Java ME : une présentation

Jean-Marc Farinone

But de l’exposé

- Comprendre, définir, situer les termes :
  - Java ME, J2ME, CDC, CLDC, Configuration, Profiles, MIDP (1.0, 2.0), MIDlet, jad, etc.
  - Donner des références
  - Donner des exemples de programmes

Les concepts fondamentaux de Java ME

Java ME = ?

- Java ME = Java Micro Edition
- Site de référence = `http://java.sun.com/javame/index.jsp`
- S'appelait anciennement J2ME : terme encore très souvent employé.
- Famille de spécifications pour développer des logiciels pour des objets électroniques (device = périphérique) comme 
  - les téléphones portables,
  - les assistants personnels (PDA)

  - « téléphones intelligents » (smart phones)

Java ME = Configuration et Profiles

- Le monde des périphériques électroniques est vaste, divers et varié.
- => Pas de possibilités d'avoir un seul environnement uniforme pour tous (!= Java SE)
- => Architecture en couche :
  - Bibliothèques de base : les configurations
  - Les ajouts à ces bibliothèques : les profiles

Configuration

- = Spécifications pour un ensemble de périphériques ayant des caractéristiques similaires comme :
  - Le type et la quantité mémoire disponible
  - Le type de processeur (vitesse, etc.)
  - Le type de réseau disponible pour ce périphérique
- Configuration = plate-forme minimale pour cet ensemble. Pas d'extension ni de retrait possible
- => portabilité

Les deux configurations fondamentales

- CLDC (Connected Limited Device Configuration), CDC (Connected Device Configuration)
- CLDC ~ wireless Java.
  - Pour téléphone cellulaire, PDA ayant 192 Ko de mémoire minimum (CLDC 1.1) pour la JVM
  - Téléchargement de programmes Java
  - 2 versions 1.0 (JSR-30 Mai 2000), 1.1 (JSR-139 Mars 2003)

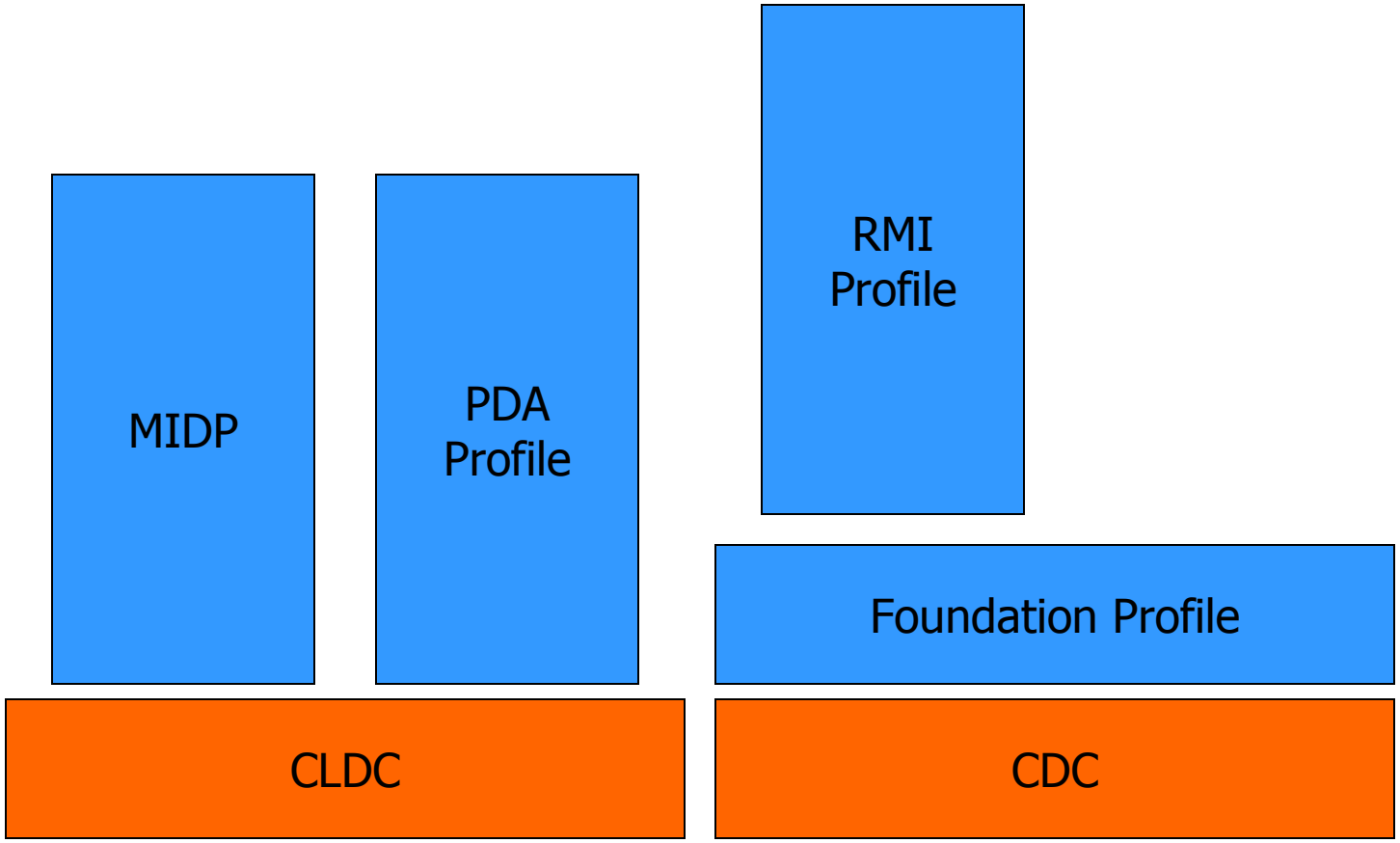
Les deux configurations fondamentales (suite)

- CDC = entre CLDC et Java SE
  - Périphériques ayant 2Mo ou plus de mémoire : smart phone, web téléphone, boitier TV (set-top boxes).

