Wireless Online Game Development for Mobile Device

Lesson 2

What have we learnt last week?

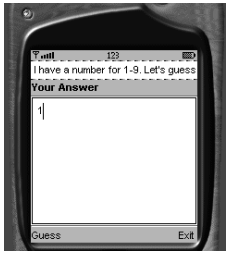
- The Basic Concept of J2ME
- Installation and Configuration of J2ME Wireless Toolkit and Eclipse SDK
- The Basic Structure of a MIDP program
- Write a Sample Program

User Interface Design

- The user interface requirements for mobile devices are different from those for desktop computers.
- The display size of handheld devices is smaller, and the input devices do not always include pointing devices (a mouse or pen input).
- You cannot follow the same user interface programming guidelines for applications running on desktop computers and hand-held devices.

Structure of the MIDP User Interface API

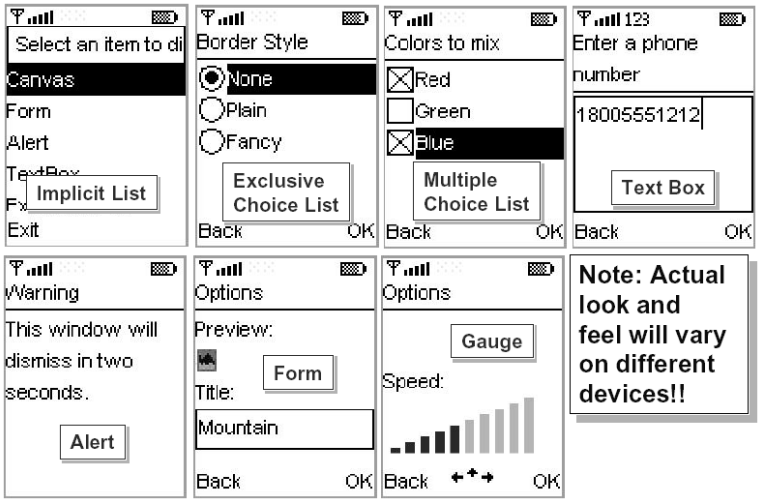
- The MIDP User Interface consists of high-level and low-level APIs.



High-Level API

- Designed for applications whose client parts run on mobile devices that portability is important.
- Employs a high-level abstraction.
- Gives you little control over its look and feel.
- Interaction with components is encapsulated by the implementation and the application is not aware of such interactions.
- The underlying implementation does the necessary adaptation to the device's hardware and native user interface style.
- Implemented by classes that inherit from the **Screen** class.

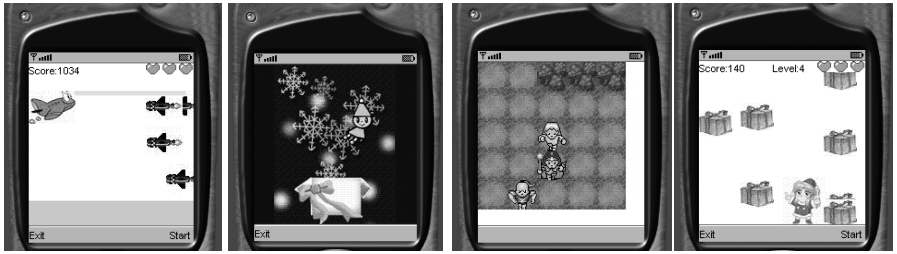
Example of High Level UI



Low-Level API

- Designed for applications that need precise placement and control of graphic elements and access to low-level input events.
- Provides little abstraction.
- Gives the application full control over what is being drawn on the display.
- Implement by the **Canvas** and **Graphics** classes.
- MIDlets that access the low-level API are not guaranteed to be portable because this API provides mechanisms to access details that are specific to a particular device.

Example of Low Level UI



The MIDP GUI Programming Model

- The central abstraction of the MIDP User Interface is a **Screen**, which is an object that encapsulates device-specific graphics rendering user input.
- Only one screen can be visible at a time, and the user can traverse only through the items on that screen.
- The main reason for the screen-based design is that mobile information devices have different display and keypad solutions.
- If an application has to be aware of component layout, scrolling, and focus traversal it compromises portability.

