Wireless Online Game Development for Mobile Device

Lesson 3

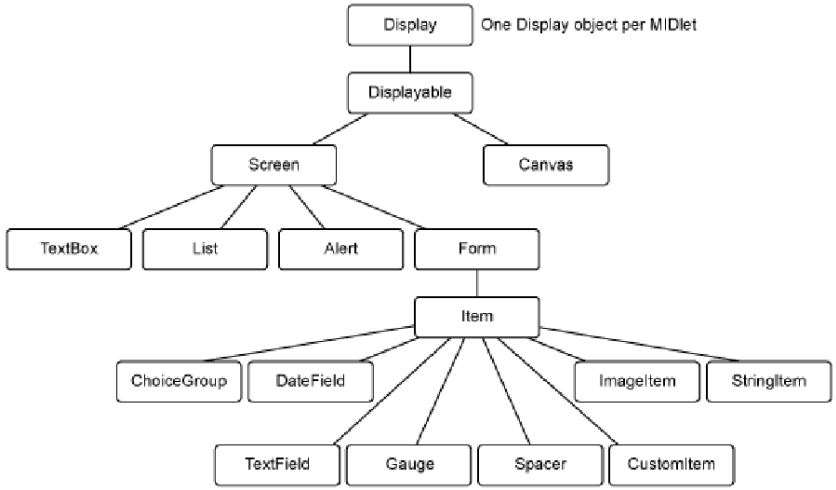
What have we learnt last week?

- High-Level User Interface Object: TextBox, List, and Alert
- Command Button and Event Handling
- The Random Function

Form and Item Components

- A Form is essentially a container to hold other components, where each component is a subclass of the Item class.
- In the panels that follow the device-display components that comprise MIDP's high-level API, namely:
  - DateField
  - Gauge
  - StringItem
  - TextField
  - ChoiceGroup
  - Spacer
  - CustomItem
  - ImageItem

The Complete Device Display Hierarchy



The StringItem Component

- A StringItem component is used to display a label or text string.
- Because a user cannot change either the label or the text when the application is running, a StringItem does not recognize events.

Constructor Syntax

- To create a string in a Form, the basic syntax is:
  - ***StringItem (String label, String text, int appearanceMode)***
- Parameters:
  - "label" is the StringItem's label, or null if no label
  - "text" is the StringItem's text contents, or null if the contents are initially empty
  - "appearanceMode" is the appearance mode of the StringItem (Item.PLAIN, Item.HYPERLINK, or Item.BUTTON)

Example

- StringItem strMessage = new StringItem("Game Center:", "Please Login");

The TextField Component

- A Textfield is analogous to any typical text entry field. You can specify a label, the maximum number of characters, and the data type you accept.
- The TextField component also implements a password modifier that masks characters as they are input.

Constructor Syntax

- To create a text field in Form, the basic syntax is:
  - ***TextField (String title, String text, int MaxSize, int Constraint)***
- Parameters:
  - "title" is the title of the textbox
  - "text" is the initial text appears on the screen
  - "MaxSize" is the maximum length of characters
  - "Constraints" is used to restrict the user's input

TextField Constraints

- MIDP defines the following constraint parameters for the TextField component:
  - ANY allows any characters.
  - EMAILADDR allows only valid email addresses.
  - NUMERIC allows any numeric value.
  - PHONENUMBER allows only phone numbers.
  - URL allows only characters that are valid within URL.
  - PASSWORD masks all characters as they are input.

Example

- TextField txtUserName = new TextField ("User Name:", "", 8, TextField.ANY);

The DateField Component

- A DateField is an editable component for presenting calendar (date and time) information that may be placed into a Form.
- Value for this field can be initially set or left unset.
- Calendar calculations in this field are based on default locale and defined time zone. Because of the calculations and different input modes date object may not contain same millisecond value when set to this field and get back from this field.

The DateField Component

- The DateField component provides a means to visually manipulate a Date object, as defined in **java.util.Date**.
- When creating a DateField object you specify whether the user can edit the date, the time, or both.