Profile

- = un complément à une configuration.
- Apporte des classes supplémentaires pour un domaine ou un marché spécifique
- Les profiles au dessus de CLDC :
  - MIDP (Mobile Information Device Profile)
  - PDA Profile
- Les profiles au dessus de CDC :
  - Foundation Profile
  - RMI Profile

Configuration et Profile : conclusion



MIDP =

- Mobile Information Device Profile
- Amène :
  - la partie réseau (+ HTTP 1.1)
  - des composants d'IHM
  - le stockage local
    à CLDC

Java ME : les restrictions % Java SE

- Règles fondamentales :
  - Une interface de programmation qui existe dans Java SE et qui est reprise dans Java ME se trouve dans le même paquetage, la même classe avec la même sémantique que dans Java SE
  - Il peut y avoir des champs et méthodes en moins dans une classe
- Les notions propres à Java ME en ajout % Java SE se trouvent dans des paquetages autres que ceux de Java SE.

CLDC : les restrictions % Java SE (suite)

- 3 paquetages repris (pas en totalité) :
  - `java.io`, `java.lang`, `java.util`
- Des paquetage additionnels, sous paquetages de `javax.microedition`

MIDP

Introduction

- Pas d'APIs d'interaction utilisateur, de stockage, de réseau, dans CLDC
- d'où MIDP
- applications MIDP = MIDlets
- réseau par HTTP 1.1 au moins (pas forcément TCP/IP)

IHM MIDP

- IHM sur un "petit" écran :
  - au moins 96 pixels de large sur 54 pixels de haut,
  - 2 couleurs,
  - rappel !! PDA = 160x160, 65536 couleurs
- "petit" clavier 

ou 

  au moins les chiffres de 0 à 9, les flêches, un bouton de sélection (ou équivalents).

`jar`, `jad` et cie

- Les MIDlets et leur ressources sont mises dans un `.jar`
- … qui peut être très gros
- Le contenu du `.jar` est décrit par son fichier `META-INF\MANIFEST.MF` (comme d'hab)
- Pour éviter d'avoir à charger tout le `.jar` pour avoir des infos sur l'archive (et éventuellement alors l'ignorer !!) une copie du manifeste est créée et peut être chargée : le `.jad`

Exemple de `jad`

- Rappel : le format d'un `jad` est celui d'un fichier manifeste.

`FPDemo.jad`

```
MIDlet-1: Calculator, calculator.png, calculator.CalculatorMIDlet
MIDlet-Description: Floating Point demonstration MIDlet
MIDlet-Jar-Size: 2451
MIDlet-Jar-URL: http://www.monSite.fr/FPDemo.jar
MIDlet-Name: FPDemo
MIDlet-Vendor: Sun Microsystems, Inc.
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.0
```

- Champs importants :
  - `MIDlet-Jar-Size: 2451`
  - `MIDlet-Jar-URL: http://www.monSite.fr/FPDemo.jar`