Three Types of Screens:

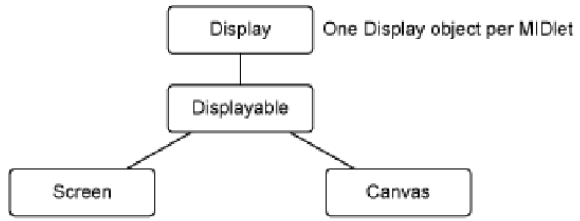
- Screens that encapsulate a complex user interface component that involves a **List** or **TextBox** component.
  - The structure of these screens is predefined, and the application cannot add other components to these screens.
- Generic screens that use a **Form** component.
  - The application can add text, images, and a simple set of related UI components to the form.
- Screens used within the context of the low-level API, such as a subclass of the **Canvas** class.

How to Display?

- The Display class is the display manager that is instantiated for each active MIDlet and provides methods to retrieve information about the device's display capabilities.
- A screen is made visible by calling the **Display.setCurrent()** method.

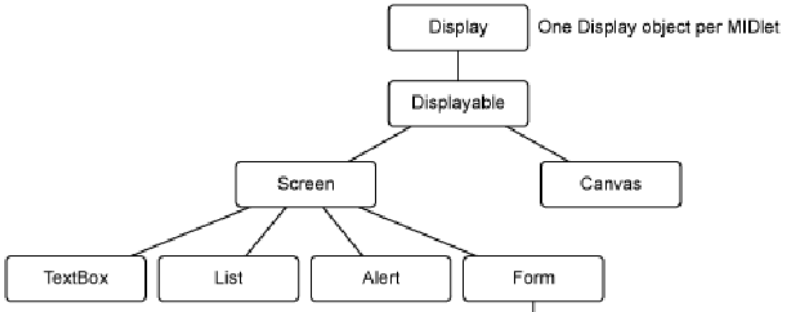
Displayable Object

- Although there is only one Display object per MIDlet, many objects within a MIDlet may be displayable (Forms, TextBoxes, ChoiceGroups, etc…)
- A Displayable object is a component that is visible on a device.
- MIDP contains two subclasses of Displayable: **Screen** and **Canvas**.



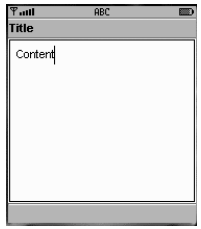
Screen Object

- A Screen object is not something that is visible on the device. Screen is subclassed by high-level components, which the end-user then interacts with on the display.



Text Box

- TextBox components are used to multiple-line input.
- The TextBox and TextField components share the same constraints for specifying the type of content allowed.



Constructor Syntax

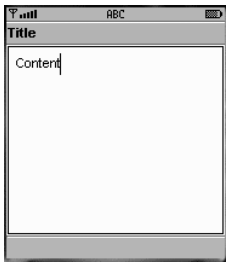
- To create a text box, the basic syntax is:
  - ***TextBox (String title, String text, int MaxSize, int constraint)***
- Parameters:
  - "title" is the title of the textbox
  - "text" is the initial text appears on the screen
  - "MaxSize" is the maximum length of characters
  - "Constraints" is used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

Constraint Setting

Constraint	Description
ANY	The user is allowed to enter any text.
EMAILADDR	The user is allowed to enter an email address.
NUMERIC	The user is allowed to enter only an integer value.
PHONENUMBER	The user is allowed to enter a phone number.
URL	The user is allowed to enter a URL.
PASSWORD	The text entered must be masked so that the characters typed are not visible.

Example

- TextBox txtHeadling = new TextBox (“Title", “Content", 50, TextField.ANY);



Event Handling

- When an event occurs on a mobile device, a Command object holds information about that event.
- This information includes the type of command executed, the label of the command, and its priority.
- In J2ME, commands are commonly represented with soft-buttons on the device.
- The figure shows two Command objects, one with the label "Exit" and one with label “Start“.

Multiple Command Handling

- If there are too many commands to be shown on the display, a device will create a menu to hold multiple commands.

