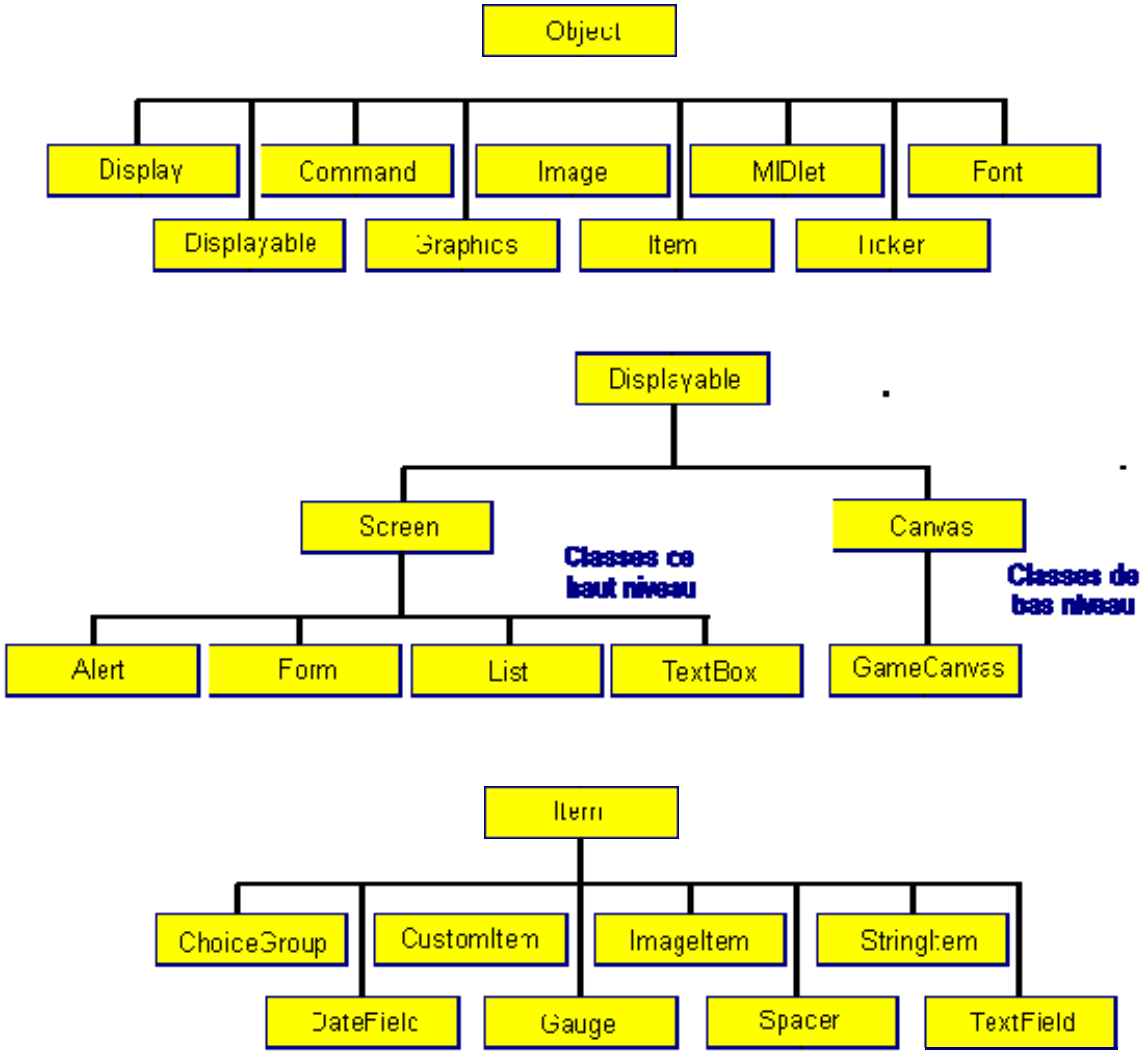
L'interface utilisateur

Généralités

MIDP 2.0 fournit deux paquetages pour l'interface utilisateur : javax.microedition.lcdui (interface commune pour écrans LCD) et javax.microedition.lcdui.game (pour les jeux).