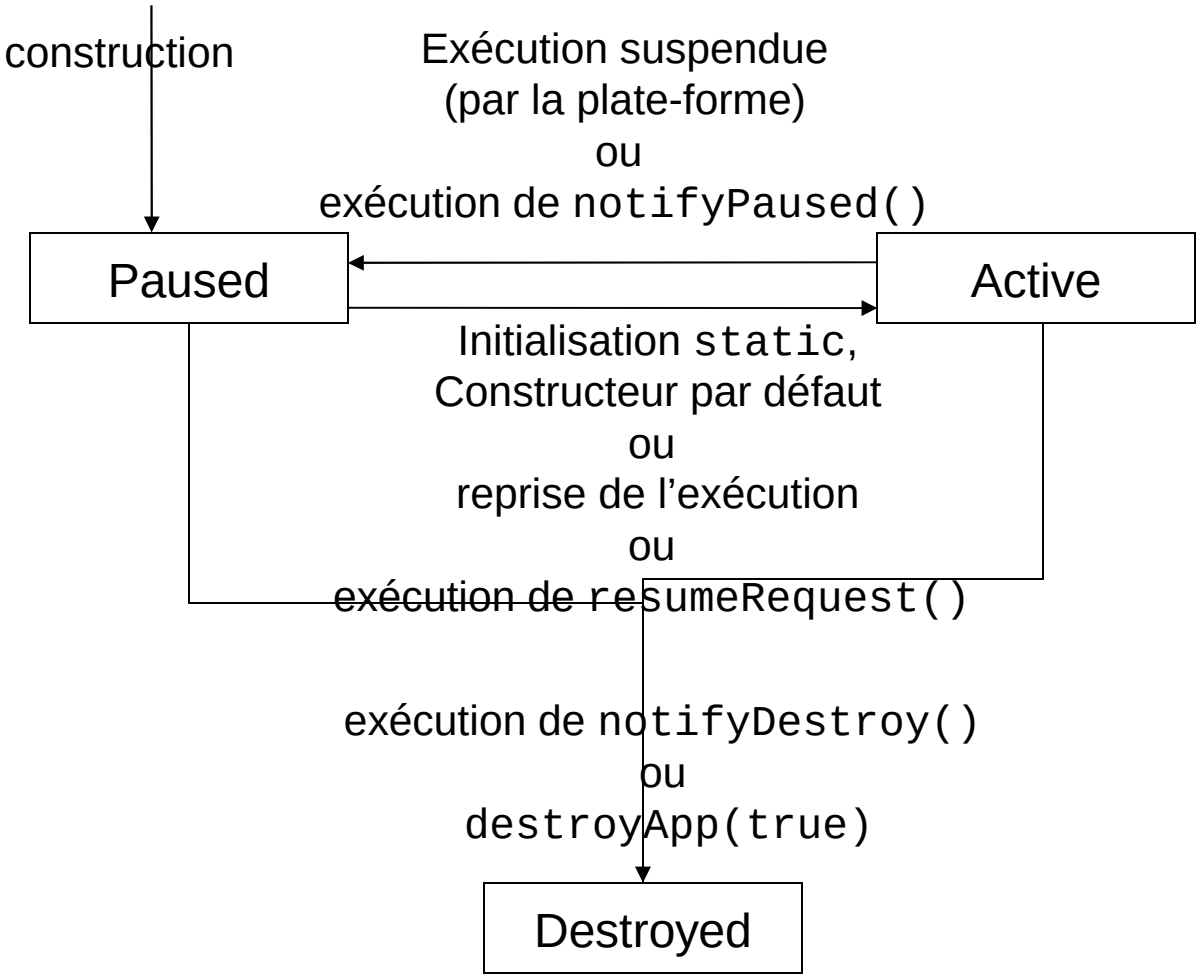
MIDlet

- Dérive de la classe abstraite `javax.microedition.midlet.MIDlet`
- Doit avoir un constructeur par défaut (éventuellement donné par le compilateur)
- La MIDlet minimale :

```
// pour la classe abstraite MIDlet
import javax.microedition.midlet.*;

public class TrameMIDletJMF extends MIDlet {
    // les 3 méthodes abstraites de MIDlet
    public void destroyApp(boolean condition)
         throws MIDletStateChangeException {}
    public void pauseApp() { }
    public void startApp() throws MIDletStateChangeException {}
    public TrameMIDletJMF(){ }
}
```

MIDlet : son cycle de vie



Développer une MIDlet

- Installer Java SE.
- Charger gratuitement l'environnement "Wireless toolkit" à partir de `http://java.sun.com/products/j2mewtoolkit/index.html`
- Eventuellement être inscrit au Download Center.

Développer une MIDlet (suite)

- Lancer la Ktoolbar (soit par windows soit par des commandes en ligne)
- Créer un projet (New Project). Donner un nom de projet, le nom de la classe MIDlet (ici `PremiereMIDletJMF`, cf. diapo suivante) Cliquer "Create Project". =>
  - 1°) Les infos du `.jad` sont affichées.
  - 2°) un répertoire du nom du projet a été créé sous l'environnement wireless toolkit.
- On placera sources, ressources, etc. dans ce répertoire.
- Début de la demo

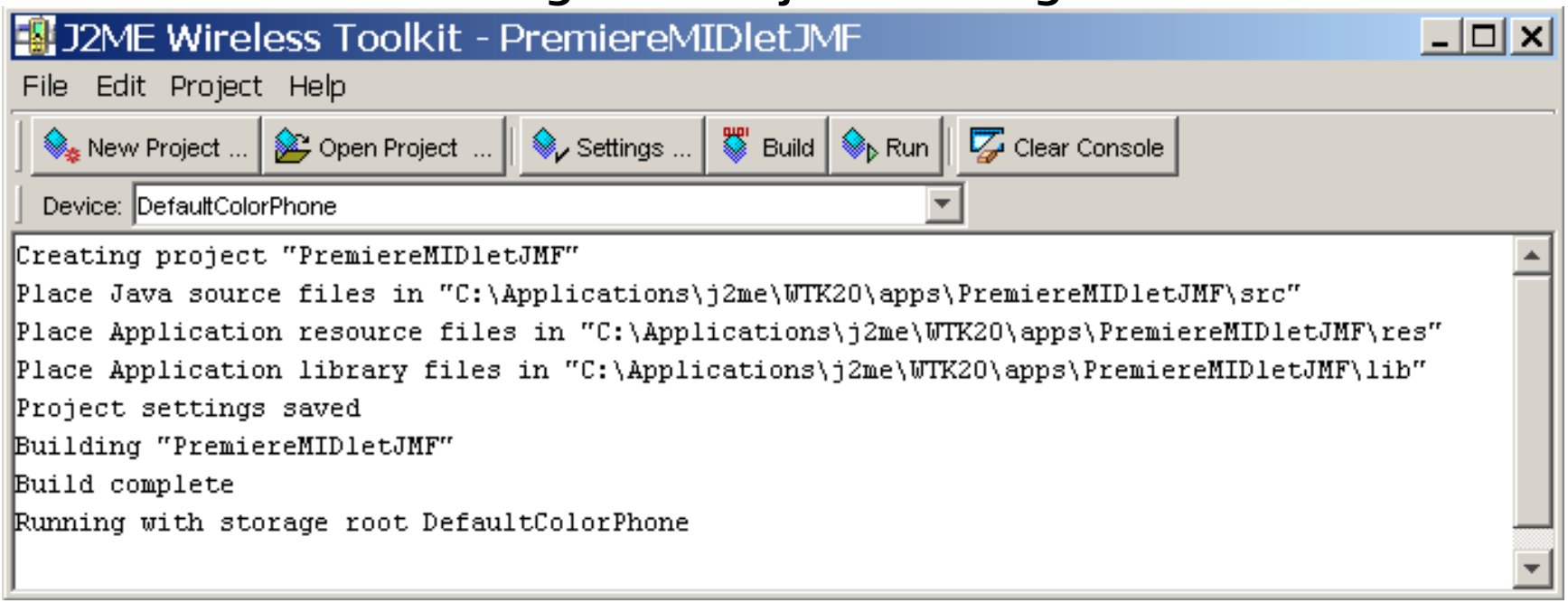
Développer une MIDlet (suite)

