CTIS 487: Lecture Notes 4

User Interface Introduction

User-interface requirements for small handheld devices are different from personal computers. Because comparatively the display size of handheld devices is smaller. That’s why we cannot follow the personal computers user-interface programming guidelines for handheld devices. In JME, the CLDC itself does not define any GUI functionality. The official GUI classes for the JME are included in profiles such as the MIDP. The GUI classes in the MIDP are not based on the Abstract Window Toolkit (AWT). The limited CPU memory of handheld devices, cannot handle the AWT. The MIDP contains its own GUI guidelines. The MIDP GUI consists of high-level and low-level APIs, each with their own set of events.

API's

MIDP offers a high-level API objects for user interface development. This is used to build common user-interface components such as Forms, TextBox and similarly others.

Display Object

A MIDlet has one instance of a Display object. This object is used to obtain information about the current display and includes methods for requesting the objects being displayed. The Display object is essentially the manager of the device’s display. Display is represented by the javax.microedition.lcdui.Display class. The Display class is the one and only display manager that is instantiated for each active MIDlet and provides methods to retrieve information about the device’s display capabilities. A reference to the device’s display can be obtained by providing a MIDlet reference to the static getDisplay() method in the startApp() method of a MIDlet.

    public Display getDisplay(MIDlet lMidlet)
Once we have a reference then we can create GUI component. The two forms of the `setCurrent()` method with this reference as parameter are shown here.

```java
setCurrent(Displayable dispRef)
setCurrent(Alert aRef, Displayable dispRef);
```

**Displayable Object**

There is only one Display object per MIDlet, but many objects within a MIDlet may be displayable. A Displayable object is a component that is visible on a device. MIDP contains two subclasses of Displayable: Screen and Canvas.

```java
abstract public class Displayable
public abstract class Canvas extends Displayable
public abstract class Screen extends Displayable
```

We can find out that, what is currently being displayed on the device. We can also find the types of colors the display object supports. Here are the methods.

```java
public Displayable getCurrent( )
// Returns TRUE if it supports Color, FALSE if it Supports grayscale
public void boolean isColor( )
// Returns number of colors it supports
public int numColors( )
```

**DISPLAYABLE : Screen Object**

User interacts with the device through Screen. Screen combines and organizes graphics objects and manages user input through the device. Screens are represented by the `javax.microedition.lcdui.Screen` object. Then by calling `setCurrent()` they will shown by the Display object. There can be several screens in an application, but only one screen at a time can be visible in a display. There are four types of screens, TextBox, List, Alert, and Form. Screen can have two characteristics: title and ticker.
The title is a string that appears above the screen contents. The following methods are used to set and retrieve the title of the screen.

```java
public void setTitle(String title)
public String getTitle()
```

The Ticker is an image that stays above the title. The Ticker class implements a ticker tape. Starting and Stopping the Ticker is not done using methods. The string associated with Ticker scrolls continuously, whose direction of scrolling and the speed of the scrolling depends on the MIDP implementation. The following are the list of methods used in the Ticker.

```java
// Ticker's Constructor
public Ticker(String string)

// To set the Ticker
public void setTicker(Ticker ticker)

// To get the Ticker
public Ticker getTicker()

//To get the string associated with the Ticker
public String getString()

// To set the string associated with the Ticker
public setString(String s)

//To attach the Ticker to screen
setTicker(new Ticker("Welcome to CTIS 487 class"));
```

**Form**

A form is a collection of instances of the Item interface. The TextBox class is a standalone UI element, while the TextField is an Item instance. Essentially, a textbox can be shown on a device screen without the need for a form, but a text field requires a form.

An item is added to a form using the `append(Item item)` method, which simply tacks the added item to the bottom of the form and assigns it an index that
represents its position in the form. The first added item is at index 0, the second at
index 1, and so on. You can also use the `insert(int index, Item newItem)` method to
insert an item at a particular position or use `set(int index, Item newItem)` to replace
an item at a particular position specified by the index.

There are eight Item types that can be added to a form.