Les classes du paquetage javax.microedition.lcdui sont issues du paquetage java.lang contenant la classe mère Object. Sur le schéma ci-dessous, nous montrons quelques classes du package javax.microedition.lcdui (seule, la classe GameCanvas appartient au package javax.microedition.lcdui.games).



Toutes les classes d'objets affichables héritent de la classe abstraite Displayable. La classe Displayable comporte les sous-classes indiquées dans le schéma précédent. Elle possède 4 méthodes : addCommand(Command cmd), setCommandListener(CommandListener listener), removeCommand(Command cmd) que nous expliciterons plus loin au sujet de la gestion des événements, et isShown().

La classe Display

La gestion de l'affichage est effectuée à l'aide de la classe Display dont les objets représentent une "encapsulation" d'écran de terminal mobile. Ses méthodes principales sont les suivantes :

- public static Display getDisplay(MIDlet mm) : fourniture de l'instance courante d'affichage de la MIDlet mm (il n'y a qu'une seule instance de Display pour chaque Midlet)
- public Displayable getCurrent() : fourniture de l'élément courant affiché
- public void setCurrent(Displayable element) : affichage d'un élément (liste, formulaire,...)
- public void setCurrent(Alert alerte, Displayable prochain_element) : rend courant une alerte et annonce le prochain élément à afficher après le traitement de l'alerte
- public boolean isColor() : retourne "vrai" si le terminal permet l'affichage couleur
- public int numColors() : affichage du nombre de couleurs du terminal si isColor est true ou de nuances de gris si isColor() est false.

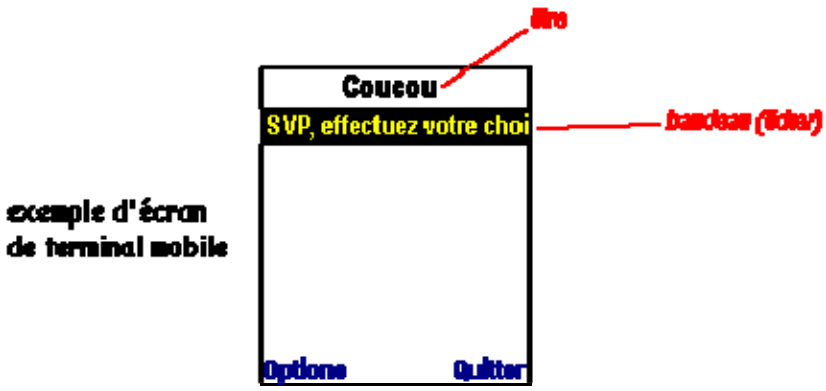
exemple1 : Reprenons le code de l'exemple précédent :

```
package exemple1.intro;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;
public class Hello extends MIDlet
{
    Alert timeAlert;
    public Hello()
    {
        timeAlert = new Alert("Hello !");          ← création d'un élément "Displayable" time
        timeAlert.setString("Bonjour les petits enfants !");
    }
    public void startApp()
    {
        Display.getDisplay(this).setCurrent(timeAlert);   ← affichage de l'élément timeAlert
                                                            défini comme élément courant
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```

La classe Displayable

Il s'agit d'une classe représentant le contenu d'un écran d'affichage. Elle correspond à deux sous-classes : Screen (classes de haut niveau) et Canvas (classes de bas niveau). La classe Displayable procure des méthodes utiles dont nous donnons ci-dessous un échantillon :

public int getWidth() : largeur utile de l'écran (en pixels)
public int getHeight() : hauteur utile de l'écran (en pixels)
public boolean isShown() : un objet affichable est-il présent ?
public void setTitle(String title) : définition d'un titre à afficher
public String getTitle() : récupération du titre
public void setTicker(Ticker ticker) : définition d'un bandeau défilant à afficher
public Ticker getTicker() : récupération du bandeau défilant
public void addCommand(Command cmd) : ajout d'une commande à un élément affichable
public void setCommandListener (CommandListener cl) : définition d'un gestionnaire de commande (remplace un précédent gestionnaire éventuellement)



Les classes de haut niveau

Passons en revue les quatre éléments de haut niveau

- **Alert** : permet d'afficher un message (dit d'alerte) sous forme d'une boîte de dialogue qui s'affiche pendant un temps limité sur l'écran. Le titre est défini à la création : Alert("Hello !") et ne peut être ensuite changé. le message est défini avec la méthode setString(<message>).Le temps d'affichage est défini usuellement par défaut. Si on veut le définir, il faut utiliser la méthode setTimeout(int time) ; si time vaut Alert.FOREVER, le message sera persistant. Les 5 types de messages d'alerte sont définis par la classe AlertType (qui est également dans le paquetage javax.microedition.lcdui) :

ALARM : l'utilisateur est informé d'un événement programmé ;
CONFIRMATION : une confirmation est demandée à l'utilisateur ;
ERROR : l'utilisateur est informé d'une erreur ;
INFO : information de l'utilisateur ;
WARNING : avertissement de l'utilisateur.

On définit le type d'alerte avec la méthode public void setType(AlertType type)

Les types d'alerte ont différentes apparences et peuvent être sonorisés. Une image peut également être associée à une alerte avec la méthode setImage(Image img) ; on peut aussi associer un indicateur à une alerte avec la méthode setIndicator(Gauge jauge).

- **List** : il s'agit de liste d'items (texte et/ou image) présentés sur des lignes successives avec éventuellement des cases à cocher ou des boutons radio. Une liste peut être créée vide et remplie ensuite par ajout ou insertion ou bien être définie à l'avance et affichée comme telle. Une liste permet d'effectuer des choix (interface Choice) ; le mode de choix peut être
    - Choice.EXCLUSIVE (boutons radio : un seul choix est possible),
    - Choice.MULTIPLE (cases à cocher : plusieurs choix possibles),
    - Choice.IMPLICIT (sélection de l'élément sélectionné comme dans un menu déroulant).

exemple 2 : choix de fruits dans une liste de fruits

