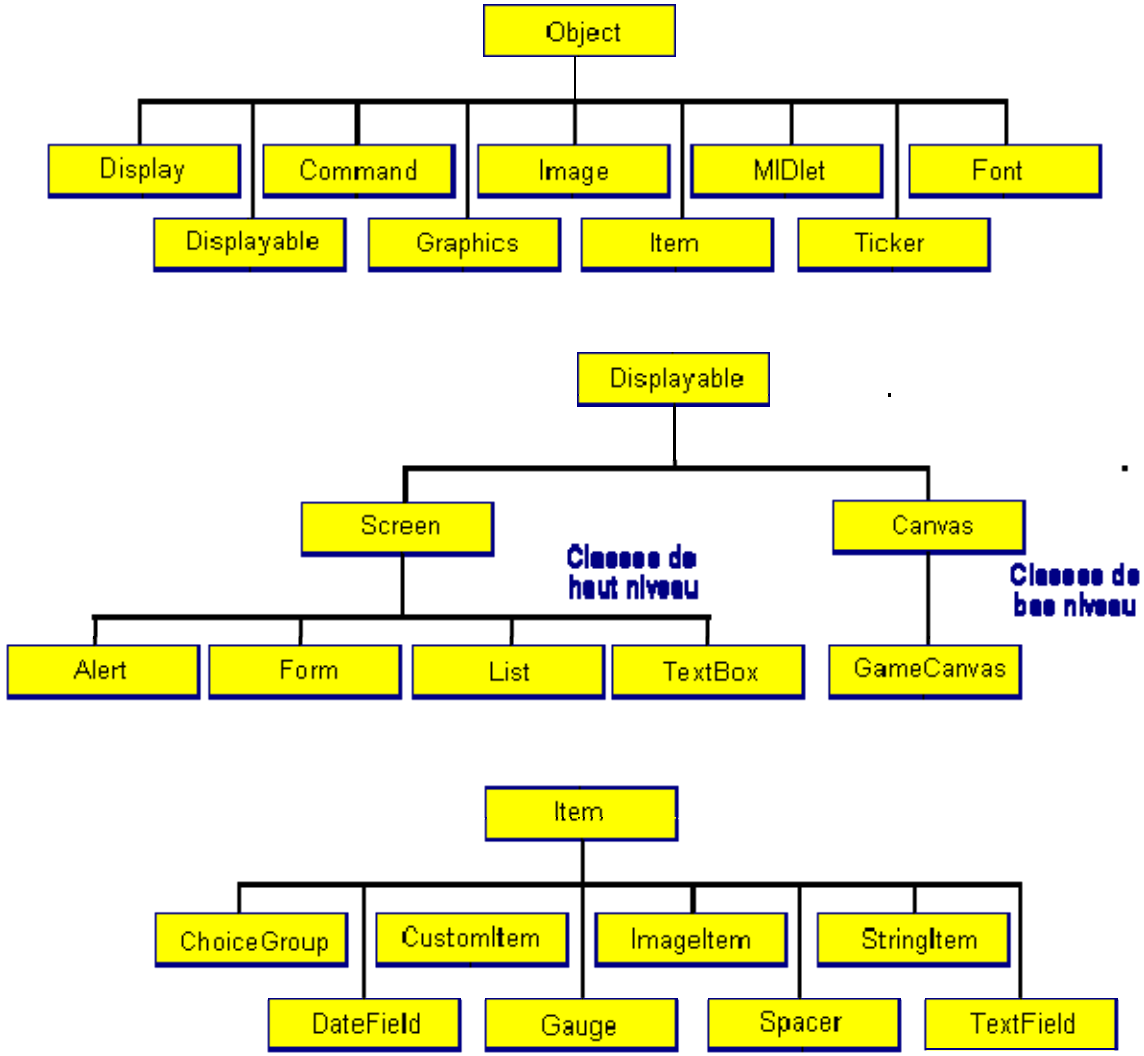
IHM

Présentation

- MIDP 2.0 fournit deux paquetages pour l'interface utilisateur
  - API de haut niveau (indépendante du terminal) javax.microedition.lcdui (interface commune pour écrans LCD)
  - API de bas niveau (dépendante du terminal) javax.microedition.lcdui.game (pour les jeux)



IHM

- API haut niveau
  - Formulaires
  - ≪ Look and feel ≫ minimal, pris en charge par la JVM
  - Affichage indépendant du terminal.
- API bas niveau
  - Contrôle précis de l’affichage et de la position
  - Très dépendante du terminal

La classe Display

- La classe Display représente l'écran d'un terminal et permet la gestion de l'affichage.
- Il existe une seule instance de la classe Display par Midlet, la méthode getDisplay() permet de récupérer sa référence

  public static Display getDisplay(MIDlet m).
- L'élément à afficher doit être passé à l'objet Display via la méthode setCurrent

  public void setCurrent([Alert alerte,] Displayable prochain_element).
- La méthode getCurrent permet de récupérer l'objet en cours d'affichage

  public Displayable getCurrent()
- Autres méthodes:
  - public boolean isColor() : retourne "vrai" si le terminal permet l'affichage couleur
  - public int numColors() : affichage le nombre de couleurs du terminal si isColor est true ou de nuances de gris si isColor() est false.