- Code de la MIDlet à sauvegarder dans `src`

```
import javax.microedition.midlet.*;
// pour CommandListener
import javax.microedition.lcdui.*;
public class PremiereMIDletJMF extends MIDlet implements CommandListener {
    // les 3 méthodes abstraites de MIDlet
    public void destroyApp(boolean condition){}
    public void pauseApp(){}
    public void startApp(){
         Display.getDisplay(this).setCurrent(mMainForm);
    }
    // La methode de l'interface CommandListener
    public void commandAction(Command c, Displayable d) {}
    public PremiereMIDletJMF() {
         mMainForm = new Form("Ma Premiere MIDlet JMF");
         mMainForm.append(new StringItem(null, "Bonjour à tous"));
         mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
         mMainForm.setCommandListener(this);
    }
    private Form mMainForm;
}
```

Développer une MIDlet (fin)

- Cliquez Build. L'environnement a :
  - Créer les répertoires `classes`, `tmpclasses`.
  - Compiler les sources Java, résultat dans `tmpclasses`
  - Prévérifier ces `.class` et mis dans `classes`
  - Construit les `.jar` et ajuste le `.jad`



Exécuter la MIDlet

- Cliquer "Run"
- Changer de périphérique par Device (QwertyDevice)
- Une demo : OK !
- Plus de code ...
- … au chapitre suivant (programmation réseau avec MIDP)

Construction d'IHM pour Java ME (CLDC, MIDP)

Jean-Marc Farinone

Présentation

- MIDP propose 2 bibliothèques pour faire des IHM
  - la bibliothèque bas niveau. Pour faire des dessins ~ classe `java.awt.Graphics` de Java SE
  - la bibliothèque haut niveau qui apporte des composants graphiques comme AWT, Swing, ... Evidemment cette bibliothèque est moins riche que AWT, Swing, ...
- Les classes pour ces bibliothèques sont dans le paquetage `javax.microedition.lcdui` et `javax.microedition.lcdui.game` pour les jeux

IHM en MIDP : le principe

- On ne présente pas des fenêtres à l'utilisateur
- On présente une seule fenêtre à chaque instant, qui occupe tout l'écran
- Donc gestionnaire en pile de cartes (cf. `CardLayout`)
- Bref on voit un écran à la fois et on passe d'un écran à l'autre.

Le principe : un peu de code (1/2)

- On récupère l'écran physique d'affichage par la méthode statique de la classe `Display` :

```
public static Display getDisplay(MIDlet m)
```

donc à lancer par

```
Display disp = Display.getDisplay(this);
```

sur la midlet.

- Un seul display pour une midlet (design pattern singleton).
- Le display = l'écran physique (multiple buffering)
- Puis on positionne un écran logique déjà créé dans un composant graphique (`form`) sur l'écran physique par :

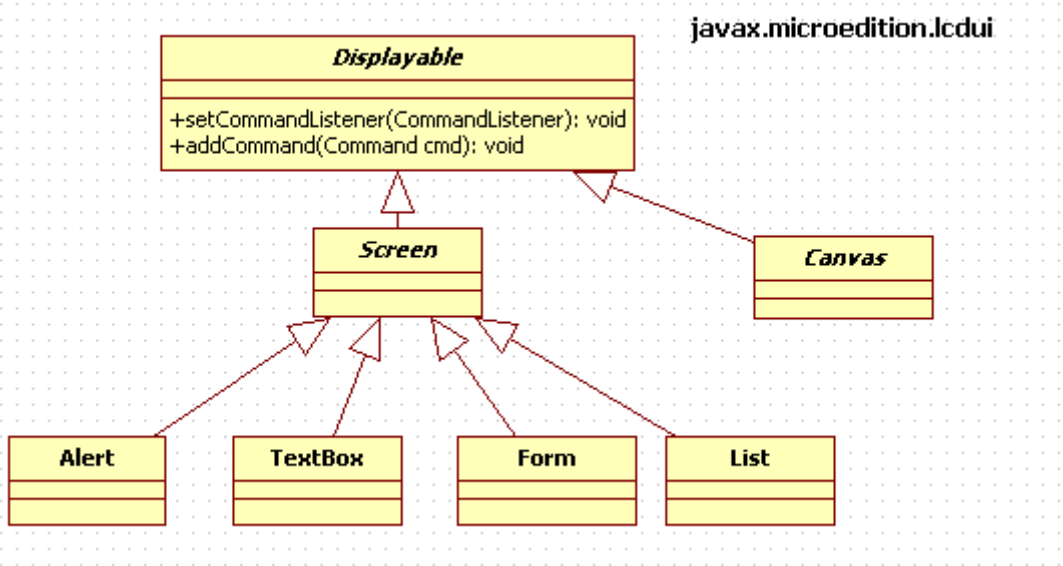
```
disp.setCurrent(form);
```

Le principe : un peu de code (2/2)

- On utilise donc la méthode

```
public void setCurrent(Displayable
  nextDisplayable)
```

- Et les composants graphiques héritent de `Displayable`
- On a d'ailleurs



Les composants graphiques

- Un `Canvas` est un composant sur lequel on dessine. Classe destinée à être dérivée. Utilisé pour l'API bas niveau
- Un `Form` est le composant standard de haut niveau pour construire un bel écran contenant des `Item` (= composant graphique avec un texte). Bref `Form` et `Item` de Java ME ~ conteneur et contrôles des IHM
- `List`, `Alert`, `TextBox` de Java ME ~ `Dialog` de AWT

IHM haut niveau en MIDP

Un premier exemple : `TextBox`

- Comme un `TextBox` est un `Screen`, il occupe tout l'écran.
- C'est une zone de texte multi-ligne.
- Ne possède qu'un seul constructeur :

```
public TextBox(String title,String text, int maxSize, int
   constraints)
```

- `title` = titre de cet écran
- `text` = le texte contenu dans la `TextBox`
- `maxSize` = le nombre maximal de caractères à afficher
- `constraints` = famille de `TextBox` (`NUMERIC`, `PHONENUMBER` = affichage comme un numéro de téléphone, etc.) Ce sont des constantes de la classe `TextField`.