Event processing with Command objects

- The only MIDP components that can manage commands are **Form**, **TextBox**, **List** and **Canvas**.
- The basic steps to process events with a Command object are as follows:
  - Create a Command object.
  - Add the Command to a Form, TextBox, List, or Canvas.
  - Create a listener.
- Upon detection of an event, the listener will generate a call to the method **commandAction()**. Within this method you can determine which command generated the event and process it accordingly, as shown in the next panel.

Constructor Syntax

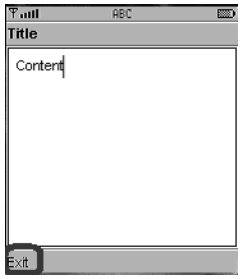
- To create a command button, the basic syntax is:
  - ***Command (String label, int commandType, int priority) ;***
- Parameters:
  - "label" is the title of the Command
  - "commandType" used to specify the intent of this command
  - "priority" is the priority for display

Command Type Setting

Command Type	Description
BACK	Back to previous screen
CANCEL	Cancel the current dialog screen
EXIT	Exit the application
HELP	Request for online help
OK	Confirm the current dialog screen
STOP	Stop the current application
SCREEN	Specifies an application-defined command that pertains to the current screen. (E.g. "Load", "Save", etc …)

Example

- Command cmdExit = new Command ("Exit", Command.EXIT, 0);



Detecting Command Events

- When we choose to exit the MIDlet by pressing the soft-button with the Exit label, a Command event will be generated.
- For each Command event, the **commandAction()** method is called.

Example

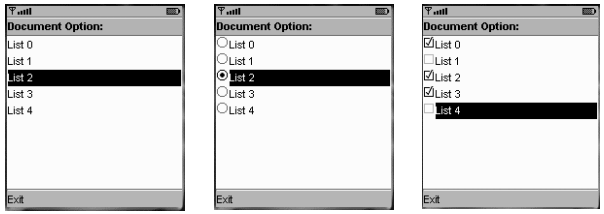
```
// Create Command object, with label, type and priority
Command cmExit= new Command("Exit", Command.EXIT, 0);
    ...
// Create Textbox and give it a title
TextBox txtHeadling = new TextBox ("Title", "Content", 50, TextField.ANY);

txtHeadling.addCommand(cmExit);          // Add Command
txtHeadling.setCommandListener(this);    // Listen for command
...
// Handle user command
void commandAction(Command c, Displayable s) {
  if (c == cmExit) {
    destroyApp(true);
    notifyDestroyed();
  }
}
```


The List Component

- The notion of a select operation on a List element is central to the user's interaction with the List.
- On devices that have a dedicated hardware **[Select]** or **[Go]** key, the select operation is implemented with that key.

The List Component

- List objects may be created with Choice types of
  - Choice.IMPLICIT (List)
  - Choice.EXCLUSIVE (Radio Button)
  - Choice.MULTIPLE (Check Box)

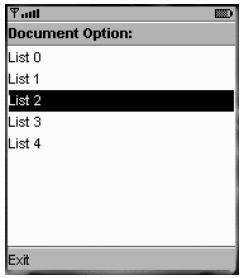


Constructor Syntax

- To create a list, the basic syntax is:
  - ***List (String Title, int ListType, String[] StringElements, Image[] ImageElements)***
- Parameters:
  - “label" is the title of the Command
  - “Title” is the title of the screen
  - “ListType” the type of list (IMPLICIT, EXCLUSIVE, or MULTIPLE)
  - “StringElements” is list of string to be display
  - “ImageElements” is the list of image to be display

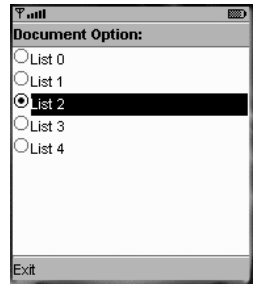
Example – Implicit List

- String StringList[] = {"List 0", "List 1", "List 2", "List 3", "List 4"};
- List ImplicitList = new List ("Document Option:", List.IMPLICIT, StringList, null);



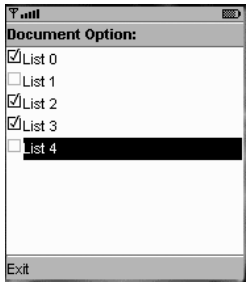
Example – Exclusive List

- String StringList[] = {"List 0", "List 1", "List 2", "List 3", "List 4"};
- List ExclusiveList = new List ("Document Option:", List.EXCLUSIVE, StringList, null);



Example – Multiple List

- String StringList[] = {"List 0", "List 1", "List 2", "List 3", "List 4"};
- List MultipleList = new List ("Document Option:", List.MULTIPLE, StringList, null);



Selection in IMPLICIT Lists

- When the user performs the select operation, the system will invoke the select command by notifying the ***CommandListener*** of the List
- The default select command is the system-provided command ***SELECT_COMMAND***.
- The select command may be modified by the application through use of the ***setSelectCommand*** method.