```
package exemple2;

import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;

public class ListeFruits extends MIDlet
{
   List FList1;
   public ListeFruits()
   {
      FList1 = new List("Liste 1 - Choisissez vos fruits", Choice.MULTIPLE);
      FList1.append("Poire", null);
      FList1.append("Pomme", null);
      FList1.insert(1, "Raisin", null); // insertion de "Raisin" entre "Poire" et "Pomme"
   }
   public void startApp()
   {
      Display display = Display.getDisplay(this);
      display.setCurrent(FList1);
   }
   public void pauseApp()
   {
   }
   public void destroyApp(boolean unconditional)
   {
   }
}
```

exemple 2bis : autre affichage

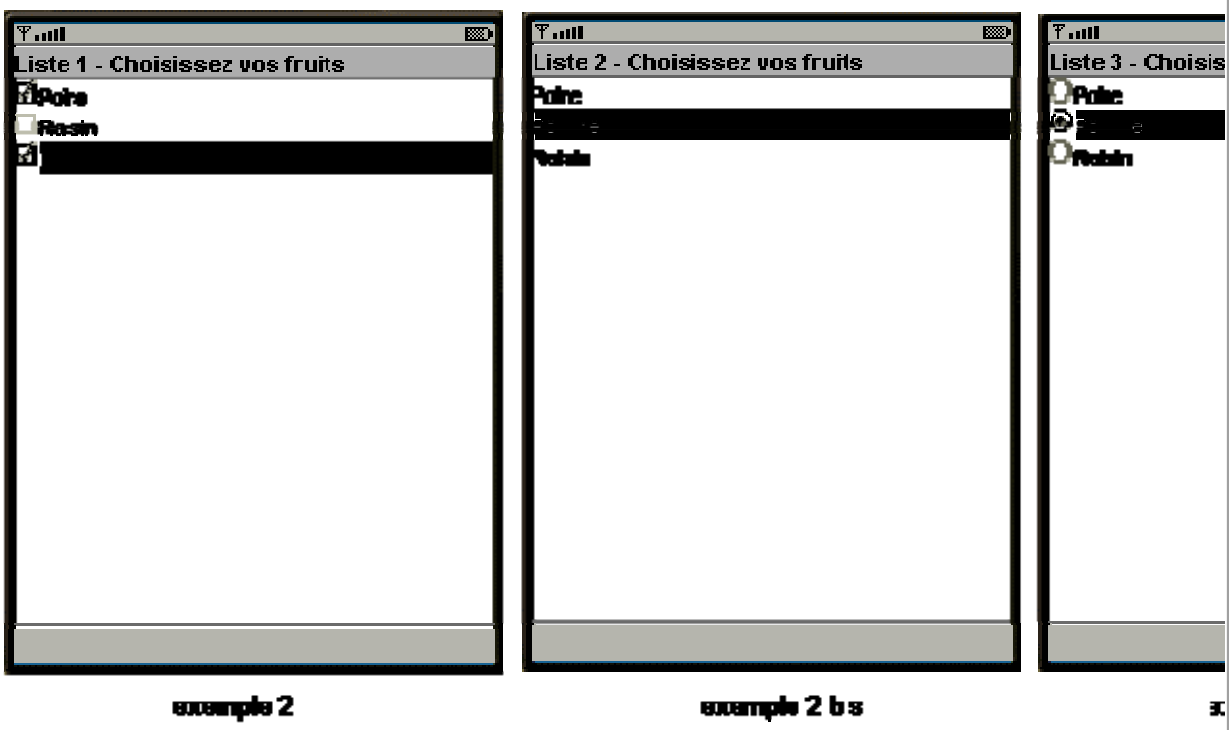
```
package exemple2bis;