Un code complet de TextBox

```
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
public class MaTextBoxMIDlet extends MIDlet {
  private static final int MAX_TEXT_SIZE = 64;
  protected TextBox textBox;
  protected Display display;
  protected boolean started;
  protected void startApp() {
    if (!started) {
      String str = null;
      str = "Second programme avec TextBox";
      textBox = new TextBox("TextBox Example", str, MAX_TEXT_SIZE, TextField.ANY);
      display = Display.getDisplay(this);
      display.setCurrent(textBox);
      started = true;
    }
  }
  protected void pauseApp() { }
  protected void destroyApp(boolean unconditional) {}
}
```

Résultat : TextBox

Le composant MIDP : `List`

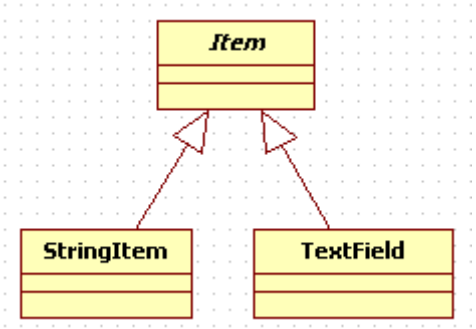
- Une `List` est un `Screen`, elle occupe tout l'écran.
- Elle possède 2 constructeurs dont le plus simple est

```
public List(String title, int listType)
```

- Elle présente des `String` qui peuvent être choisies en mode :
  - `Choice.EXCLUSIVE` : comme des boutons radios
  - `Choice.MULTIPLE` : comme des cases à cocher
  - `Choice.IMPLICIT` : comme des boutons radios
- En mode `MULTIPLE` et `EXCLUSIVE` , un changement de sélection n'est pas notifié : il faut ajouter une `Command`.
- En mode `IMPLICIT`, le changement de sélection avertit le `CommandListener` associé à cette `List`, qui lance alors sa méthode `commandAction()`.

Le conteneur MIDP : `Form`

- Une `Form` est un `Screen`, elle occupe tout l'écran.
- On met des `Item` dans une `Form`
- Champs de texte (`TextField`) et label (`StringItem`) sont des `Item`



- D'autres `Item` sont possibles : `DateField` (`TextField` pour saisir des dates), `Gauge`, `ChoiceGroup`, `ImageItem`.

Utiliser le conteneur MIDP : `Form`

- `Form` a 2 constructeurs : `Form(String title)` et

`Form(String title, Item[] items)`

- On ajoute des `Item` dans une `Form` grâce à la méthode :

```
public void append(Item unItem)
```

- Pour les `ImageItem` et les `StringItem`, on peut utiliser les méthodes

```
public void append(Image uneImage)
public void append(String uneChaine)
```

- Une `Form` a au moins la taille de l'écran. Elle peut avoir plus et dans ce cas un mécanisme de scrolling est construit automatiquement

Placement dans une `Form`

- Il n'y a pas de `LayoutManager` en MIDP
- Les composants permettant des saisies (`TextField`, `DateField`, `Gauge`, `ChoiceGroup`) sont placés les uns à la suite des autres dans l'ordre de l'exécution du programme.
- Les `StringItem` sans label sont placés horizontalement les uns à la suite des autres
- Les `StringItem` avec label sont placés les uns en dessous des autres
- Au fait `StringItem` a pour constructeur

`public StringItem(String label, String text)`,

un `label` étant un texte plus en évidence que `text`.

- On a des notions similaires pour les `ImageItem`

Le code partiel de `Form` (O'Reilly)

```
protected void startApp() {
    if (!started) {
        display = Display.getDisplay(this);
        Form form = new Form("Item Layout");
       form.append("Hello");
        form.append("World");
         // ajout JMF
         //form.append(new StringItem("leLabel", "leTexte"));

        form.append("\nLet's start\na new line\n");
        form.append("This is quite a long string that may not fit on one
line");

        form.append(new TextField("Name", "J. Doe", 32, TextField.ANY));
        form.append("Address");
        form.append(new TextField(null, null, 32, TextField.ANY));
         display.setCurrent(form);

        started = true;
    }
}
```

Démonstration de `Form` (O'Reilly)

- Dans projet exemplesIHM, MIDlet MesStringItemMIDlet

Interaction : traitements des événements

- Une façon de traiter les événements est d'utiliser des `Command`
- Un objet de la classe `Command` est un "bouton MIDP" que l'utilisateur va pouvoir actionner à l'aide des touches clavier.
- Les `Displayable` (et donc les `Screen` et donc les `TextBox`, etc.) possède une méthode

```
public void addCommand(Command);
```

- Ce bouton va être ajouté dans l'interface graphique du `Displayable`.
- L'endroit où ce bouton est ajouté dépend ... de beaucoup de choses (nombre de `Command` déjà mis, type d'écran, etc.)

