme vous pouvez le voir ci-dessous :

- Le composant permettant d'afficher des pages HTML est `BrowserField`.
- Pour afficher du code HTML « brut », il faut utiliser la méthode `displayContent()`.
- Il est possible d'afficher des pages locales ou web grâce à la méthode `requestContent()`.
- Le chargement d'une page web nécessite l'installation d'un outil approprié : le **BlackBerry MDS Simulator**.
- La classe `BrowserFieldConfig` permet de configurer un objet `BrowserField` et ainsi mieux contrôler la navigation.

Partie 4 : Les finitions

Tester son application en profondeur

Lorsqu'une application est terminée, il est important de pouvoir voir comment celle-ci réagit suivant le smartphone utilisé. Dans ce chapitre, nous allons donc voir un certain nombre de choses. Dans un premier temps, nous verrons comment bien préparer son application avant cette grande série de tests et avant sa publication officielle. Ensuite nous verrons où trouver de nouveaux simulateurs et comment les utiliser dans Eclipse. Enfin, nous verrons comment tester une application directement sur un terminal physique, grâce au logiciel **BlackBerry Desktop Manager** que vous possédez peut-être déjà !

Préparer son application

Avant tout, il faut que vous sachiez que votre application sera distribuée et installée sous la forme d'un fichier `.alx`. Ce format de fichiers est spécifique à BlackBerry, et votre application devra être compilée sous celui-ci. En réalité, toute compilation de votre projet crée automatiquement un fichier `.alx`. Ce dernier est alors disponible dans le sous-répertoire `..\deliverables\Standard\` de votre projet. Ce fichier `.alx` sera également utile pour tester votre application sur un appareil physique avant une publication.

Tester son application sur divers appareils

Ce chapitre a pour principal objectif de vous sensibiliser et de vous initier aux tests finaux *indispensables* à votre application, avant de pouvoir la déployer. Une fois une application terminée, nous avons souvent tendance à vouloir précipiter sa publication, ce qui est une grave erreur.

En effet, si votre application fonctionne bien sur votre simulateur, celle-ci peut très bien générer des erreurs lors de son passage sur un smartphone physique. Il est donc nécessaire de la tester sur un terminal physique et même de voir sa compatibilité avec différents modèles et différentes versions de l'OS. L'idéal serait évidemment de pouvoir tester cette dernière sur une large gamme de smartphones. Toutefois ceci n'est pas envisageable, surtout pour des développeurs amateurs et indépendants. Une alternative consiste alors à utiliser le simulateur en variant au maximum le modèle utilisé. Vous devez savoir que les problèmes viendront le plus souvent de l'affichage, c'est pourquoi vous devrez utiliser un maximum d'écrans différents.

Au cours de ce chapitre, nous verrons donc comment utiliser de nouveaux simulateurs et comment transférer et installer son application sur un appareil physique.

Quelques actions préalables

Avant de vous lancer tête baissée, je vais ici vous présenter quelques actions utiles, principalement dans le cas d'essais sur un terminal physique.

Nettoyer son projet

Lors de la phase de conception, nous avons souvent besoin de retravailler plusieurs fois son application. C'est pourquoi à la fin du projet, nous nous retrouvons très régulièrement avec des « inutilités » dans ce genre :

- fonctions ou méthodes non utilisées dans la version finale
- classes de tests
- fichiers (images, sons,...) inutiles au projet final.

Le but est donc de supprimer tous ces éléments pour « alléger » le fichier `.alx`, en pensant bien à faire une copie au préalable pour ne perdre aucune donnée importante. Ce nettoyage va surtout être utile dans le cas d'essais sur un terminal physique, car cela va vous permettre de réduire au maximum la taille de votre application et ainsi éviter de surcharger ce dernier.

Changer d'icône

En général après un certain temps de développement, nous nous sommes habitués à l'icône par défaut et n'y prêtons plus attention. Néanmoins, n'oubliez pas de dessiner une icône appropriée à votre application avant la publication et le déploiement final ; c'est donc l'occasion de s'y intéresser. Ceci est d'autant plus important maintenant que nous nous apprêtons à tester notre application sur une grande variété de terminaux et simulateurs. Ainsi pour éviter de vous mélanger les pinceaux parmi les diverses applications que vous testerez, il est préférable de changer dès maintenant d'icône, ce qui vous permettra de mieux les distinguer.

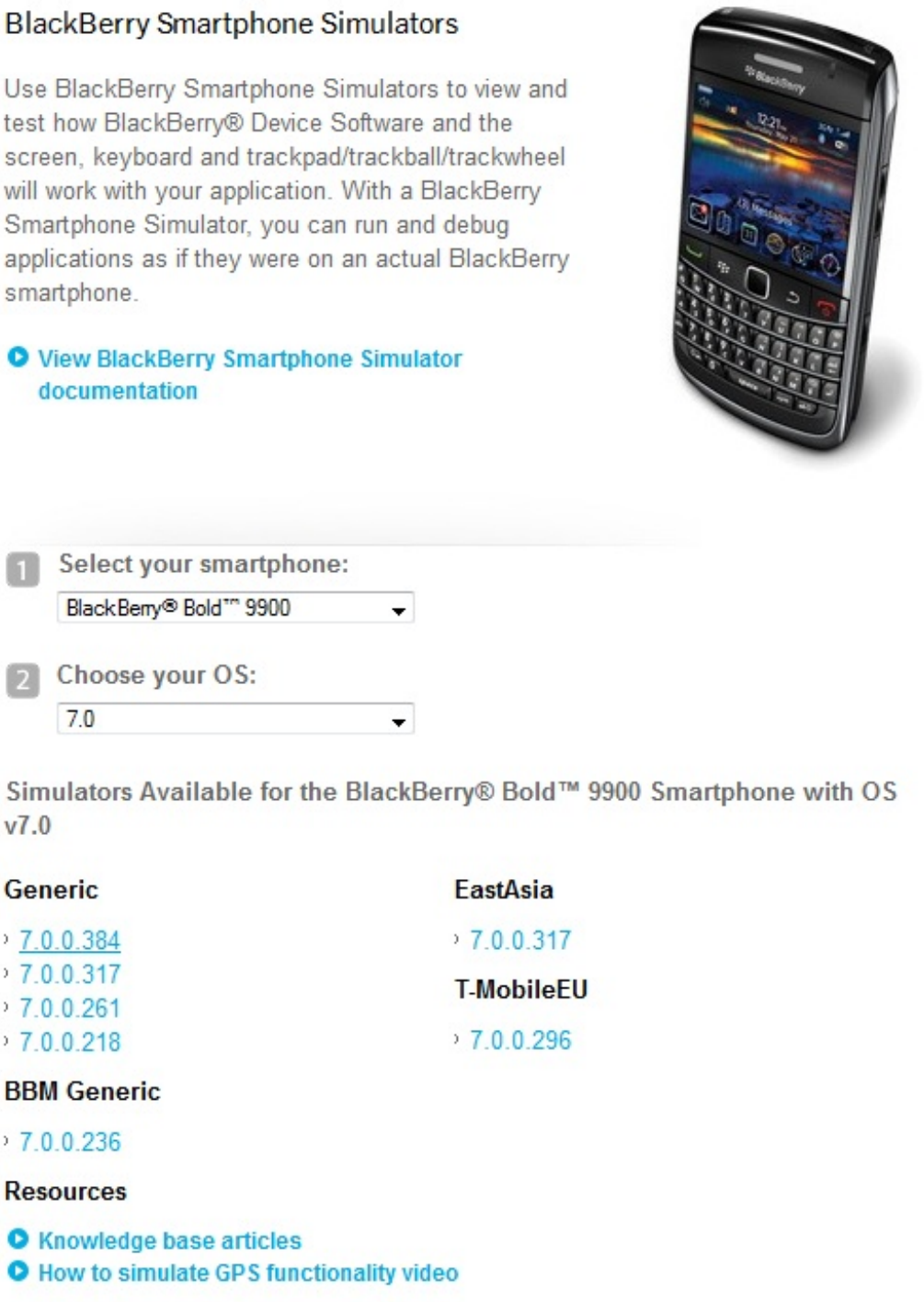
Recompiler son projet

Enfin avant de passer à la suite, n'oubliez pas de recompiler votre projet. Cette opération vous servira principalement à incorporer les manipulations précédentes à votre application. L'objectif de ces rectifications, est de tester votre application au plus proche possible de sa version finale définitive.
Pour compiler un projet « manuellement » dans Eclipse, vous devez décocher l'option `Build Automatically` du menu `Project`. La compilation se fait alors en sélectionnant le projet concerné, puis en cliquant sur `Project > Build Project`.

Utiliser de nouveaux simulateurs

Télécharger et installer un nouveau simulateur

Comme vous avez certainement pu l'imaginer, il est possible de tester son application sur un autre simulateur que celui proposé par défaut. Cela permet notamment de s'assurer de la compatibilité de votre application sur l'ensemble des supports.
Pour cela, RIM vous propose l'ensemble de ses produits sur la page de téléchargement des simulateurs. Pour l'exemple, nous prendrons le simulateur du BlackBerry Bold 9900. Il vous sera également demandé de spécifier votre OS, et sachant que nous travaillons depuis le départ avec la version 7.0, je vous conseille donc de prendre cette version.
Enfin, prenez la dernière version de type `Generic`, comme indiqué sur la figure suivante.



Une fois le téléchargement achevé, lancez l'installation du simulateur. Notez bien cependant l'emplacement où celui-ci sera installé, car nous en aurons besoin pour la suite !