Example

```
public void commandAction(Command c, Displayable s) {
    if (c == List.SELECT_COMMAND) {
      switch (ImplicitList.getSelectedIndex()) {
        case 0:
        …
        case 1:
        …
      }
    }
}
```


The Alert

- An Alert is essentially a very scaled-down dialog box.
- There are two types of Alert:
  - ***Modal***: displays the dialog until acknowledged by the user
  - ***Non-modal***: displays for a specified number of seconds.

The Alert

- To create a alert dialog, the basic syntax is:
  - ***Alert (String title, String alertText, Image alertImage, AlertType alertType)***
- Parameters:
  - "title" is the title string
  - "alertText" is the string contents
  - "alertImage" is the image contents
  - "alertType is the type of the Alert

The AlertType Component

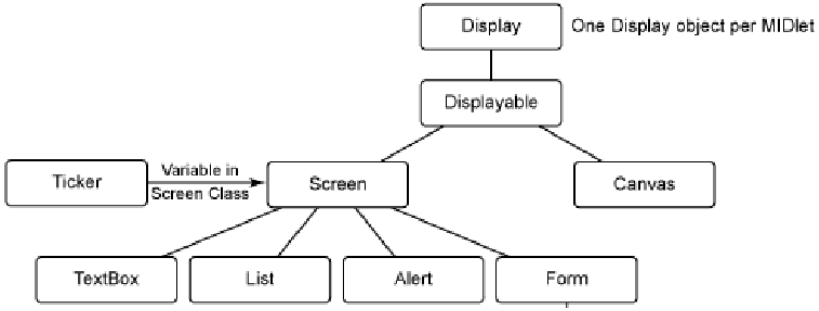
- The ***AlertType*** component uses sound to notify the user of an event.
- You could program the ***AlertType*** to play a particular sound to signal a user error.
- The component comes with five pre-defined sounds:
  - Alarm
  - Confirmation
  - Error
  - Info
  - Warning

Example

- Alert al = new Alert ("List 0", "Alarm Sound", null, AlertType.ALARM);

The Ticker Component

- Ticker component presents a horizontal scrolling banner.
- You can specify the text message to display
- The rate and direction of the scrolling is determined by the device implementation.

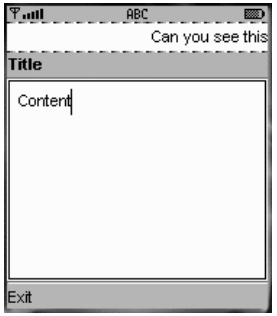


Constructor Syntax

- To create a ticker, the basic syntax is:
  - ***Ticker(String str)***
- Parameters:
  - "str" is string to be set for the Ticker

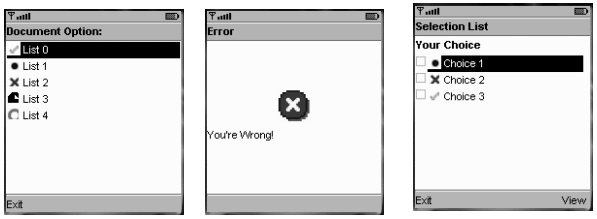
Example

- Ticker myTicker = new Ticker ("Can you see this Ticker on your screen?");



Where can you place Image?

- Images may be placed within Alert, Choice, Form, or ImageItem objects.



Random Function

- MIDlet does not provide direct function to generate a random number.
- To generate a "real" random number, you need to:
  - Import the Java Random Class
    - ***import java.util.Random;***
  - Declare the Random seek
    - ***private Random rand = new Random();***
  - Generate the random number
    - ***RandonNumber = Math.abs(rand.nextInt())%Range;***