import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Choice;
```

```
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;

public class ListeFruits extends MIDlet
{
   List FList2;
   public ListeFruits()
   {
      String fruits2[] = {"Poire", "Pomme", "Raisin"};
      FList2 = new List("Liste 2 - Choisissez vos fruits", Choice.IMPLICIT, fruits2, null);
    }
   public void startApp()
   {
      Display display = Display.getDisplay(this);
      display.setCurrent(FList2);
   }
   public void pauseApp()
   {
   }
   public void destroyApp(boolean unconditional)
   {
   }
}
```

exemple 2ter : encore un autre affichage

```
package exemple2ter;

import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;

public class ListeFruits extends MIDlet
{
   List FList3;
   public ListeFruits()
   {
      String fruits3[] = {"Poire", "Pomme", "Raisin"};
      FList3 = new List("Liste 3 - Choisissez votre fruit", Choice.EXCLUSIVE, fruits3, null);
    }
   public void startApp()
   {
      Display display = Display.getDisplay(this);
      display.setCurrent(FList3);
   }
   public void pauseApp()
   {
```

```
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```



exemple 2 exemple 2 bis

On peut modifier une liste après sa création avec les méthodes delete(int index) et deleteAll().

- **TextBox** : Cette classe permet à l'utilisateur d'éditer du texte. Une boîte de texte a une longueur qui peut être définie par l'utilisateur avec la méthode setMaxSize(<longueur>) où la longueur est spécifiée en nombre de caractères. Toutefois, le terminal ne peut afficher plus de 32 caractères. Le type de texte à entrer dans le TextBox est spécifié grâce à la classe TextField :

- TextField.EMAILADDR : adresses de messagerie
- TextField.UNEDITABLE : texte non éditable
- TextField.NUMERIC : seulement des nombres
- TextField.PHONENUMBER : numéros de téléphone
- TextField.PASSWORD : mot de passe
- TextField.URL : adresse URL
- TextField.SENSITIVE : données sensibles à ne pas mettre dans un dictionnaire ou une table
- TextField.NON_PREDICTIVE : le texte contient des mots non répertoriés dans un dictionnaire
- TextField.INITIAL_CAPS_WORD : la première lettre de chaque mot soit être en majuscule.
- TextField.INITIAL_CAPS_SENTENCE : la première lettre de chaque phrase doit être en majuscule.
- TextField.ANY : tout ce qu'on veux