Configurer Eclipse

Une fois que le nouveau simulateur est installé, celui-ci n'est malheureusement pas encore visible dans Eclipse. C'est pourquoi il faut faire quelques manipulations supplémentaires. Je vous propose donc de vous rendre dans le dossier d'installation d'Eclipse et plus particulièrement dans le dossier suivant : `..\Eclipse\plugins\net.rim.ejde.componentpackXXXXXX\components\simulator\`. À l'intérieur, vous devriez trouver un fichier nommé `SimPackage-JDE.rc`.
Ouvrez donc celui-ci avec un éditeur de texte quelconque tel que le Bloc Note, vous trouverez alors à l'intérieur les lignes ci-dessous :

Code : Autre

```
## RIM Java Development Environment
```

```
# Settings file
SimulatorCommand9930-
JDE=<install_dir>\fledge.exe /app=Jvm.dll /handheld=9930 /session=9930 /app-
param=DisableRegistration /app-
param=JvmAlxConfigFile:9930.xml /pin=0x2100000A  /app-param=NoTutorial
SimulatorDirectory9930-JDE=<install_dir>
```

Comme vous pouvez le voir, les deux dernières lignes permettent de spécifier l'emplacement du simulateur par défaut. Vous constaterez également que le nouveau simulateur n'y est pas, et n'est donc pas pris en compte par Eclipse.
Ainsi pour l'ajouter, il suffit de copier les deux dernières lignes et de les coller juste en dessous. Ensuite vous devez remplacer `<install_dir>` par l'emplacement où est stocké le fichier `fledge.exe` de notre nouveau simulateur. Également vous devez mettre à jour le nom du simulateur, ici 9900 à la place de 9930.
Au final, voici ce que donne mon fichier après l'ajout du simulateur 9900 :

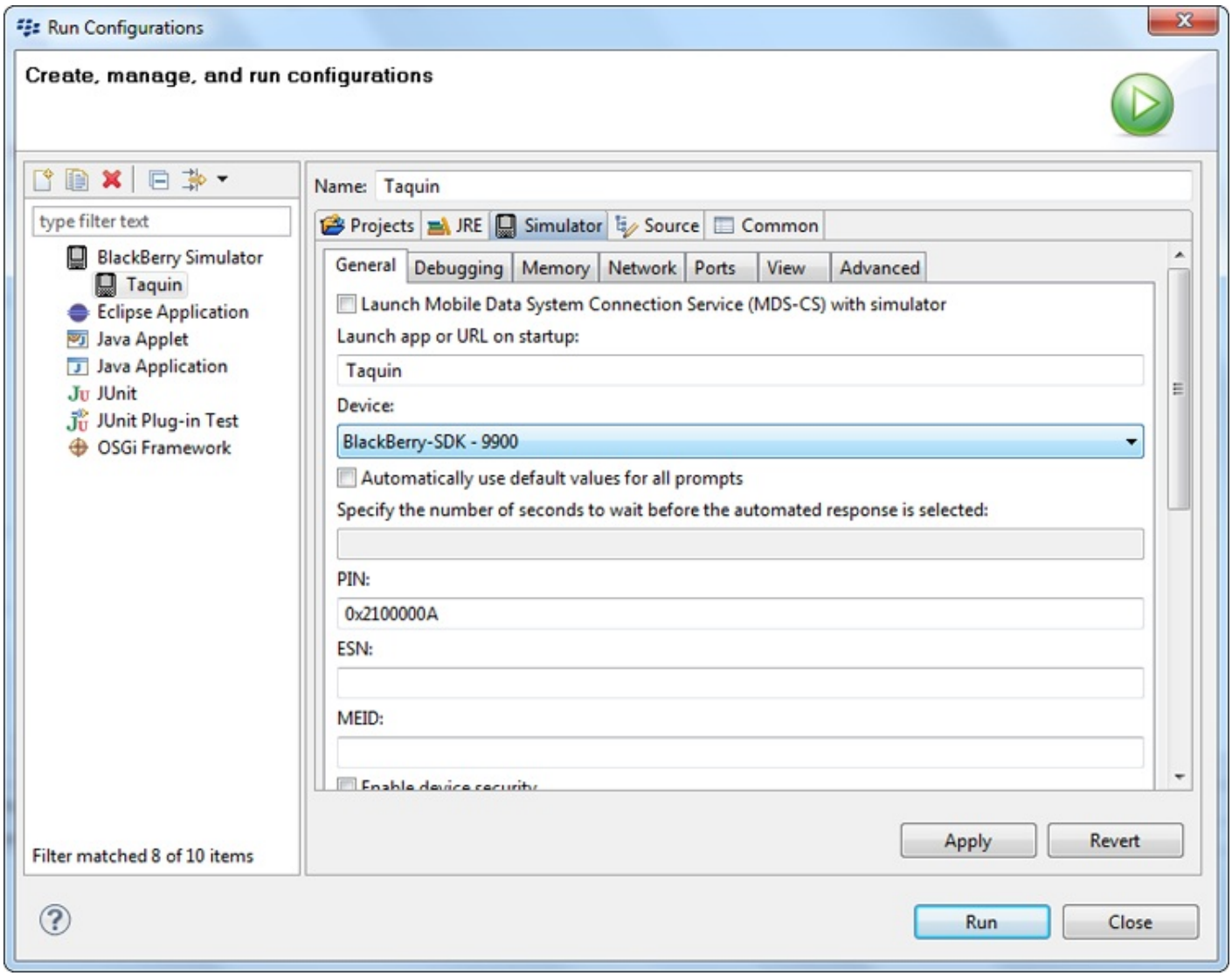
Code : Autre

```
## RIM Java Development Environment
# Settings file
SimulatorCommand9930-JDE=<install_dir>\fledge.exe /app=Jvm.dll /handheld=9930 /se
param=JvmAlxConfigFile:9930.xml /pin=0x2100000A  /app-param=NoTutorial
SimulatorDirectory9930-JDE=<install_dir>

SimulatorCommand9900-
JDE=E:\BlackBerry\BlackBerry Smartphone Simulators 7.0.0\7.0.0.384 (9900)\fledge.
param=DisableRegistration /app-param=JvmAlxConfigFile:9900.xml /pin=0x2100000A  /
SimulatorDirectory9900-JDE=E:\BlackBerry\BlackBerry Smartphone Simulators 7.0.0\7
```

Maintenant tout est prêt, il ne nous reste plus qu'à spécifier dans Eclipse quel simulateur utiliser pour nos projets.

Lancer l'application

Une fois la manipulation précédente effectuée, Eclipse prend en compte notre nouveau simulateur et nous pouvons maintenant choisir le simulateur que nous souhaitons pour tester nos applications. Pour cela, allez dans `Run > Run Configurations...`, sélectionnez un projet puis cliquez sur l'onglet `Simulator`. Enfin vous y trouverez une liste déroulante de l'ensemble des simulateurs.
Sélectionnez donc notre nouveau simulateur nommé `BlackBerry-SDK-9900` :



Ça y est !
Vous pouvez désormais lancer votre application dans le nouveau simulateur. Notez que vous pouvez à tout moment changer de simulateur pour n'importe quelle application.

Utiliser un appareil physique

Installer le Desktop Manager

Voyons maintenant comment transférer et installer nos applications sur un « vrai » smartphone. Pour faire ça, nous aurons besoin du **BlackBerry Desktop Manager**, qui est un logiciel permettant de gérer son smartphone depuis son ordinateur. Grâce à lui, il est possible de mettre à jour son système d'exploitation, récupérer des fichiers multimédias présents sur l'appareil, ou encore installer de nouvelles applications !
Il est possible que vous ayez déjà cet outil. Dans le cas contraire, je vous invite à télécharger ce logiciel sur cette page, puis à l'installer.

Je ne détaillerai pas les étapes d'installation du Desktop Manager à ce stade du cours. Je considère donc que vous êtes tout à fait capables de suivre la procédure par vous-mêmes.

Premier démarrage du logiciel

Ouverture du Desktop Manager

Une fois que votre installation est terminée, lancez l'exécution du logiciel. L'ouverture de ce dernier peut être plus ou moins longue, c'est pourquoi ne vous étonnez pas si vous restez bloqués un petit moment sur la page de présentation suivante.

Au premier démarrage, il est probable que vous tombiez sur une page de configuration dans ce genre (voir la figure suivante).