Syntax

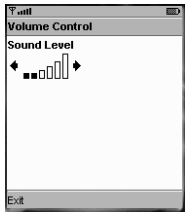
- To create a DateField, the basic syntax is:
  - ***DateField(String label, int mode, TimeZone timeZone)***
- Parameters:
  - **label** – item label
  - **mode** – the input mode, one of DATE, TIME or DATE_TIME
  - **timeZone** – a specific time zone, or null for the default time zone

Example

- DateField dfCalendar = new DateField ("Set Alarm Time", DateField.DATE_TIME);

The Gauge Component

- A Gauge component displays a progress-meter style interface. There are two types of Gauge: interactive and non-interactive.
  - The Interactive Gauge component allows the user to make changes to the gauge.
  - The Non-interactive Gauge component requires the developer to update the gauge.



The Interactive Gauge Component

- The user is allowed to modify the value.
- The user will always have the means to change the value up or down by one and may also have the means to change the value in greater increments.
- The user is prohibited from moving the value outside the established range.
- The expected behavior is that the application sets the initial value and then allows the user to modify the value thereafter.

The Non-Interactive Gauge Component

- The user is prohibited from modifying the value.
- Non-interactive mode is used to provide feedback to the user on the state of a long-running operation.
- One expected use of the non-interactive mode is as a "progress indicator" or "activity indicator" to give the user some feedback during a long-running operation.
- The application may update the value periodically using the **setValue**() method.

The Non-Interactive Gauge Component

- A non-interactive Gauge can have a definite or indefinite range.
- If a Gauge has definite range, it will have an integer value between zero and the maximum value set by the application, inclusive.
- A non-interactive Gauge that has indefinite range will exist in one of four states: ***Continuous-idle, Incremental-idle, Continuous-running***, or ***Incremental-updating***. These states are intended to indicate to the user that some level of activity is occurring.

States for the Non-Interactive Gauge Component

- In the Continuous-idle or Incremental-idle state, the Gauge indicates that no activity is occurring.
- In the Incremental-updating state, the Gauge indicates activity, but its graphical representation should be updated only when the application requests an update with a call to setValue().
- In the Continuous-running state, the Gauge indicates activity by showing an animation that runs continuously, without update requests from the application.

Notes for Developers

- If the application can observe the progress of the operation as it proceeds to an endpoint known in advance, then the application should use a non-interactive Gauge with a definite range.
- If the application cannot observe the progress of the operation, it should use a non-interactive Gauge with indefinite range. The application should call setValue(INCREMENTAL_UPDATING) periodically, perhaps each time its input buffer has filled. This will give the user an indication of the rate at which progress is occurring.

Notes for Developers

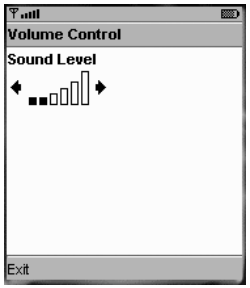
- If the application is performing an operation but has no means of detecting progress, it should set a non-interactive Gauge to have indefinite range and set its value to CONTINUOUS_RUNNING or CONTINUOUS_IDLE as appropriate.
- An application using the Gauge as a progress indicator should typically also attach a STOP command to the container containing the Gauge to allow the user to halt the operation in progress

Syntax

- To create a Gauge, the basic syntax is:
  - ***Gauge (String label, boolean interactive, int maxValue, int initialValue)***
- Parameters:
  - **label** - the Gauge's label
  - **interactive** - whether the user can change the value
  - **maxValue** - the maximum value, or INDEFINITE
  - **initialValue** - the initial value in the range [0..maxValue], or one of CONTINUOUS_IDLE, INCREMENTAL_IDLE, CONTINUOUS_RUNNING, or INCREMENTAL_UPDATING if maxValue is INDEFINITE

Example

- Gauge gaVolume = new Gauge ("Sound Level", true, 6, 2);



The ImageItem and Image Components

- The Image class is used to hold graphical image data.
- They exist only in off-screen memory and will not be painted on the display unless an explicit command is issued by the application (such as within the paint() method of a Canvas) or when an Image object is placed within a Form screen or an Alert screen and that screen is made current.

The ImageItem and Image Components

- Two classes are used to work with display images: Image and ImageItem.
  - **Image** is used to create an image object and holds information such as the height and width, and whether or not the image is mutable.
  - **ImageItem** defines how an image will be displayed; that is, whether the image will be centered, to the left, at the top of the screen, etc.