Ces spécifications sont imposées avec la méthode setConstraints()

exemple 3 : édition de texte par l'utilisateur

```
package exemple3;

import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;

public class Texte extends MIDlet
{
    private TextBox txtBox1;
    private TextBox txtBox2;
    public Texte()
    {
        txtBox1 = new TextBox("Votre nom ?", "", 50, TextField.ANY);
        txtBox2 = new TextBox("Votre code numérique ?", "", 4, TextField.NUMERIC |
TextField.PASSWORD);
    }
    public void startApp()
    {
        Display display = Display.getDisplay(this);
        display.setCurrent(txtBox1);
        try
        {
            Thread.currentThread().sleep(6000);
        } catch(Exception e) {}
        display.setCurrent(txtBox2);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```

exemple 3bis : écriture de texte par le programme : le nom "Alfrd Dupont" est écrit, puis la lettre "e" est insérée entre le "r" et le "d".

```
package exemple3bis;

import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;
```

```
public class Texte extends MIDlet
{
    private TextBox txtBox;
    public Texte()
    {
        txtBox = new TextBox("","",40,TextField.ANY);
    }
    public void startApp()
    {
        Display display = Display.getDisplay(this);
        display.setCurrent(txtBox);
        try
        {
            Thread.currentThread().sleep(5000);
        } catch(Exception e) {}
        txtBox.setString("Alfrd Dupont");
        try
        {
            Thread.currentThread().sleep(3000);
        } catch(Exception e) {}
        txtBox.insert("e", 4); // insertion du caractère "e" à la 5ème position
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```

- **Form** : C'est une notion bien connue, celle de formulaire, qui correspond ici à une collection d'instances de la classe Item. On insère une instance avec la méthode insert(int index, Item newItem) ; on peut aussi modifier une instance par set(int index, Item newItem). Les éléments de formulaire sont au nombre de 8 :

- StringItem : un label (titre et texte) non modifiable
- DateField : date sous trois formats : DATE, TIME, DATE_TIME
- TextField : comme TextBox
- ChoiceGroup : comme List
- Spacer : possibilités d'espacement
- Gauge : simulation d'une barre de progression (une jauge quoi !) ; voir plus loin.
- ImageItem : affichage d'une image, bien sûr
- CustomItem : définition d'éléments spéciaux

exemple4 : illustration des Items de Form

```
package exemple4;

import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;
import javax.microedition.lcdui.Spacer;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.TextField;
import javax.microedition.lcdui.DateField;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;

public class Formulaire extends MIDlet
{
    private Form form;
    private Gauge jauge;
    private Spacer espace;
    private ImageItem image;
    private TextField champ;
    private DateField champdate;
    private StringItem chaine;
    private ChoiceGroup choiceGroup;
    public Formulaire()
    {
        form = new Form("Vos caractéristiques");
        chaine = new StringItem("Voici votre code d'accès : ", "alpha123");
        form.append(chaine);
        champdate = new DateField("Entrez votre date de naissance : ", DateField.DATE);
        form.append(champdate);
        champ = new TextField("Entrez votre nom : ", "", 50, TextField.ANY);
        form.append(champ);
        choiceGroup = new ChoiceGroup("Choisissez votre repas : ", Choice.EXCLUSIVE, new
String[] {"Poulet rôti", "Sole meunière"},  null);
        form.append(choiceGroup);
        espace = new Spacer(20,10); // espace entre les items
        form.append(espace);
        jauge = new Gauge("QI de 1 à 5", true, 5, 1);
        form.append(jauge);
        try
        {
            image = new ImageItem("UPJV : ", Image.createImage("/logoUPJV.png"),
ImageItem.LAYOUT_DEFAULT, "logoUPJV");
            form.append(image);
```

```
        } catch(Exception e) {}
    }
    public void startApp()
    {
        Display display = Display.getDisplay(this);
        display.setCurrent(form);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```

Autres éléments : Images, Tickers et Jauges

La classe Image fournit plusieurs méthodes pour créer ou acquérir des images utilisées dans des MIDlets. Une image peut être de deux types : modifiable (mutable) ou non modifiable (immutable). On reconnaît si une image est modifiable ou non avec la méthode isMutable().

Une image non modifiable ne peut être modifiée après son affichage ; les images non modifiables peuvent être placées dans Alert, Choice, ImageItem. Pour acquérir une image non modifiable, on utilise la méthode createImage(String imageName) :

```
Image img = Image.createImage("labelleimage.png")
```

Il faut noter que les MIDlets emploient des images en format PNG (Portable Network Graphics). Les autres formats (GIF, JPEG) ne sont pas garantis sur tous les portables.

Les images modifiables sont affichées dans un Canvas par appel de l'objet graphique les représentant.