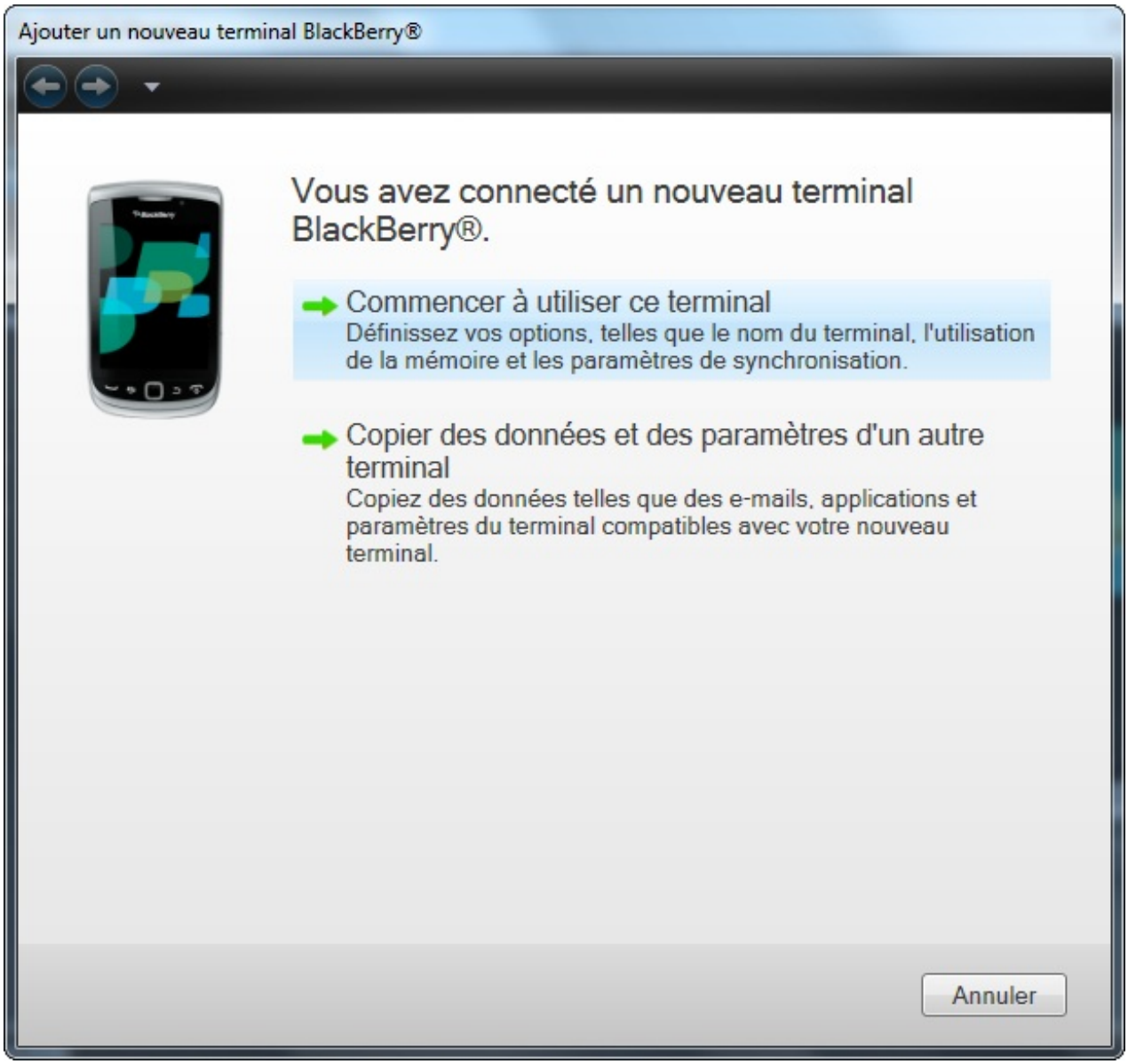
Ces configurations sont plutôt des choix personnels, je vous laisserai donc le soin de définir les paramètres selon vos souhaits. Une fois ouvert, le Desktop Manager ressemble à ceci :

Notez toutefois qu'il nous est pour l'instant d'aucune utilité tant que nous n'avons pas connecté un appareil.

Connexion au terminal

Si vous avez jeté un œil aux paramètres de configuration décrits plus haut, vous aurez peut-être remarqué qu'il est possible de connecter votre smartphone en **Bluetooth**. Néanmoins, ce mode de connexion ne permet pas l'installation d'applications. Ainsi nous allons plutôt utiliser un raccordement en USB.
Branchez donc votre appareil à l'aide du câble USB fourni avec celui-ci. Une fois branché, vous devriez voir apparaître la bulle suivante.

Attendez jusqu'à ce que l'installation du périphérique soit achevée, puis lancez le Desktop Manager si ce n'est pas déjà fait !
Une fois le périphérique prêt à être utilisé, le logiciel vous fera savoir qu'il a détecté l'appareil que vous venez de brancher par la fenêtre suivante.

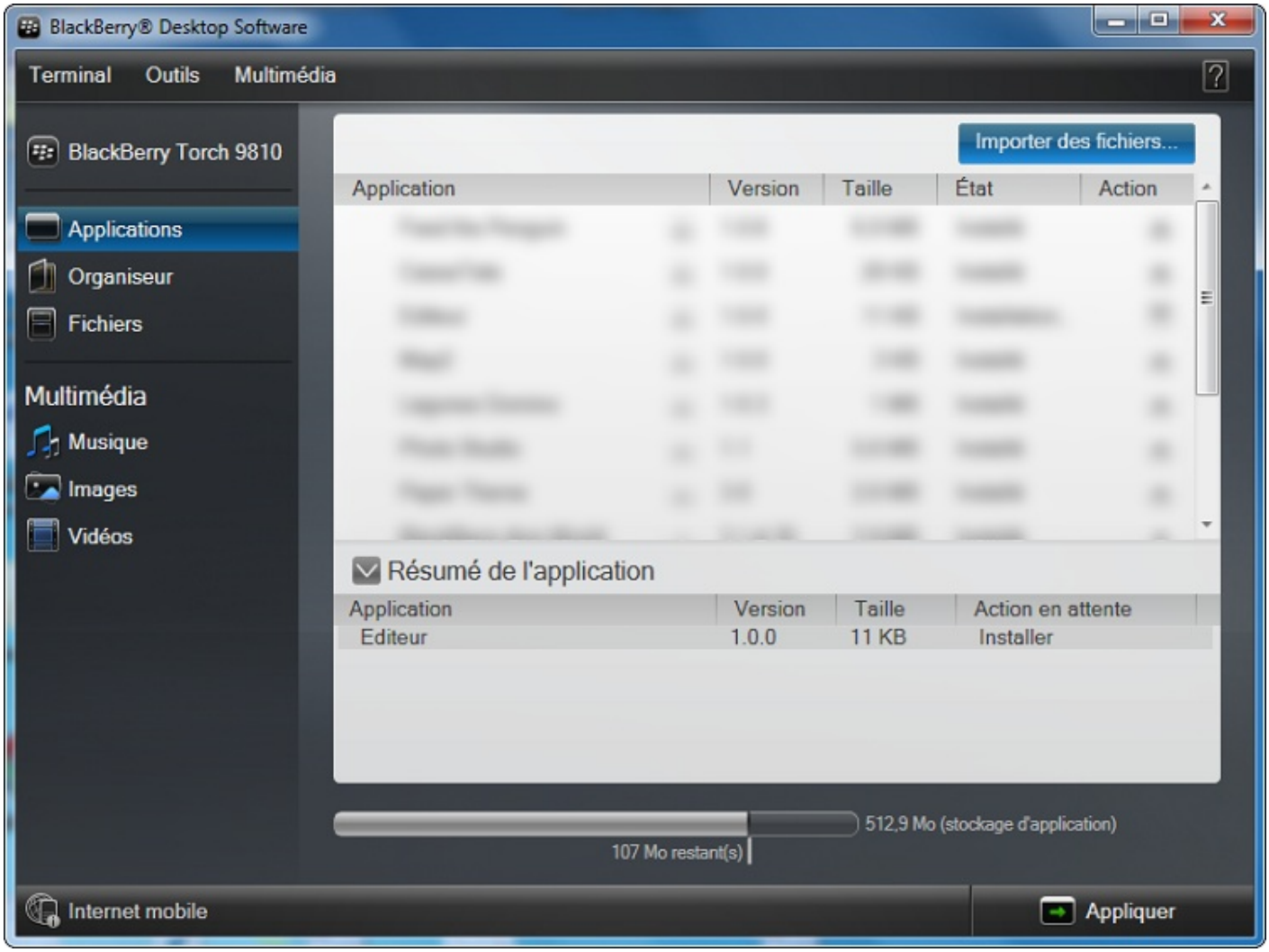


Cliquez alors sur le bouton `Commencer à utiliser ce terminal` ; nous sommes à présent prêts pour installer et tester une application sur le smartphone !