1. **StringItem**: A label that cannot be modified by the user. This item may
   contain a title and text, both of which may be null to allow it to act as a
   placeholder. The Form class provides a shortcut for adding a StringItem,
   without a title: `append(String text)`
2. **DateField**: Allows the user to enter date/time in one of three formats: DATE,
   TIME, or DATE_TIME.
3. **TextField**: Same as a TextBox.
4. **ChoiceGroup**: Same as a List.
5. **Spacer**: Used for positioning UI elements by putting some space between
   them. This element is an invisible UI element and can be set to a particular
   size.
6. **Gauge**: A gauge is used to simulate a progress bar. However, this progress
   bar look can also be used in an interactive mode by the user. For example, if
   you wanted to show the user a volume control, a gauge would be used to
   show an interactive knob.
7. **ImageItem**: An item that holds an image. Like the StringItem, the Form class
   provides a shortcut method for adding an image: `append(Image image)`.
   More about images in a later section.
8. **CustomItem**: CustomItem is an abstract class that allows the creation of
   subclasses that have their own appearances, their own interactivity, and their
   own notification mechanisms. If you require a UI element that is different
   from the supplied elements, you can subclass CustomItem to create it for
   addition to a form.

Its constructors are:

```java
public Form(String str);
public Form(String str, Item[] items);
```

It will arrange its components as a list. Like the choices within a list, items within a
form can be edited using insert, append, and delete. The **Item** abstract class acts as
the base class for all components that can be placed on a form. All Item objects
have a label:
public String getLabel()

public void setLabel(String s)

**StringItem**

It is an item that can contain a string. A StringItem is display-only; the user cannot edit the contents. Both the label and the textual content of a StringItem may be modified by the application. The visual representation of the label may differ from that of the textual contents. Its constructors are:

```java
public StringItem(String label, String text)
// Either label or text may be present or null.

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

**TextField**

TextField object is an editable text component. A TextField has a capacity, which is the number of characters that can be stored in the object. There must be a boundary on the maximum size, which we can get using `getMaxSize()`. TextField is used when the MIDlet requires input from the user. A TextField has the constructor:

```java
public TextField(String label, String text, int maxSize, int constraints)
```

The constraints field is used to limit the user input like `TextField.ANY`, `TextField.EMAILADDR`, `TextField.NUMBER`, `TextField.PASSWD`, `TextField.PHONENUMBER`, and `TextField.URL`. 
**TextBox**

Text is entered by the user using a textbox. Like the other UI elements, a textbox has simple features that can be set based on your requirements. You can restrict the maximum number of characters that a user is allowed to enter into a textbox, but you need to be aware of the fact that the implementation of this value depends upon the device that you are running it on. For example, suppose that you request that a textbox is allowed a maximum of 50 characters, by using `setMaxSize(50)`, but the device can only allocate a maximum of 32 characters. Then, the user of your MIDlet will only be able to enter 32 characters.

You can also constrain the text that is accepted by the textbox, as well as modify its display using bitwise flags defined in the TextField class. For example, to only accept email addresses in a textbox, you will need to set the `TextField.EMAILADDR` flag using the method `setConstraints()`. To make this field uneditable, you will need to combine it with the `TextField.UNEDITABLE` flag. This is done by doing a bitwise OR operation between these two flags: `setConstraints(TextField.EMAILADDR | TextField.UNEDITABLE)`.

There are six constraint settings for restricting content: ANY, EMAILADDR, NUMERIC, PHONENUMBER, URL, and DECIMAL. ANY allows all kinds of text to be entered, while the rest constrains the input data according to their names. Similarly, there are six constraint settings that affect the display. These are: PASSWORD, UNEDITABLE, SENSITIVE, NON_PREDICTIVE, INITIAL_CAPS_WORD, and INITIAL_CAPS_SENTENCE. Not all of these settings may be functional in all devices.

To set the contents of a textbox, you can use a couple of methods. Use `setString(String text)` to set the contents with a String value. Use `insert(String text, int position)` to position text where you want it to go.

**List**

A list contains one or more choices (elements), which must have a text part, an optional image part, and an optional font for the text part. The List element implements the Choice interface, which defines the basic operations of this element. The list must itself have a title, and must define a policy for the selection of its elements. This policy dictates whether only one element can be selected
(Choice.EXCLUSIVE), multiple elements can be selected (Choice.MULTIPLE), or the currently highlighted element is selected (Choice.IMPLICIT).

You can create a list in one of two ways.

- Create a list that contains no elements, and then append or insert individual elements.
- Create the elements beforehand and then create a list with these elements.

List elements can be modified after the list has been created. You can modify individual elements by changing their text, text font, or image part, using the list index (starting at 0). You can delete elements using `delete(int index)` or `deleteAll()`. Any changes take effect immediately, even if the list is the current UI element. Its constructors are:

```
public List(String name, int listType)
public List(String name, int listType, String[] stringElements, Image[] imageElements)
```

**DateField**

A DateField object is a component for representing date and time information. It can be configured to accept date or time information. It has the following constructors:

```
public DateField(String label, int mode)
public DateField(String label, int mode, TimeZone timeZone)
```

The first one is used to create a DateField object with the specified label and mode. It has some static fields: `DateField.DATE`, `DateField.TIME`, or `DateField.DATE_TIME`. The `DateField.DATE` input mode used to set date information, `DateField.TIME` for clock time information, and `DateField.DATE_TIME` for both. This one has the following methods:

```
public Date getDate( )
public int getInputMode( )
public void setDate(Date date)
public void setInputMode(int mode)
```
**ChoiceGroup**

A ChoiceGroup object represents a list of selectable choices to be placed on a Form object. It implements the Choice interface. In this object single choice can be made, or it may allow multiple choices. It has two constructors:

```java
public ChoiceGroup(String label, int choiceType)
public ChoiceGroup(String label, int choiceType, String[] stringElements, Image[] imageElements)
```

The first constructor is used to create an empty choice group, specifying its label and type. Here two choices are available: EXCLUSIVE and MULTIPLE. The second constructor can be used to create a new choice group, specifying its title and type. We can insert, append, or replace choices in this. Each choice has an integer index that represents its position in the list. The first choice starts at 0 and extends to the current size of the list minus one. It has the following methods.

```java
public int append(String stringElement, Image imageElement)
public void insert(int index, String stringElement, Image imageElement)
public void set(int index, String stringElement, Image imageElement)
```

**Images, Tickers, and Gauges**

Using images, tickers, and gauges as UI elements in MIDlets is quite straightforward. A gauge is an item that can only be displayed on a form to indicate progress or to control a MIDlet feature (like volume). A ticker, on the other hand, can be attached to any UI element that extends the Displayable abstract class, and results in a running piece of text that is displayed across the screen whenever the element that is attached to it is shown on the screen. Finally, an image, can be used with various UI elements, including a form, as we saw in the last section.

Since the ticker can be used with all displayable elements, it provides a handy way to display information about the current element on the screen. The Displayable class provides the method `setTicker(Ticker ticker)`, and the ticker can itself be created using its constructor `Ticker(String msg)`, with the message that you want
the ticker to display. By using `setString(String MSG)`, you can change this message, and this change is effected immediately.

**Image and ImageItem**

An ImageItem object is an image component that contains a reference to an Image object. Images can either be immutable or mutable. Immutable images are generally created by loading image data. They cannot be modified after creation. Mutable images are created in off-screen memory and can be modified. A mutable image can be created using static `createImage()` methods of the Image class.

```java
public static Image createImage(int width, int height)
// Three static createImage( ) methods used to create immutable images

public static Image createImage(Image image)
public static Image createImage(String name)
public static Image createImage(byte[] imageData, int imageOffset, int imageLength)
```

**Alert**

Alert is a screen which contains text or image. It is used to show the errors and exceptions. A modal alert is a alert which need confirmation from user to close it. But the timed alert doesn’t need the confirmation from user. It shows the error for certain amount of time and terminates. Here is the constructor for Alert:

```java
public Alert(String name)
public Alert(String name, String message, Image msgImage, AlertType errorType)
```

We can use a timeout value for displaying the Alert message, meaning for how much time the Alert message will be displayed. Initially we can set the timeout and later we can change it. If we want to know the current timeout value we can get it. Similarly we can set and get the Image and String values associated with the Alert. Here are the some of the methods for these purposes.

```java
public int getDefaultTimeout( )
public int setTimeout( int t)
public Image getImage( )
public String getString( )
```
The `AlertType` class has provided with five different types of alerts: AlertType.ERROR, AlertType.ALARM, AlertType.CONFIRMATION, AlertType.WARNING, and AlertType.INFO. Here are some methods to set and get the Alert Types.