```
Image img = Image.createImage(150, 50);
Graphics g = img.getGraphics();
```

On peut obtenir les dimensions d'une image avec les méthodes :

```
public int getHeight()
public int getWidth()
```

qui retournent des valeurs en nombre de pixels.

Enfin, on peut savoir si une image est modifiable ou non avec la méthode

public boolean isMutable()

dont la réponse est "vrai" si l'image est modifiable.

Un ticker est un élément associé à un élément de l'interface utilisateur qui consiste en un bloc de texte qui défile indéfiniment à l'écran pendant que l'élément associé est affiché. C'est une façon commode d'afficher de l'information sur l'élément associé. Un ticker est créé par la méthode setTicker(Ticker ticker) et son constructeur Ticker(String msg) définit le message à afficher. Le message peut ensuite être modifié par setString(String msg).

exemple 5: affichage d'un bandeau défilant

```
package exemple5;

import javax.microedition.lcdui.Ticker;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;

public class Banderole extends MIDlet
{
    private Form form;
    private Ticker bandeau;

    public Banderole()
    {
        form = new Form("banderole");
        bandeau = new Ticker("Bienvenue sur ce terminal mobile !...");
        form.setTicker(bandeau);
    }
    public void startApp()
    {
        Display display = Display.getDisplay(this);
        display.setCurrent(form);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```

Une jauge peut être non interactive ou interactive comme celle utilisée dans un exemple précédent. Elle peut aussi être interactive dans 4 états : CONTINUOUS_IDLE, INCREMENTAL_IDLE, CONTINUOUS_RUNNING, INCREMENTAL_UPDATING.