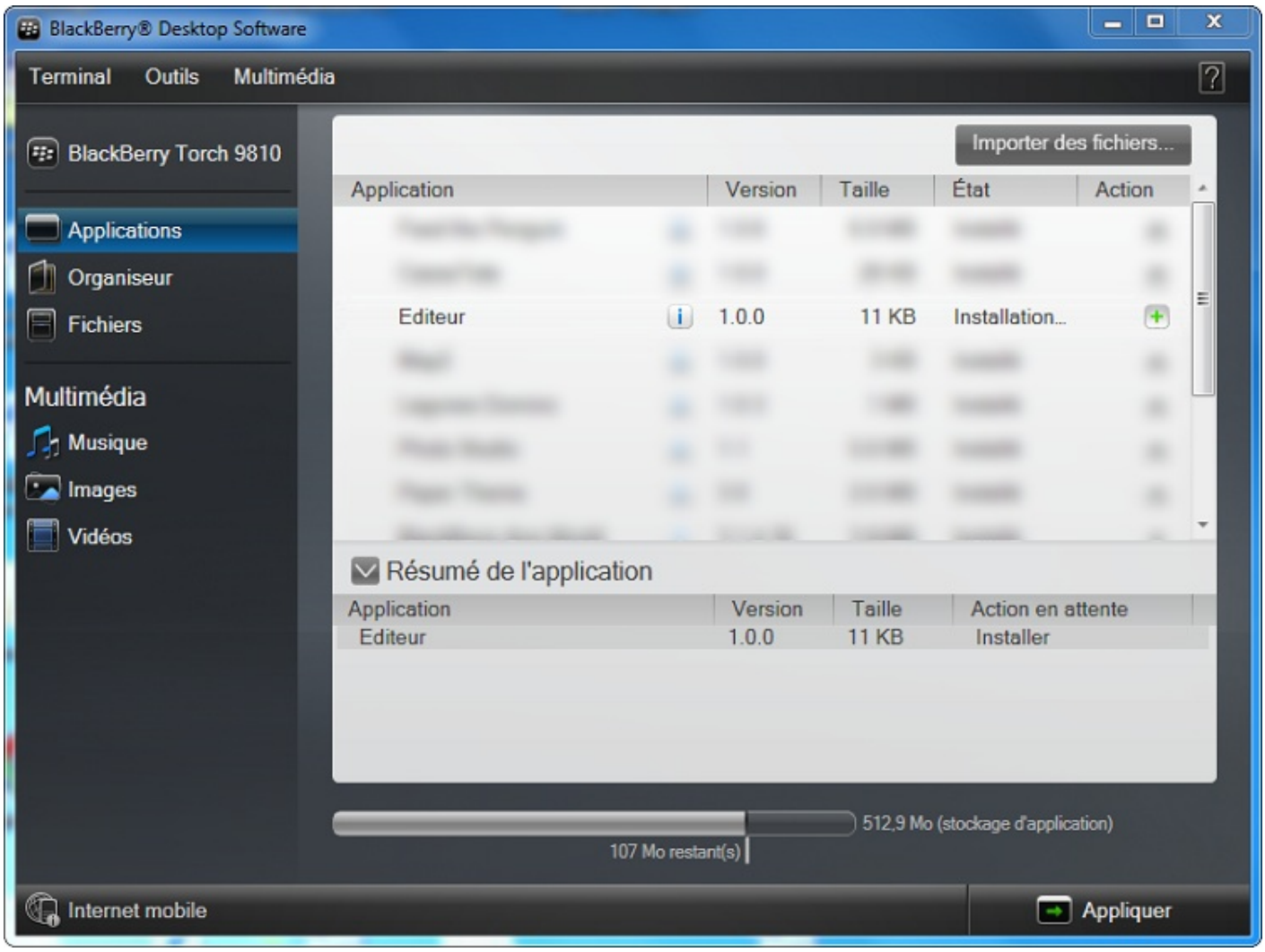
Installation d'une application

Comme vous l'aurez deviné, pour installer une application il faut se rendre dans la rubrique `Applications`. Après un certain temps de chargement, vous verrez le logiciel vous lister l'ensemble des applications déjà présentes sur votre appareil.
Dans notre cas, nous voulons plutôt en installer une nouvelle. Pour cela, cliquez sur le bouton

`Importer des fichiers...` en haut à droite de la fenêtre, comme sur l'image suivante.

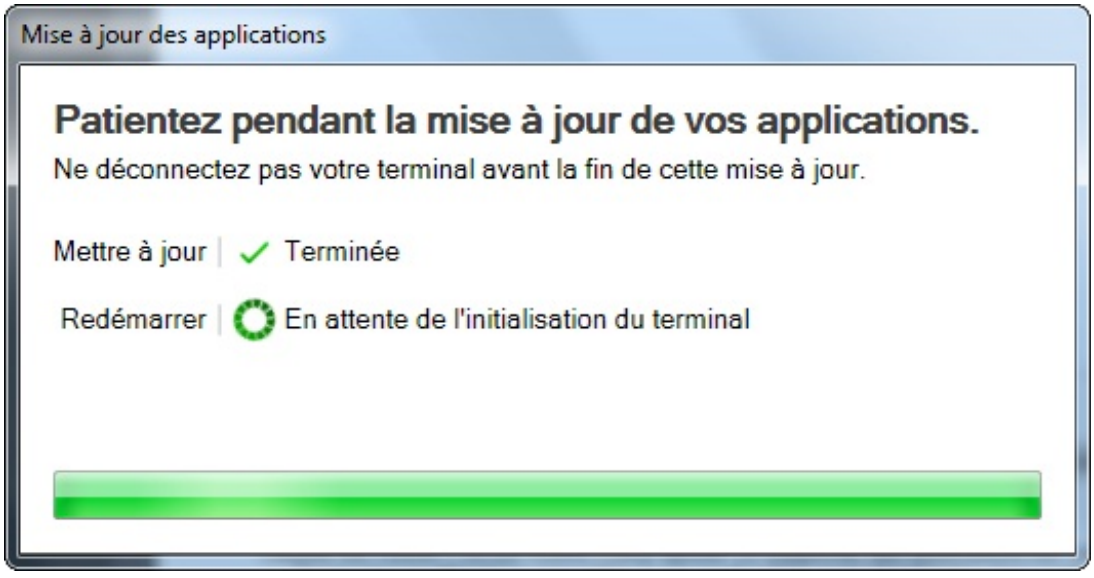


C'est maintenant que nous allons avoir besoin de ce fameux fichier `.alx` dont je vous avais parlé en début de chapitre. Vous allez donc vous rendre dans le dossier `..\deliverables\Standard\` de l'application que vous désirez installer. À l'intérieur, sélectionnez le fichier `.alx`.
À la suite de ceci, vous devriez normalement voir votre application apparaître parmi les autres, avec un statut « Installation... » :



Attention, à ce stade l'application n'est pas présente sur le smartphone. Le Desktop Manager n'applique les changements qu'après validation de ceux-ci.

Appuyez donc sur `Appliquer` pour valider l'ensemble des opérations. Attendez ensuite que la fenêtre suivante disparaisse.



Voilà, votre application est bien installée. Vous la trouverez comme d'habitude dans la section `Downloads`.

- Une application est distribuée et installée sous la forme d'un fichier `.alx`.
- Avant de publier une application, il est *important* de la tester sur une large gamme de simulateurs et terminaux physiques.
- Avant d'entamer cette phase de test, il serait judicieux de nettoyer son application.

- RIM met à disposition des simulateurs correspondant à l'ensemble des smartphones disponibles à la vente.
- Le **BlackBerry Desktop Manager** permet de se connecter à un appareil physique et d'y installer une application.

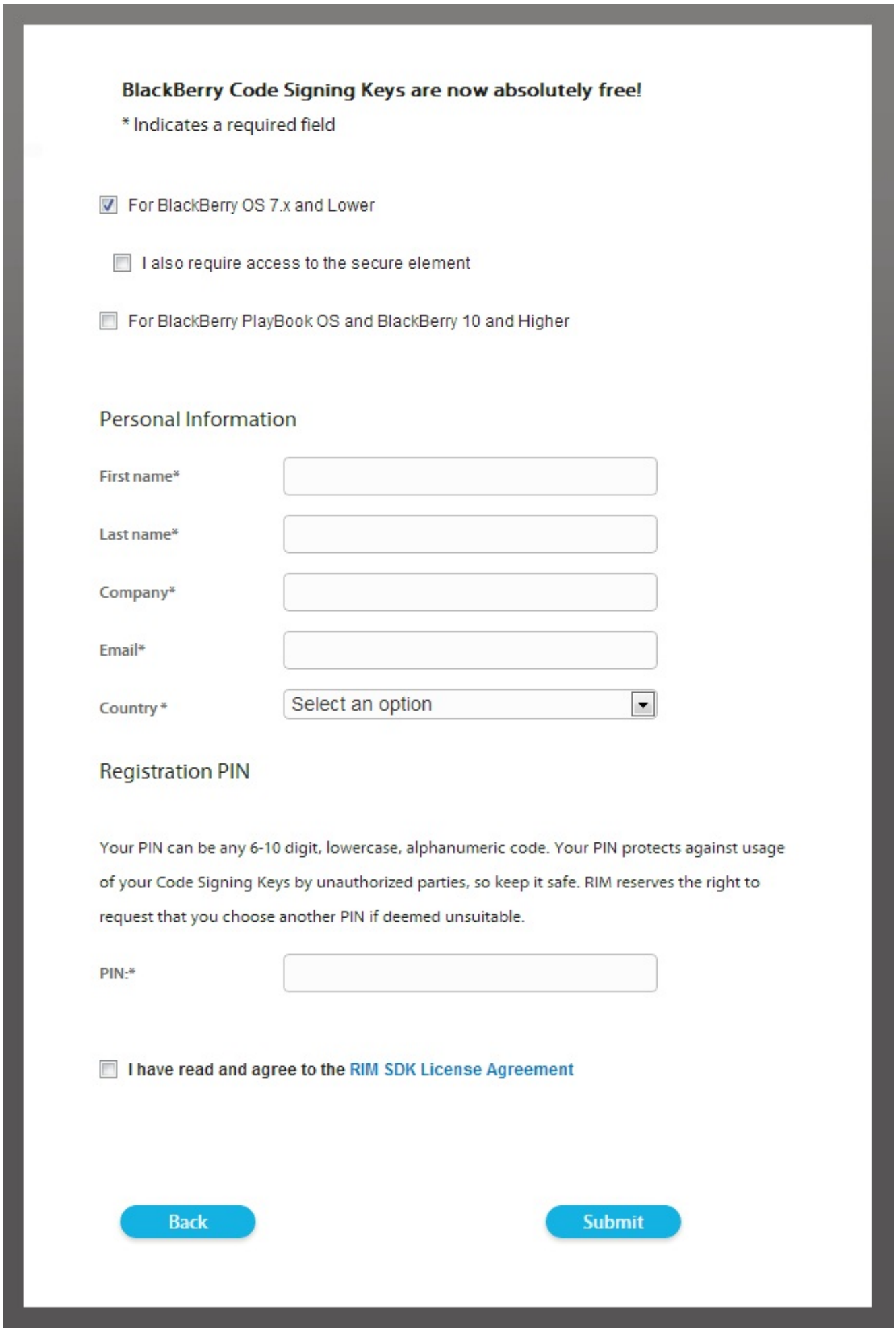
Déployer et distribuer

Pour ce dernier chapitre du cours, nous allons voir comment finaliser son application.
D'une part, nous verrons comment obtenir, installer et utiliser des clés permettant de signer une application. Ces signatures permettent d'identifier le développeur à l'origine d'une application, et sont indispensables dans certains cas.
Enfin nous terminerons le chapitre en décrivant les différents moyens de distribuer une application. Nous introduirons notamment les outils **BlackBerry App World** et **BlackBerry Enterprise Server**.