La classe Command

- Elle possède un seul constructeur

```
public Command(String label, int type, int priority);
```

- `label` = le texte de la `Command`
- `type` = est une constante de la classe `Command`.
  - `OK` : suggère le lancement d'un traitement
  - `BACK` : doit ramener à l'écran précédent
  - `CANCEL` : suggère de ne pas lancer un traitement
  - `STOP` : suggère d'arréter un traitement
  - `EXIT` : doit arréter la MIDlet
  - `HELP` : doit afficher une aide
- Il faut évidemment écrire le code suggéré par la `Command` : son type ne suffit pas.
- `priority` = les valeurs les plus petites amènent une `Command` mieux placée dans l'interface

Les interactions (1/2)

- La programmation est similaire à Java SE
- On associe un (seul) listener au composant (!= Java SE)
- Le listener lance une méthode convenue lorsque la `Command` associé au `Displayable` a été manipulée par l'utilisateur (= Java SE)
- L'association est faite par

```
public void setCommandListener(CommandListener l)
```

- La méthode lancée par le listener est

```
public void commandAction(Command c, Displayable d)
```

- Le premier argument indique la `Command` de l'interface graphique qui a été utilisée
- Pour être un auditeur de `Command` il faut être un objet d'une classe qui implémente `CommandListener` (= Java SE)

Les interactions (2/2) : remarques

- Comme une même `Command` peut être mis dans plusieurs `Displayable`, le second argument est nécessaire. Il indique le `Displayable` qui contient la `Command` actionnée par l'utilisateur
- Il n'y a pas d'événement créé (!= Java SE) : il faut avoir tout le contexte au moment de traiter l'interaction.
- Remarque (très) importante
  - Contrairement à Java SE, la méthode `setCommandListener()` est lancée sur le `Displayable` contenant la `Command`, pas sur la `Command`

Un code complet d'interaction (1/2)

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
public class TextBoxInteractionMIDlet extends MIDlet implements CommandListener {
    private static final int MAX_TEXT_SIZE = 64;
    protected TextBox textBox;
    protected Display display;
    protected boolean started;
    Command exitCommand, reverseCommand;
    protected void startApp() {
        if (!started) {
            String str = null;
             str = "Second programme avec TextBox";
             textBox = new TextBox("TextBox Example", str, MAX_TEXT_SIZE, TextField.ANY);
             exitCommand = new Command("Exit", Command.EXIT, 0);
             reverseCommand = new Command("Reverse", Command.OK, 0);
             textBox.addCommand(exitCommand);
             textBox.addCommand(reverseCommand);
             textBox.setCommandListener(this);
            display = Display.getDisplay(this);
            display.setCurrent(textBox);
            started = true;
        }
    }
```

Un code complet d'interaction (2/2)

```
    protected void pauseApp() {}

     protected void destroyApp(boolean unconditional) { }

    public void commandAction(Command cmd, Displayable d) {
         if (cmd == exitCommand) {
              destroyApp(true);
              notifyDestroyed();
         } else if (cmd == reverseCommand) {
              String text = textBox.getString();
              if (text != null) {
                   StringBuffer str = new StringBuffer(text);
                   textBox.setString(str.reverse().toString());
              }
         }
    }
}
```

Démonstration de `Command` (O'Reilly)

- Dans projet exemplesIHM, MIDlet TextBoxInteractionMIDlet

IHM et architecture d'une MIDlet

- En général, on prépare tout dans la MIDlet :
  - les divers écrans qui risquent d'apparaître
  - les divers `Command` utiles à ces écrans
- Puis on fait en sorte que la MIDlet soit auditeur de ces `Command`
- Ainsi lorsque l'utilisateur appuie sur une `Command`, la main est repassée à la MIDlet qui redirige vers le prochain écran.

Interaction sur des `Item`

- Certains `Item` peuvent notifier leur changement immédiatement : on est pas obligé d'associer une `Command` à leur conteneur
- C'est le cas des `TextField`.
- Si une `Form` contient un `TextField` et que cette `Form` possède un `ItemStateListener`, ce listener sera notifié lors de changement de contenu par l'utilisateur
- On récupère alors la chaîne du `TextField` par

```
public String getString()
```

lancé sur le `TextField`

Interaction sur des `TextField`

- On a un code comme :

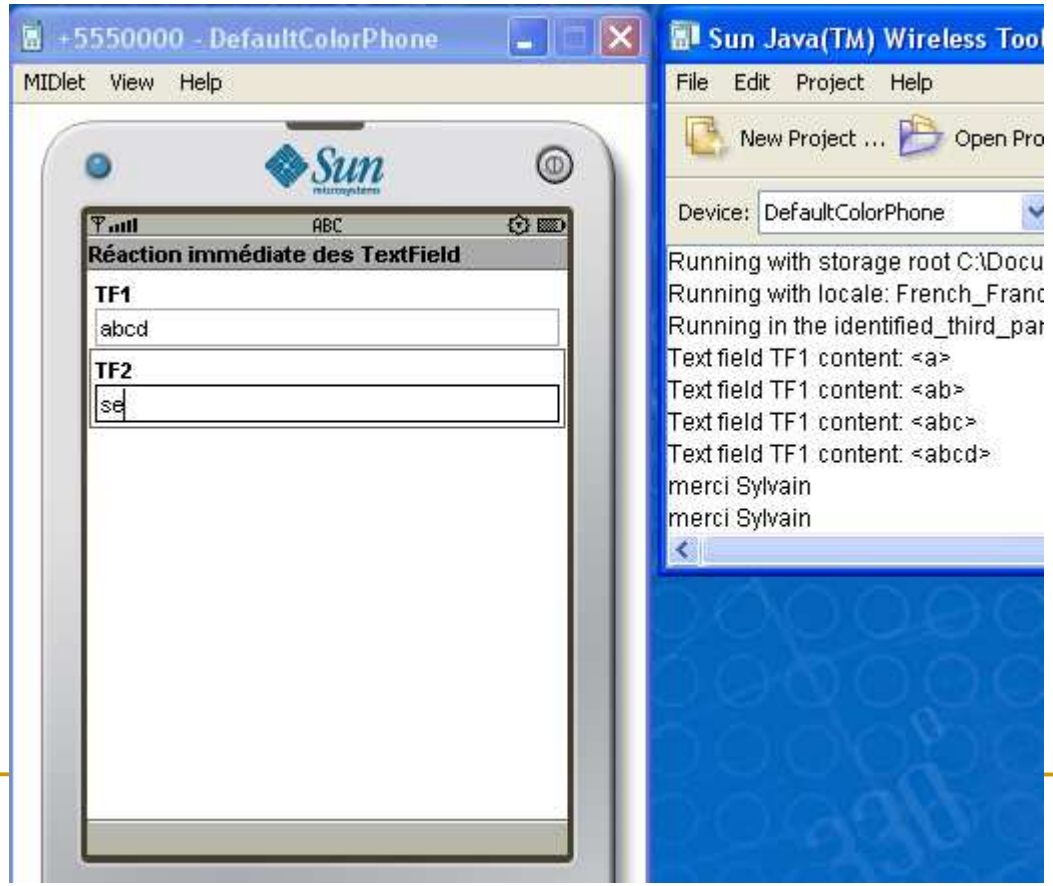
```
public class ItemMIDlet extends MIDlet implements CommandListener,
    ItemStateListener {