La classe Displayable

- les classes affichables héritent de la classe abstraite Displayable.
- Méthodes:
  - 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: Alert

Alert : affiche un message 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 :

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)

TextBox

- **TextBox**(String title, String text, int maxSize, int constraints);
- 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.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

List

- Représente une liste d'éléments (texte et/ou image)avec éventuellement des cases à cocher ou des boutons radio. Une liste peut être créée vide et remplie ensuite par ajout (méthode append) 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).

Form

- Un formulaire peut contenir une collection d'instances de la classe Item.
- Création:
  - **Form**(String title)
    Creates a new, empty Form.
  - **Form**(String title, Item[] items)
    Creates a new Form with the specified contents.
- La méthode append permet d'ajouter un élément de type Item dans le formulaire
- On insère une instance avec la méthode insert(int index, Item newItem) .
- On modifie 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
    - ImageItem : affichage d'une image, bien sûr
    - CustomItem : définition d'éléments spéciaux

Les commandes

- MIDP comporte, une interface Listener. On considère deux types d'événements:
  - Command
  - 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 de type Command ; les objets de la classe Form seulement peuvent être déclencheurs d'événements du type ItemStateChanged.
- Une commande possède les propriétés suivantes:
  - label : texte représentant la commande à exécuter
  - type:entier représentant des événements prédéfinis (Command.BACK, Command.CANCEL, Command.EXIT, Command.HELP, Command.ITEM, Command.OK, Command.SCREEN, Command.STOP) .
  - priorite: entier représentant le degré d'importance de la commande (1 reflète la plus propriété la plus élevée).
- On peut obtenir ces différents propriétés avec les méthodes getCommandType(), getLabel(), getPriority().

Les commandes

- Constructeur

  public Command(String label, int typecommande, int priorite)
- on peut associer à un élément affichable des objets Command à l'aide de la méthode **addCommand().**
- La méthode setCommandListener permet d'attacher un écouter à un objet affichable.
- Le Listener doit implémenter l'interface CommandListener

  public void commandAction(Command commande, Displayable element)

  où element est la source de l'événement et commande la commande invoquée.

Image et jauge

- 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().

- 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.

Form

- Un objet de type Form contient une collection de champs de type Item
- Gestion des champs d'un formulaire
  - insert(int index, Item newItem): pour insérer un champ à une position spécifiée par "index" dans le formulaire
  - set(int index, Item newItem): Pour remplacer un champ par un autre
  - append(Item i): pour ajouter un champ dans le formulaire.
- 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
    - d'une image, bien sûr
    - CustomItem :TextField : comme TextBox
    - ChoiceGroup : comme List
    - Spacer : possibilités d'espacement
    - Gauge : simulation d'une barre de progression
  - **Gauge**(String label, boolean interactive, int maxValue, int initialValue)
    - Interactive: l'utilisateur peut modifier la valeur
    -
      - ImageItem : affichage définition d'éléments spéciaux

Exemple 1

```
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 : ", Dat
//champdate=new DateField(null, DateField.TIME,TimeZone.getTimeZon
form.append(champdate);
champ = new TextField("Entrez votre nom : ", "", 50, TextField.ANY
form.append(champ);
choiceGroup = new ChoiceGroup("Choisissez votre repas : ", Choice
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
form.append(image);
} catch(Exception e) {}
```

Exemple 2

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Compteur extends MIDlet implements CommandListener {
public void commandAction(Command c, Displayable d) {
if (c.getLabel().equals("Incrémenter")) {
int v = Integer.parseInt(t.getString());
v++;
t.setString(String.valueOf(v));
} else {
int v = Integer.parseInt(t.getString());
v--;
t.setString(String.valueOf(v));
}
}
TextBox t = new TextBox("Compteur", "0", 5, TextField.NUMERIC);
public Compteur() {
Command cmd1, cmd2;
cmd1 = new Command("Incrémenter", Command.OK, 1);
cmd2 = new Command("Décrémenter", Command.OK, 2);
t.addCommand(cmd2);
t.addCommand(cmd1);
t.setCommandListener(this);
}
public void startApp() {
Display.getDisplay(this).setCurrent(t);}
public void pauseApp() {}
public void destroyApp(boolean unconditional) {}
}
```

Exemple 3

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Liste extends MIDlet implements
    CommandListener {
TextBox t = new TextBox("Ajouter un élément", "0",
    10, TextField.ANY);
List liste = new List("Eléments", List.EXCLUSIVE);
public Liste() {
Command cmd1, cmd2, cmd3;
cmd1 = new Command("Ajouter", Command.OK, 1);
cmd2 = new Command("Retour", Command.BACK, 1);
cmd3 = new Command("Afficher", Command.OK, 1);
t.addCommand(cmd1);
liste.addCommand(cmd3);
liste.addCommand(cmd2);
t.setCommandListener(this);
liste.setCommandListener(this);
}
public void startApp() {
Display.getDisplay(this).setCurrent(t);
}
public void pauseApp() {
}
public void destroyApp(boolean unconditional) {
}
public void commandAction(Command c, Displayable
    d) {
if (c.getLabel().equals("Ajouter")) {
liste.append(t.getString(), null);
Display.getDisplay(this).setCurrent(liste);
} else if (c.getLabel().equals("Retour")) {
Display.getDisplay(this).setCurrent(t);
} else {
String selection =
    liste.getString(liste.getSelectedIndex());
t.setString(selection);
Display.getDisplay(this).setCurrent(t);
}
}
}
```