Installer des clés de signature

Faire une demande de clés

Avant de déployer et de distribuer son application, il est nécessaire de la *signer*. Pour cela, RIM a établi un système de **clés** individuelles associées à chaque développeur. Vous allez donc avoir besoin de ces clés, qui se présentent sous la forme de trois fichiers différents : `client-RBB-xxxxxxx.csi`, `client-RCR-xxxxxxx.csi` et `client-RRT-xxxxxxx.csi`.
Pour obtenir ces clés, il faut en faire la demande auprès de RIM, par le biais du formulaire suivant :



BlackBerry Code Signing Keys are now absolutely free!

* Indicates a required field

☑ For BlackBerry OS 7.x and Lower

☐ I also require access to the secure element

☐ For BlackBerry PlayBook OS and BlackBerry 10 and Higher

Personal Information

First name*

Last name*

Company*

Email*

Country * Select an option

Registration PIN

Your PIN can be any 6-10 digit, lowercase, alphanumeric code. Your PIN protects against usage of your Code Signing Keys by unauthorized parties, so keep it safe. RIM reserves the right to request that you choose another PIN if deemed unsuitable.

PIN:*

☐ I have read and agree to the RIM SDK License Agreement

Back Submit

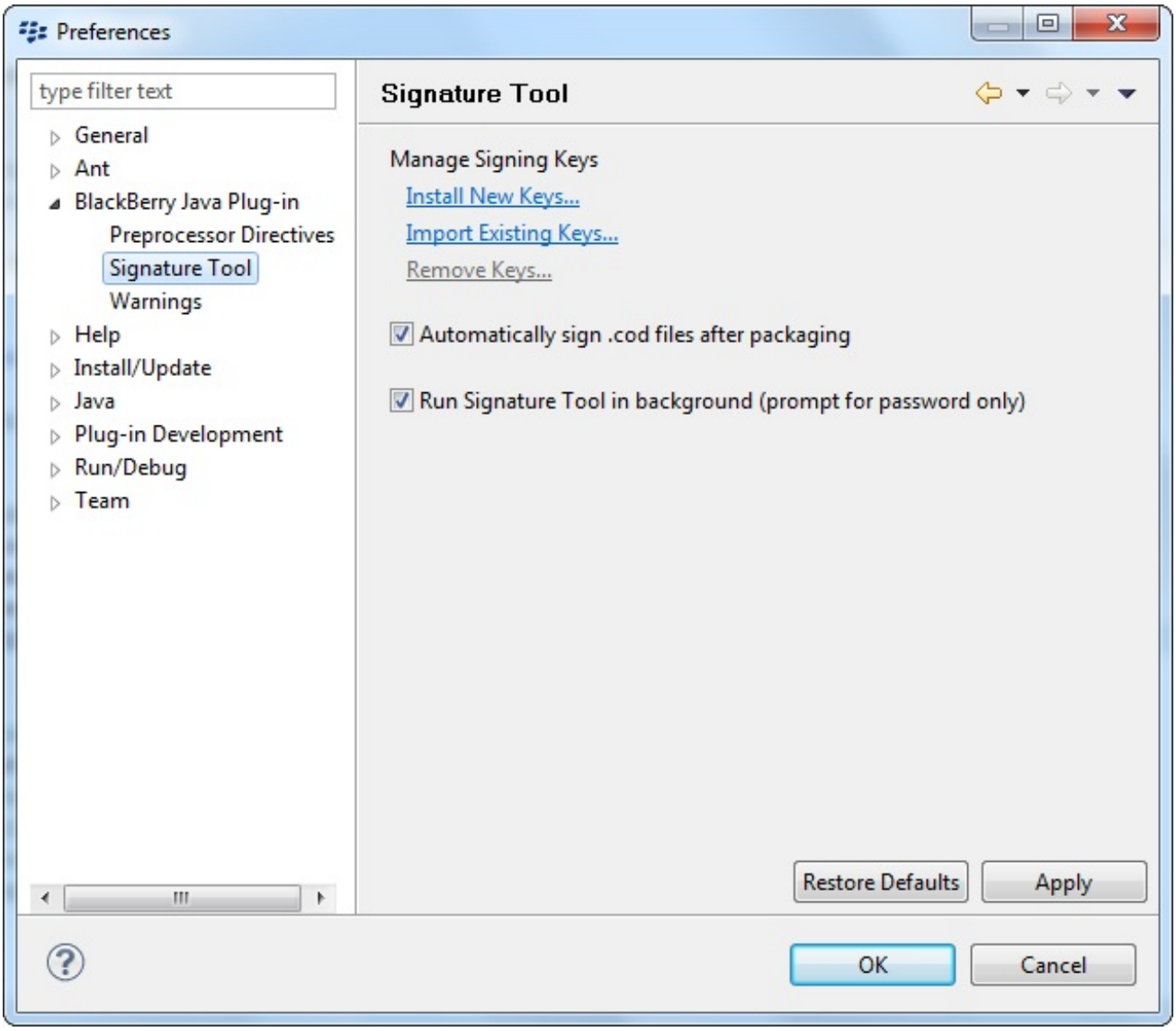
Vous remarquerez qu'à la fin du formulaire, un numéro PIN vous est demandé. Prenez soin de bien noter ce dernier, car il vous sera grandement utile lors de l'installation des clés.

Auparavant, cette demande de clés étant payante. Toutefois, elle est à présent totalement gratuite, alors profitez-en !

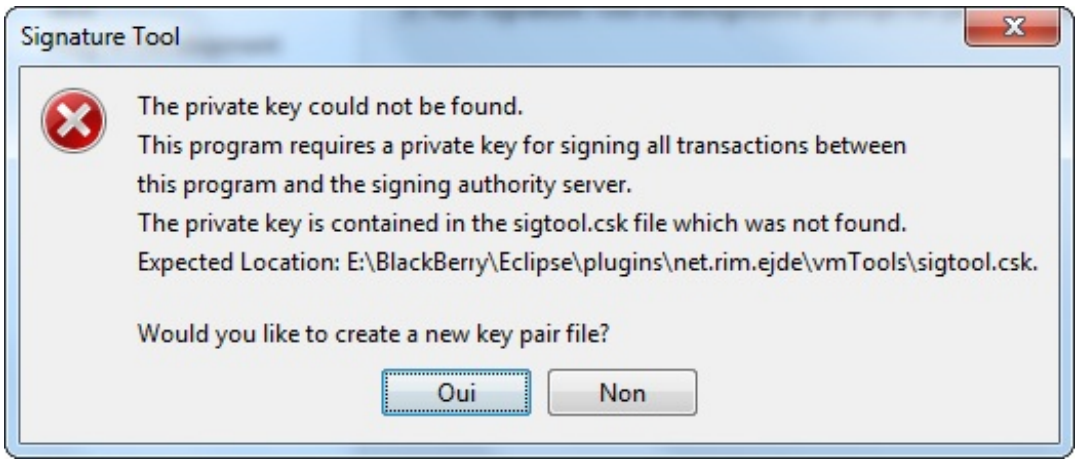
Une fois que vous aurez rempli le formulaire et que vous l'aurez validé, vous devrez attendre la confirmation de RIM, qui se fera par courriel. Pour être plus clair, lorsque votre demande aura été acceptée, vous recevrez une série de trois messages électroniques contenant chacun une clé sous la forme d'un fichier `.csi`.
Une fois que vous avez vos trois fichiers `.csi`, enregistrez-les sur votre disque dur, puis passez à l'installation de celles-ci.

Installer les clés

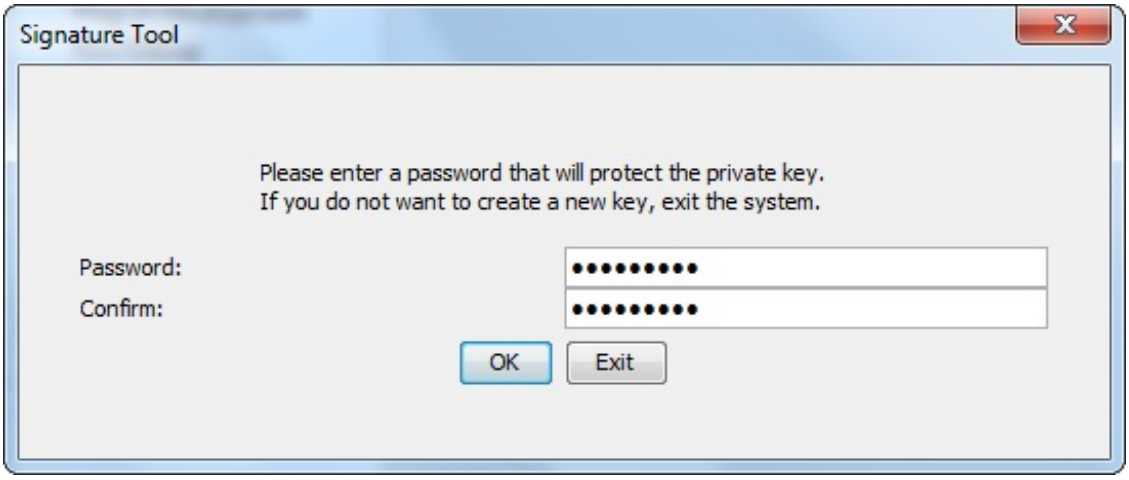
Pour installer les clés obtenues précédemment, commencez par ouvrir votre IDE : Eclipse !
Ensuite, ouvrez la fenêtre `Preferences` dans le menu `Window`. À l'intérieur de celle-ci, rendez-vous dans la section `BlackBerry Java Plug-in > Signature Tool`. À ce stade, vous obtenez la fenêtre suivante.