    Form laForme;

         creeIHM() {
              ...
              TextField tf = ...
              ...
              laForme.setItemStateListener(this);
         }
 public void itemStateChanged(Item item) {
         if (item instanceof TextField) {
             System.out.println("Text field content: <" +
                                ((TextField)item).getString() + ">");
    }
    //
}
```

Démonstration d'interaction dans des `TextField` (O'Reilly)

- Dans projet exemplesIHM, MIDlet TextBoxInteractionDirectMIDlet



Les Alertes

- Elles servent à avertir l'utilisateur souvent pour un problème
- Ce sont des `Screen`
- Les alertes peuvent être modales (qui restent à l'écran) ou non-modales
- Il est conseillé de construire des alertes modales car dans ce cas, l'environnement ajoute une `Command` Done qui permet d'enlever l'alerte
- Les alertes non modales restent une durée fixée par l'utilisateur ou par défaut (durée qui peut être mal adaptée)
- Après création on précise le caractère modale ou non modale par la méthode `setTimeout(int duree)` ou `duree` est en millisecondes. Pour une fenêtre modale (celles conseillées) on met la valeur `Alert.FOREVER`

Construction d'alertes

- Elles sont construites par :

```
public Alert(String title)
```

ou

```
public Alert(String title, String alertText,Image
  alertImage, AlertType alertType)
```

`title` est le titre de l'alerte, `alertText` est le texte à afficher, `alertImage` est l'image à afficher, `alertType` est le type d'alerte

- `alertType` peut valoir des valeurs constantes de la classe `AlertType` qui peuvent être `ALARM`, `ERROR`, `CONFIRMATION`, `WARNING`, `INFO`. Ces différentes valeurs donnent un degré d'erreur et diffèrent par exemple sur le type de sons (plus ou moins agressif) émis à l'affichage de l'alerte.

IHM bas niveau en MIDP

Le `Canvas`

- L'API MIDP de bas niveau permet de dessiner
- On dessine dans un `Canvas`. C'est un `Displayable` (donc on peut lui associer des `Command`)
- Un `Canvas` n'a pas de titre, il ne peut pas contenir de composants graphiques
- `Canvas` est une classe abstraite.
  - on doit construire son aire de dessin en dérivant de cette classe `Canvas`
  - il faut redéfinir la méthode `protected void paint(Graphics g)`
  - on peut gérer les entrées clavier (et il faut écrire le code correspondant)

`paint()` et `Graphics` de `Canvas`

- Les principes sont les mêmes qu'en Java SE
- `paint()` est appelé
  - quand le `Canvas` devient visible (par `display.setCurrent(...)`)
  - quand tout ou une partie du `Canvas` réapparait suite à un masquage par une `Alert` ou un menu
  - suite à l'exécution de `repaint()`
- L'argument `Graphics` de `paint()` permet de dessiner. Bref la classe `Graphics` donne des méthodes pour dessiner des droites, des portions d'arc, des contours de rectangles ou des surfaces, du texte, afficher des images, etc. (~ Java SE)

Texte dans un Canvas

- On peut écrire (dessiner) du texte dans un Canvas avec les méthodes de `Graphics` suivantes :
  - `public void drawChar(char character, int x, int y,int anchor)`
  - `public void drawChars(char[] data, int offset, int length, int x, int y, int anchor)`
  - `public void drawString(String str, int x, int y,int anchor)`
  - `public void drawSubstring(String str,int offset,int len,int x, int y, int anchor)`
- Elles sont plus riches que Java SE car elle possède l'argument `anchor`. Cet argument indique le point caractéristique du rectangle englobant la chaîne à écrire. Ainsi :

```
g.drawString("Coucou", canvas.getWidth(), 0, Graphics.TOP |
   Graphics.RIGHT);
```

permet de justifier en haut à droite

```
g.drawString("Coucou", canvas.getWidth()/2, 0, Graphics.TOP |
   Graphics.HCENTER);
```

permet de centrer en haut.

Police pour le texte

- Une police a 3 caractéristiques :
  - son aspect (Face)
  - son style
  - sa taille
- L'aspect est l'apparence globale des caractères. MIDP propose 3 polices :`Font.FACE_MONOSPACE`, `Font.FACE_PROPORTIONAL`, `Font.FACE_SYSTEM`,
- Les styles possibles sont `Font.STYLE_PLAIN`, `Font.STYLE_BOLD`, `Font.STYLE_ITALIC`, `Font.STYLE_UNDERLINE` et on peut les combiner
- Il existe 3 tailles : `Font.SIZE_SMALL`, `Font.SIZE_MEDIUM`, `Font.SIZE_LARGE`,
- On récupére une police par la méthode statique de la classe `Font :public static Font getFont(int face, int style, int size)`
- On peut changer la couleur, etc. Cf. demo O'Reilly chapitre 5 GraphicsMIDlet | Text

Les images : en créer

- On utilise des méthodes statiques de la classe `Image` pour récupérer ou créer des images
- On récupère des images (qui ne pourront pas être modifiées) par :

`public static Image createImage(String name) throws IOException` et `public static Image createImage(byte[] imageData,int imageOffset,int imageLength)`. Le premier appel est utilisé si l'image est dans le jar de la MIDlet. La seconde méthode est utilisée si l'image est passée par connexion réseau.

- Les images sont supportées si elles sont en format png.
- On peut créer une image éditable par `public static Image createImage(int width,int height)`

et dessiner alors dedans comme dans un `Canvas` (double buffering)

Les images : les afficher

- On utilise la méthode `public void drawImage(Image img, int x, int y, int anchor)` de la classe `Graphics`
- `x, y, anchor` ont le même sens que pour du texte

Gestion des événements dans un `Canvas`

- Un `Canvas` peut réagir au manipulation clavier et système de pointage (s'il y en a un) de l'utilisateur
- Pour le clavier, il suffit de de rédéfinir les méthodes
  - `protected void keyPressed(int keyCode)`
  - `protected void keyReleased(int keyCode)`
  - `protected void keyRepeated(int keyCode)`
- Il n'y a pas de listener à ajouter
- Pour utiliser `keyRepeated()`, il est bon de savoir si la plate-forme implémente l'événement "touche clavier enfoncée longtemps" en le vérifiant par