The ImageItem and Image Components

- MIDP offers two types of images: immutable and mutable.
  - Immutable image cannot be changed once it has been created. Typically, this type of image is read from a resource such as a file.
  - Mutable image is essentially a chunk of memory. It is up to you to create the contents of the image by writing it into the memory block.

Notes for Developers

- The high-level user interface implementation may need to update the display at any time, without notifying the application.
- In order to provide predictable behavior, the high-level user interface objects provide snapshot semantics for the image.

Notes for Developers

- When a mutable image is placed within an Alert, Choice, Form, or ImageItem object, the effect is as if a snapshot is taken of its current contents. This snapshot is then used for all subsequent painting of the high-level user interface component.
- If the application modifies the contents of the image, the application must update the component containing the image (for example, by calling ImageItem.setImage) in order to make the modified contents visible.

Syntax

- To create an image, the basic syntax is:
  - ***createImage(Image source)***
- Parameters:
  - **source** - the source image file to be copied

Syntax

- To create a new image item object, the basic syntax is:
  - ***ImageItem(String label, Image image, int layout, String altText, int appearanceMode)***
- Parameters:
  - **label** – the label string
  - **image** – the image, can be mutable or immutable
  - **layout** – a combination of layout directives
  - **altText** – the text that used in place of the image
  - **appearanceMode** – the appearance mode of the ImageItem (Item.PLAIN, Item.HYPERLINK or Item.BUTTON)

Example

- The following code shows how we could create an image from a file, associate it with an ImageItem object, and append the image onto a Form.

```
Form fmMain = new Form("Images");
...
// Create an image
Image img = Image.createImage("/picture.png");
// Append to a form
fmMain.append(new ImageItem(null, img,
ImageItem.LAYOUT_CENTER, null));
```

Color or Grayscale?

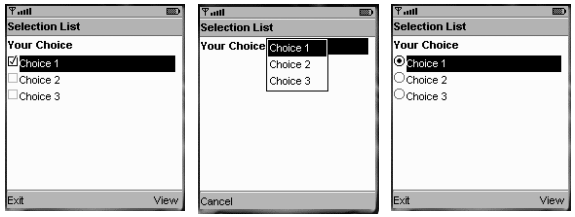
- To display the appropriate image based on whether or not a device supports color.
- When the application is started the appropriate image will be read from the JAR file, as shown below:

```
// Read the appropriate image based on color support
Image im = Image.createImage((display.isColor()) ?
"/image_color.png":"/image_bw.png");

// Create ImageItem and append to the form
fmMain.append(new ImageItem(null, im,
ImageItem.LAYOUT_CENTER, null));
```


The ChoiceGroup Component

- The ChoiceGroup components allow a user to select from a predefined list of entries.
- It is a group of selectable elements intended to be placed within a Form.



The ChoiceGroup Component

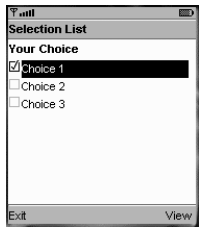
- The implementation is responsible for providing the graphical representation of these modes and must provide visually different graphics for different modes.
  - For example, it might use "radio buttons" for the single choice mode and "check boxes" for the multiple choice mode
- There are two ChoiceGroup formats:
  - Exclusive-selection, which are essentially radio groups
  - Multi-selection, which are commonly referred to as checkboxes

Syntax

- To create a ChoiceGroup, the basic syntax is:
  - ***ChoiceGroup(String label, int choiceType, String[] stringElements, Image[] imageElements)***
- Parameters:
  - **label** – the item's label
  - **choiceType** – EXCLUSIVE, MULTIPLE, or POPUP
  - **stringElements** – set of strings specifying the string parts of the ChoiceGroup elements
  - **imageElements** – set of images specifying the image parts of the ChoiceGroup elements

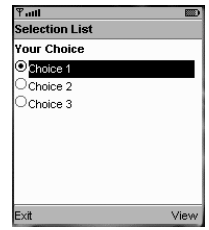
Example – Checkbox

- ChoiceGroup cgPrefs = new ChoiceGroup ("Your Choice", Choice.MULTIPLE);



Example – Radio Button

- ChoiceGroup cgPrefs = new ChoiceGroup ("Your Choice", Choice.EXCLUSIVE);



Example – Pop-up

- ChoiceGroup cgPrefs = new ChoiceGroup ("Your Choice", Choice.POPUP);