Cliquez alors sur `Install New Keys...`, et la fenêtre suivante apparaît.

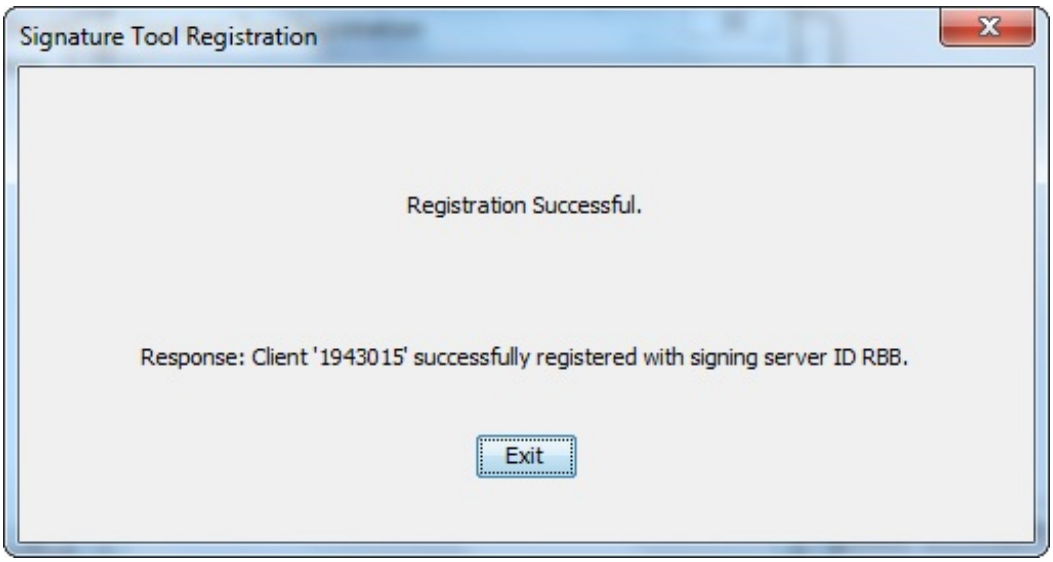


Cliquez sur le bouton `Oui`, puis dans la nouvelle fenêtre, allez chercher sur votre disque dur le premier fichier `.csi`. Une fois ce dernier sélectionné, il vous sera demandé de choisir un mot de passe, qu'il vous faudra bien entendu conserver pour la suite.



C'est maintenant que vous devez ressortir le numéro PIN renseigné précédemment, dans le formulaire de demande de clés. Saisissez-le ainsi que le mot de passe créé juste avant.

Si tout c'est correctement déroulé, une fenêtre de succès devrait s'afficher, comme à la figure suivante.



À ce stade, vous n'avez installé qu'une seule clé sur les trois. C'est pourquoi vous devez réitérer ces opérations pour les deux clés restantes, en cliquant à nouveau sur `Install New Keys...` !

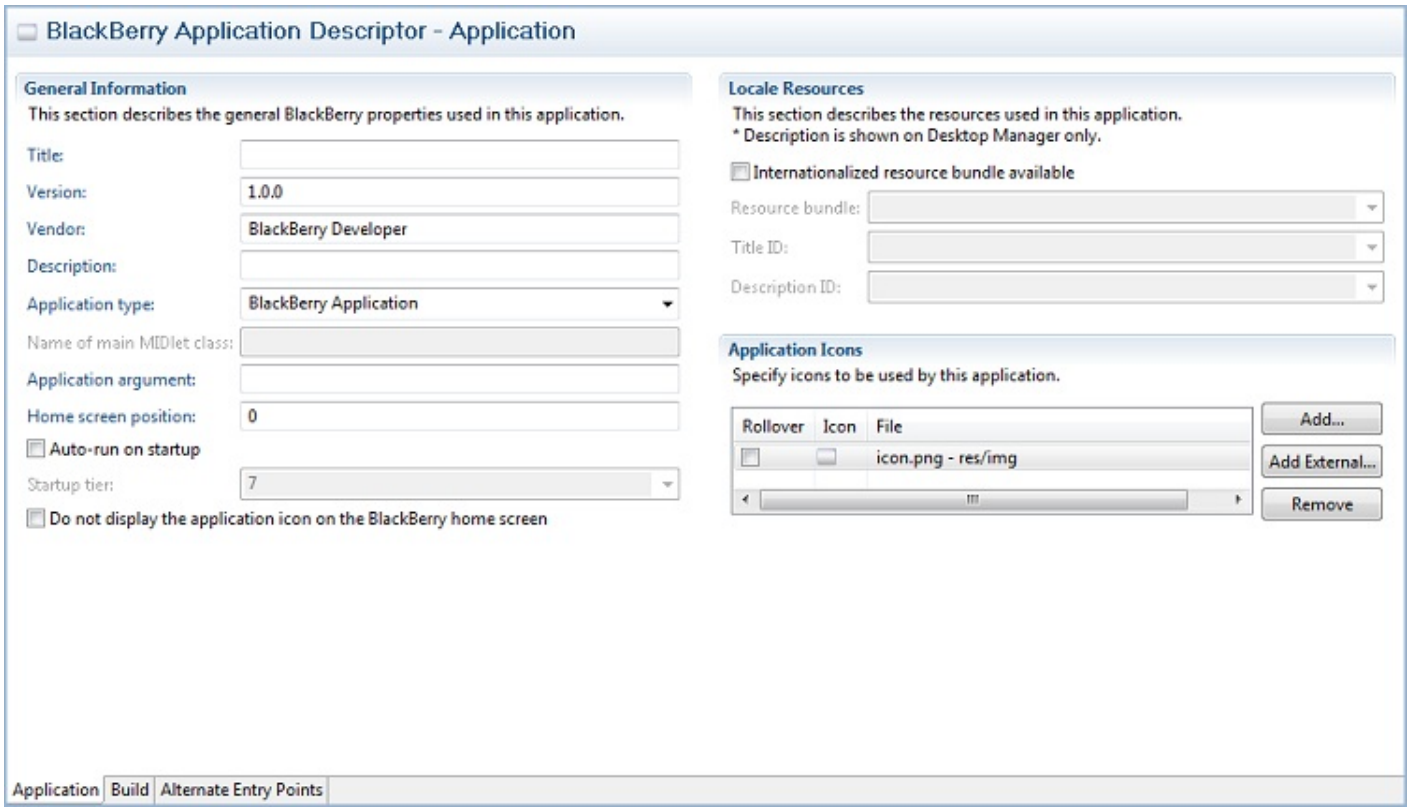
Finaliser son application

Nous allons à présent nous intéresser aux étapes de finalisation d'une application, à savoir la signature de celles-ci ainsi que le remplissage de la feuille de description de l'application.

Nous allons rapidement voir ici comment procéder pour ces étapes de finalisation d'une application. Toutefois, je vous conseille fortement d'aller faire un tour dans la documentation officielle si vous êtes sur le point de publier votre application. En effet, rien ne vaut les explications et renseignements fournis par RIM !