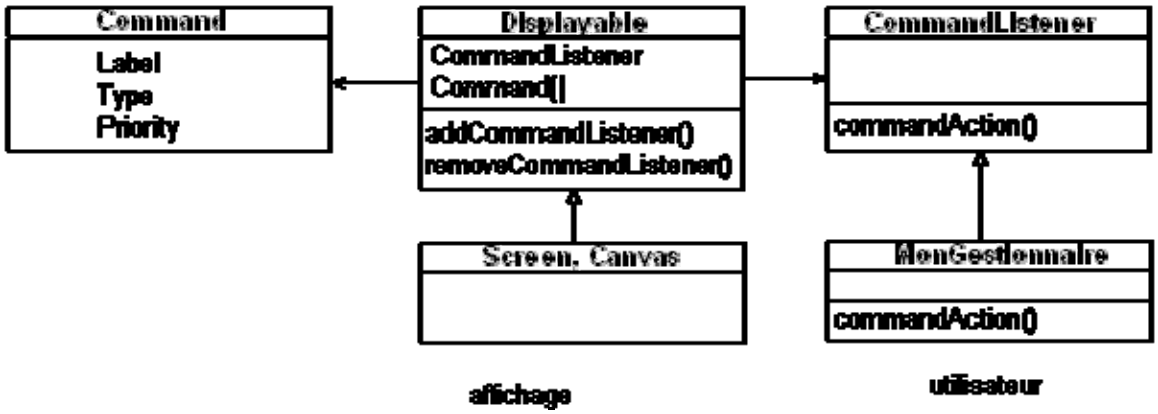
exemple 6 : les jauges

```
package exemple6;

import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;

public class Jauges extends MIDlet
{
    private Form form;
    private Gauge NI_CI;
    private Gauge NI_II;
    private Gauge NI_CR;
    private Gauge NI_IU;
    private Gauge I;

    public Jauges()
    {
        form = new Form("Exemples de jauges");
        NI_CI = new Gauge("non interactive - Continuous_idle", false, Gauge.INDEFINITE,
Gauge.CONTINUOUS_IDLE);
        form.append(NI_CI);
        NI_II = new Gauge("non interactive - Incremental_idle", false, Gauge.INDEFINITE,
Gauge.INCREMENTAL_IDLE);
       form.append(NI_II);
        NI_CR = new Gauge("non interactive - Continuous_running", false, Gauge.INDEFINITE,
Gauge.CONTINUOUS_RUNNING);
       form.append(NI_CR);
        NI_IU = new Gauge("non interactive - Incremental_Updating", false, Gauge.INDEFINITE,
Gauge.INCREMENTAL_UPDATING);
       form.append(NI_IU);
        I = new Gauge("interactive ", true, 10, 0);
        form.append(I);
    }
    public void startApp()
    {
         Display display = Display.getDisplay(this);
         display.setCurrent(form);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```

Pour représenter une jauge, chaque terminal possède ses propres images. Elles diffèrent donc en général d'un terminal à un autre (et en particulier celles du simulateur).

Gestion d'événements

Il faut aussi prévoir l'interaction entre l'utilisateur et l'application. MIDP comporte, pour ce besoin, une interface Listener. On considère deux types d'événements : Command et ItemStateChanged. L'interface utilisateur de MIDP leur fait correspondre les détecteurs (listeners) CommandListener et ItemStateListener respectivement. Chaque objet de la classe Displayable peut être un déclencheur d'événements du type Command ; les objets de la classe Form seulement peuvent être déclencheurs d'événements du type ItemStateChanged.



En se basant sur le schéma précédent, on peut associer à un élément affichable des objets Command. Chacun de ces objets est associé à un élément visuel (comme une touche de fonction) sur l'écran. Quand l'utilisateur agit sur cet élément visuel, un événement est généré sur l'élément courant par appel à la méthode commandAction() enregistrée dans l'objet CommandListener (l'affichage sera changé avec Display.setCurrent() contenues dans commandAction()). On notera que Command contient la description de l'événement tandis que CommandListener contient le traitement à effectuer.

Un événement du type Command est un objet de la classe Command dont le constructeur est

public Command(String label, int typecommande, int priorite)

- label est une chaîne de caractères à afficher représentant la commande à exécuter ;
- typecommande est un entier représentant des événements prédéfinis (Command.BACK, Command.CANCEL, Command.EXIT, Command.HELP, Command.ITEM, Command.OK, Command.SCREEN, Command.STOP) ;
- priorite est un entier représentant le degré d'importance de la commande.

On peut obtenir ces différents paramètres avec les méthodes getCommandType(), getLabel(), getPriority(). A noter que c'est le terminal qui affiche une commande, selon ses propres normes, à partir des trois paramètres précédents. L'action à exécuter est définie par le CommandListener avec la méthode

public void commandAction(Command commande, Displayable element)

où element est la source de l'événement (un objet de la classe Displayable ne peut avoir qu'un seul objet CommandListener) et commande la commande invoquée.

exemple 7 : choix offerts à l'utilisateur

```
package exemple7;

import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Command;
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;

public class Formulaire extends MIDlet implements CommandListener
{
    private Form form;
    private StringItem chaine;
    public Formulaire()
    {
        form = new Form("Votre code d'accès");
        chaine = new StringItem("code : ", "alpha123");
        form.append(chaine);
        form.addCommand(new Command("QUITTER", Command.EXIT,2));
        form.addCommand(new Command("AIDE", Command.HELP,2));
        form.addCommand(new Command("OK", Command.OK,1));
        form.setCommandListener(this);
    }
    public void commandAction(Command com, Displayable d)
    {
        String label=com.getLabel();
        if (label == "QUITTER") notifyDestroyed();
        else if (label == "AIDE") displayHelp();
              else if (label == "OK" ) processForm();
    }
    public void displayHelp()
    {
    }
    public void processForm()
    {
```

```
    }
    public void startApp()
    {
        Display display = Display.getDisplay(this);
        display.setCurrent(form);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```