```
public boolean hasRepeatEvents()
```

- MIDP suppose qu'on a au moins les touches clavier chiffres (0-9), *, #

La gestion du clavier dans un Canvas

- La classe Canvas définit des constantes représentant les touches clavier. On utilise ces constantes avec les arguments de keyXXX()
- Ces constantes sont KEY_NUM0 à KEY_NUM9, KEY_STAR, KEY_POUND (#), et des touches de jeux UP, DOWN, LEFT, RIGHT, FIRE ainsi que GAME_A à GAME_D
- L 'association d'un keyCode (valeur retournée lorsqu'on appuie sur une touche clavier) et sa signification est dépendant mobile. Aussi il faut tester la signification d'une touche clavier par

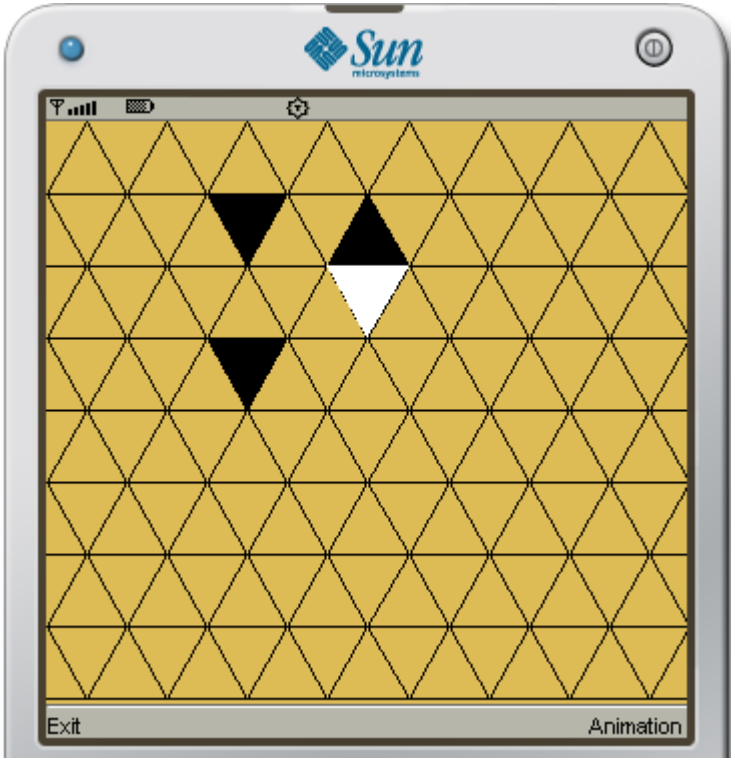
```
protected void keyPressed(int keyCode){
    if (getGameAction(keyCode) == Canvas.FIRE) // etc.
}
```

et pas

```
protected void keyPressed(int keyCode){
    if (keyCode == Canvas.FIRE) // etc.
}
```

La gestion du clavier dans un `Canvas`

- Une démo : projet GoBanDemo (dans répertoire `C:\JeanMarc\CNAM\Recherche\Shanghai2008`)



plug-in J2ME pour Eclipse : MTJ

- C'est Eclipse Mobile Tools for Java (MTJ) anciennement eclipseME (voir à `http://eclipseme.org/`)
- Il faut avoir (évidemment) installé :
  - La JVM Java2 SE, 1.4.x au moins
  - Eclipse 3.2 au moins
  - Un WTK (comme celui de SUN)
- Voir installation (en fait similaire à une install de plug-in sous eclipse) à `http://eclipseme.org/docs/installation.html` puis `http://eclipseme.org/docs/installEclipseME.html`

Utilisation du plug-in J2ME pour Eclipse

- Voir à `http://eclipseme.org/docs/configuring.html`)
- Lorsqu'on veut créer un projet Java ME choisir File | New | Project. Puis dans la fenêtre "New Project", J2ME | J2ME Midlet Suite
- Cliquer les boutons Next en complétant les champs. Si on demande des devices (i.e. les émulateurs), aller les chercher dans le WTK installé (par exemple dans `REP_INSTAL_WTK`)

Bibliographie

- `http://developers.sun.com/mobility/getstart/` : une présentation des diverses JSR de Java ME
- J2ME in a nutshell. Kim Topley ; éditions O'Reilly
- J2ME Wireless Toolkit 2.1 Download à `http://www.oracle.com/technetwork/java/javame/downloads/index.html`
- J2ME, applications pour terminaux mobiles. Bruno Delb ; éditions Eyrolles
- `http://java.sun.com/products/cldc/index.jsp` : page initiale de CLDC

Fin