Remplir le BlackBerry Application Descriptor

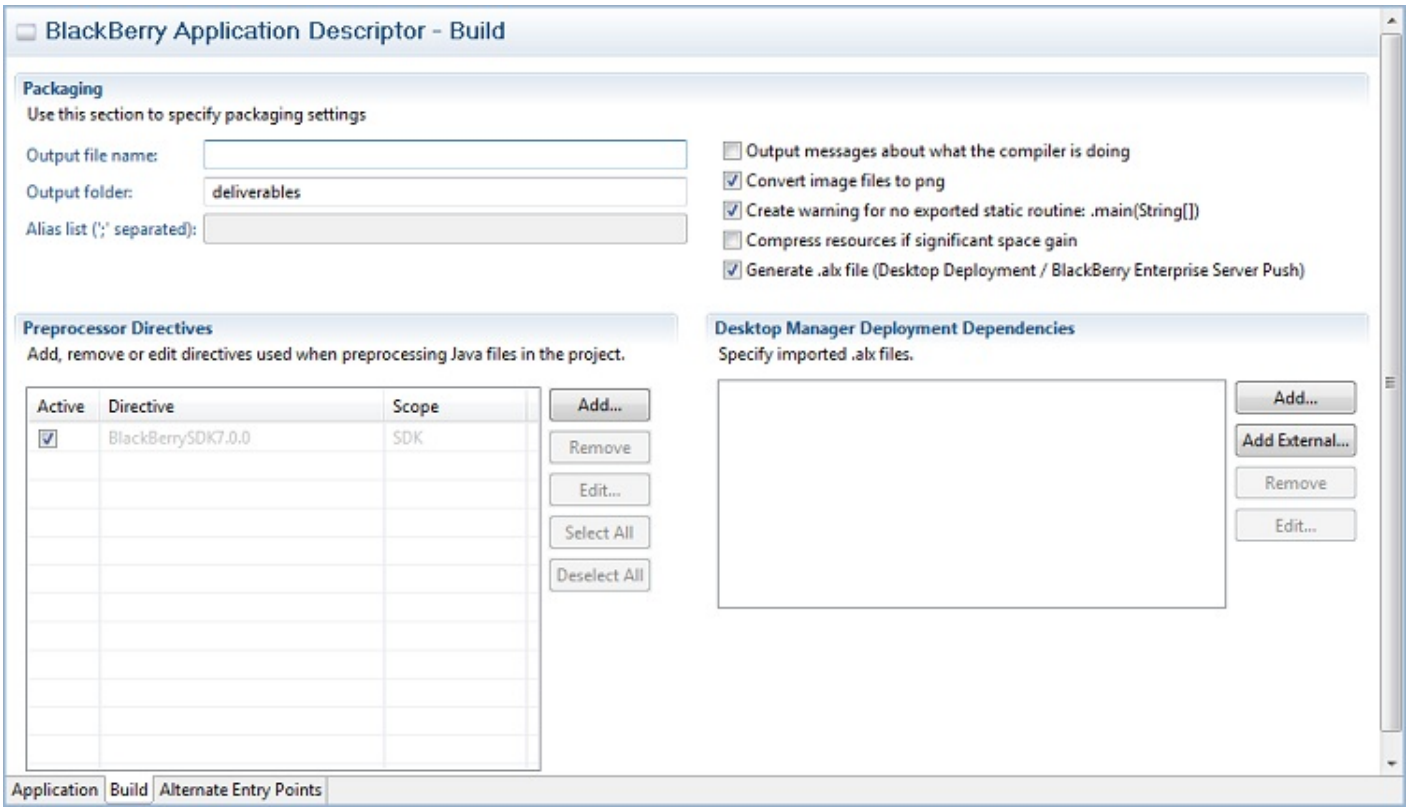
Pour commencer, revoyons ensemble ce fameux fichier de description de l'application nommé `BlackBerry_App_Descriptor.xml`.
Ouvrez-le, puis jetez un œil aux différents champs qui le composent. Vous verrez alors, comme son nom l'indique, qu'il s'agit des champs de description de l'application, tels que le nom, la version, le vendeur (vous), le type d'application, l'icône associée, etc. Voici donc à la figure suivante ce « pseudo-formulaire ».



Notez que pour spécifier le nom du vendeur, il est préférable de faire avant, une demande de création de compte vendeur sur l'**App World** auprès de RIM, mais nous y reviendrons plus tard.

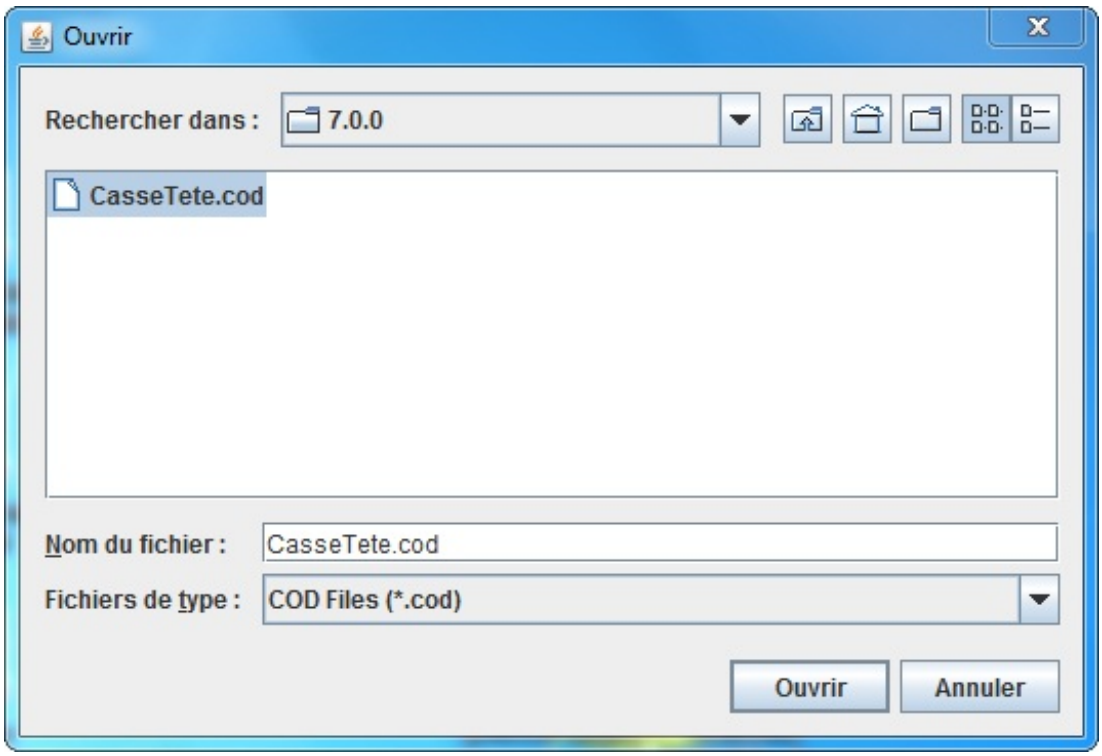
Un autre point sur lequel je voudrais également revenir est le numéro de version. En général, il se présente sous la forme `<majeur>.<mineur>.<micro>` ou `<majeur>.<mineur>.<révision>`, où chaque point sépare des évolutions dans l'application qui sont plus ou moins importantes. Ainsi, la version 2.0.3 représente la troisième révision de la première version mineure de la deuxième version majeure de votre application. En réalité, ce n'est très grave si vous ne suivez pas exactement ce schéma-là, mais tachez de ne pas faire passer un changement de police de caractères pour une évolution majeure !

Je vous laisserai donc le soin de remplir par vous-mêmes l'ensemble des champs présents dans ce fichier. Cependant avant de passer à la suite, je voudrais vous faire remarquer qu'il existe différents onglets. Pensez donc à faire un tour de l'ensemble du fichier avant de clôturer votre application.
Voici par exemple un aperçu du deuxième onglet, où nous retrouvons notamment une case à cocher permettant de générer un fichier `.alx`.

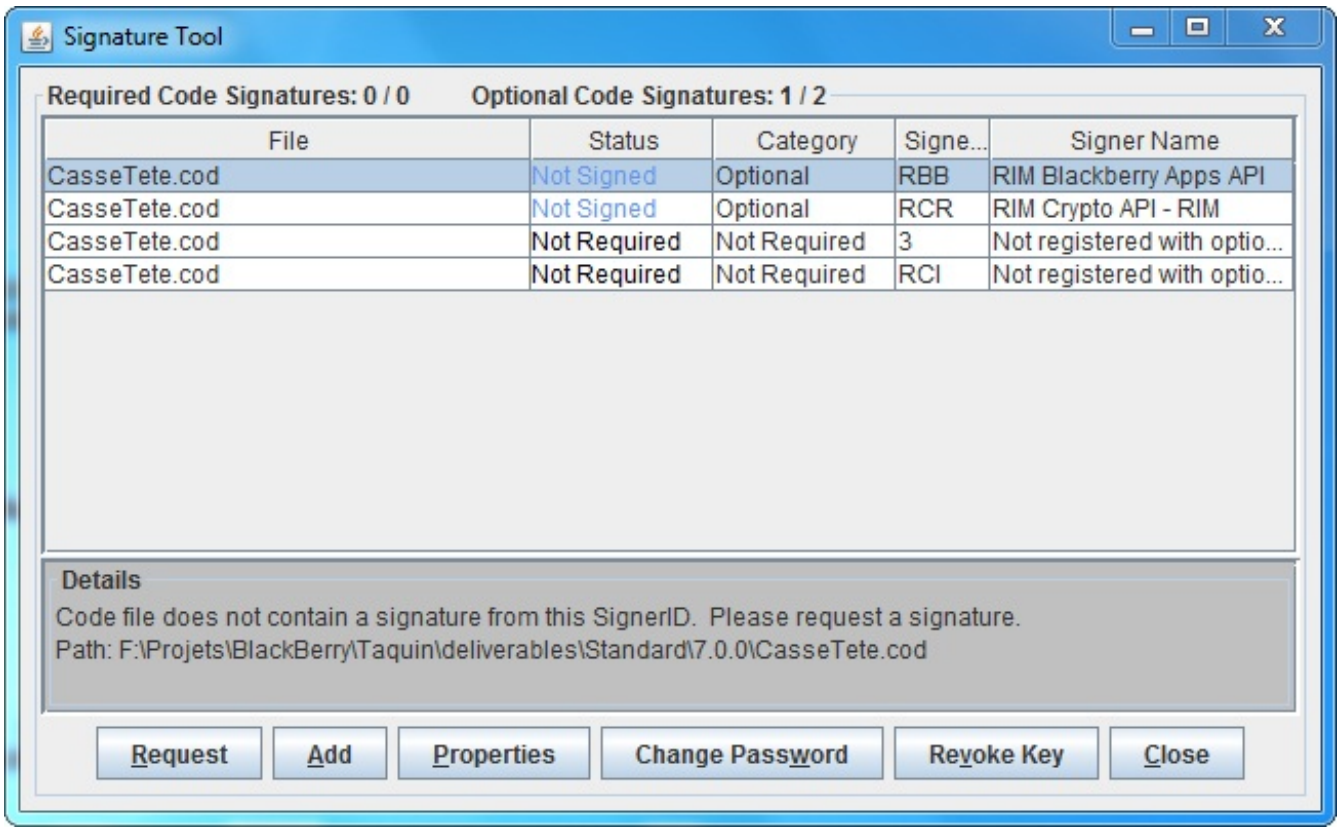


Signer son application

À l'origine pour signer son application, il suffisait de sélectionner son projet sous Eclipse, puis de faire un clic-droit et terminer en cliquant sur `BlackBerry > Sign with Signature Tool`. Néanmoins, il est peu probable que cette méthode soit encore disponible chez vous.
Pour pallier ce problème, il est toutefois possible d'ouvrir « manuellement » l'outil de signature en le cherchant dans le dossier d'installation du BlackBerry Java Plugin. Chez moi par exemple, j'ai trouvé cet outil qui se nomme « `SignatureTool.jar` » dans le sous-dossier `..\plugins\net.rim.ejde\vmTools\`.
Une fois que vous l'avez trouvé, lancez-le et vous obtiendrez la fenêtre suivante.

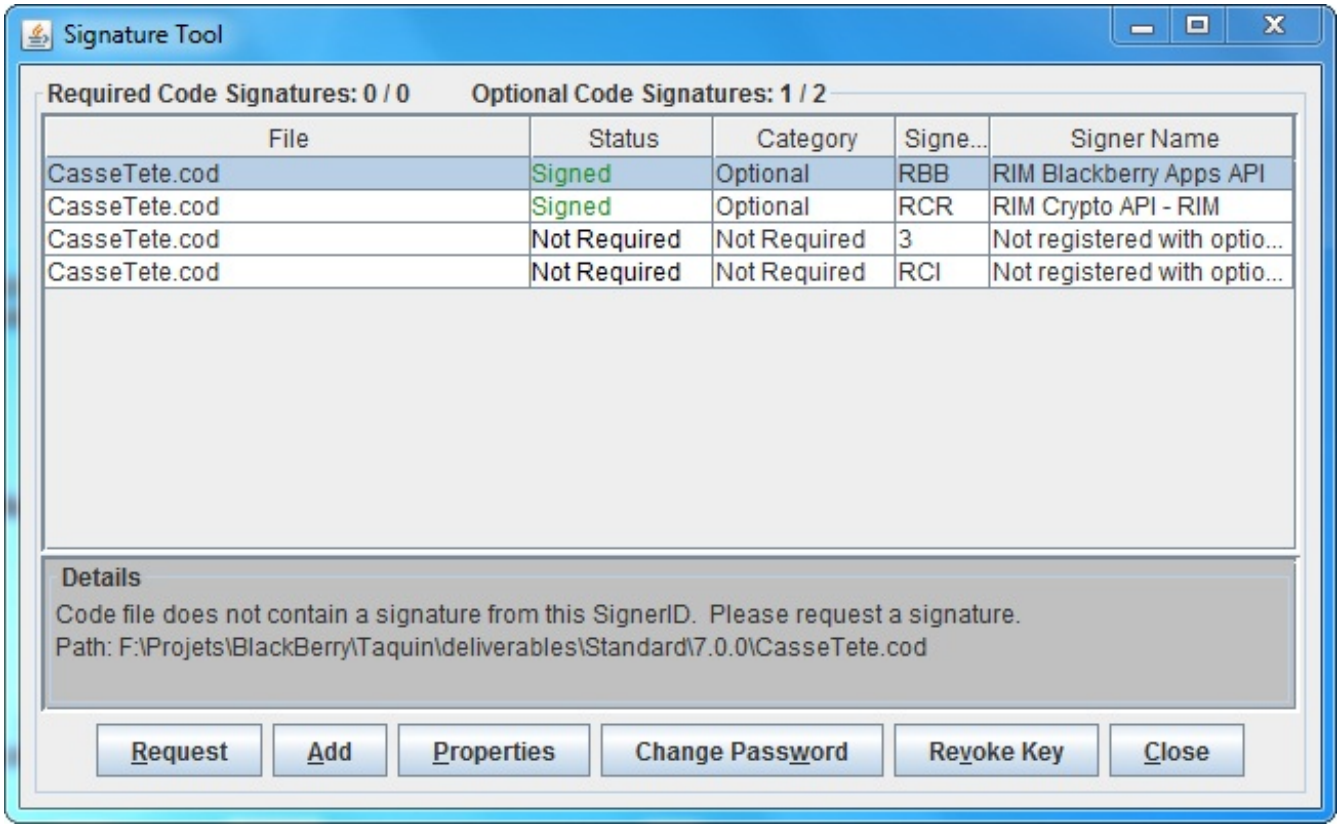


Comme vous pouvez le voir, l'outil permet d'ouvrir des fichiers de type `.cod`, qui sont générés après compilation de vos projets. Rendez-vous donc dans le dossier du projet concerné, afin d'y trouver ce fichier. Ensuite ouvrez-le, puis regardez les informations présentées par l'outil.



Comme vous le voyez, ce projet comporte des statuts « Non Signés ». Ainsi pour entamer la procédure de signature, cliquez sur le bouton `Request`, puis entrez le mot de passe créé pendant la phase d'installation des clés dans la nouvelle fenêtre.

Après un certain temps, vous pourrez voir apparaître les statuts « Signés », comme indiqué sur la figure suivante.



Ça y est, votre application est à présent signée et prête à être distribuée !

La distribution

Avant d'aller plus loin, vérifiez bien l'état de votre application. Pensez notamment à recompiler votre application et à vérifier à l'aide du `Signature Tool` que celle-ci est bien signée !

Proposer une application sur l'App World

BlackBerry App World est un service de distribution d'applications géré par RIM, qui permet de centraliser les applications disponibles pour les smartphones BlackBerry au sein d'un même endroit.
L'avantage de ce mode de distribution est donc bien entendu une grande visibilité de votre application auprès des utilisateurs BlackBerry. Cela peut donc être un bon moyen de rentabiliser votre application en la rendant payante. Pour proposer une application sur l'App World, vous devez cependant posséder un compte vendeur en en faisant la demande auprès de RIM. Pour cela, complétez donc la procédure d'enregistrement en suivant ce lien. Vous devrez alors remplir divers formulaires, puis confirmer la création de votre compte par l'envoi d'un justificatif tel qu'un scan de votre pièce d'identité si vous êtes un développeur indépendant.
Une fois enregistré auprès de RIM, vous pourrez alors déposer vos applications sur l'App World, et éventuellement les proposer suivant une certaine somme. Sachez que RIM fonctionne principalement avec les comptes Paypal pour gérer les transferts

d'argent.

Utiliser BlackBerry Enterprise Server

Pour les entreprises, il est possible d'utiliser **BlackBerry Enterprise Server**, qui est une application permettant de connecter des smartphones BlackBerry aux serveurs et applications courriels de l'entreprise. Les applications peuvent donc être installées sur des terminaux qui sont connectés à BlackBerry Enterprise Server.

Je n'entrerai pas dans les détails et j'invite les intéressés à se rendre directement sur le site de cet outil ; BlackBerry Enterprise Server.

D'autres moyens de distribution

En réalité, votre application est présentée sous la forme d'un fichier `.alx`, que vous pouvez transférer sur un terminal à l'aide du Desktop Manager. Ainsi, toute forme de distribution de ce fichier est bonne à prendre.
Par exemple, vous pouvez proposer votre application par courriel en joignant le fichier `.alx` à votre message. Néanmoins, je vous déconseille d'utiliser une application de source inconnue. N'utilisez les boîtes électroniques qu'avec des connaissances.
Ensuite par le bouche-à-oreille, votre application pourrait très bien trouver un certain succès !
Une autre manière de distribuer son application est d'utiliser le web. En effet, il n'est pas rare de se voir télécharger un logiciel directement depuis le site web du développeur ou de la compagnie à l'origine de celui-ci. Vous pouvez donc tout à fait proposer votre application au téléchargement depuis votre site web, ou même mieux, réaliser un mini-site pour votre application.

Utilisez donc tous les moyens à votre disposition pour élargir votre communauté d'utilisateurs !

- Pour signer ses applications, il est nécessaire de disposer de trois clés sous forme de fichiers `.csi`.
- **Signature Tool** est un outil permettant de vérifier et d'appliquer une signature aux applications.
- Le fichier `BlackBerry_App_Descriptor.xml` permet de décrire l'application et est principalement utile dans le cas d'une publication sur l'App World.
- **BlackBerry App World** est le meilleur moyen de distribuer une application pour un développeur indépendant.
- Les entreprises peuvent se servir de **BlackBerry Enterprise Server** pour diffuser des applications parmi leurs collaborateurs.

Ce tutoriel est à présent terminé !
Toutefois si vous le désirez, vous pouvez encore vous perfectionner en vous rendant par exemple sur le site officiel de développement BlackBerry en Java.

Remerciements

Par ailleurs, je tiens à remercier les personnes ci-dessous sans qui ce tutoriel n'aurait jamais vu le jour :

- l'ensemble des validateurs, et plus particulièrement **Guizmo2000**, qui contribuent énormément à la bonne rédaction des

tutoriels, et dont le travail n'est pas toujours mis en valeur,
- **AnnaStretter** qui a suivi et accompagné ce tutoriel de près et a ainsi permis d'améliorer la qualité de ce dernier,
- **M@teo21** et l'équipe de **Simple IT** qui font vivre le Site du Zéro et permettent à de nombreuses personnes de se former sur une grande variété de langages et de domaines,
- l'ensemble des lecteurs qui ont contribué à l'avancement de ce cours par leur intérêt pour celui-ci.