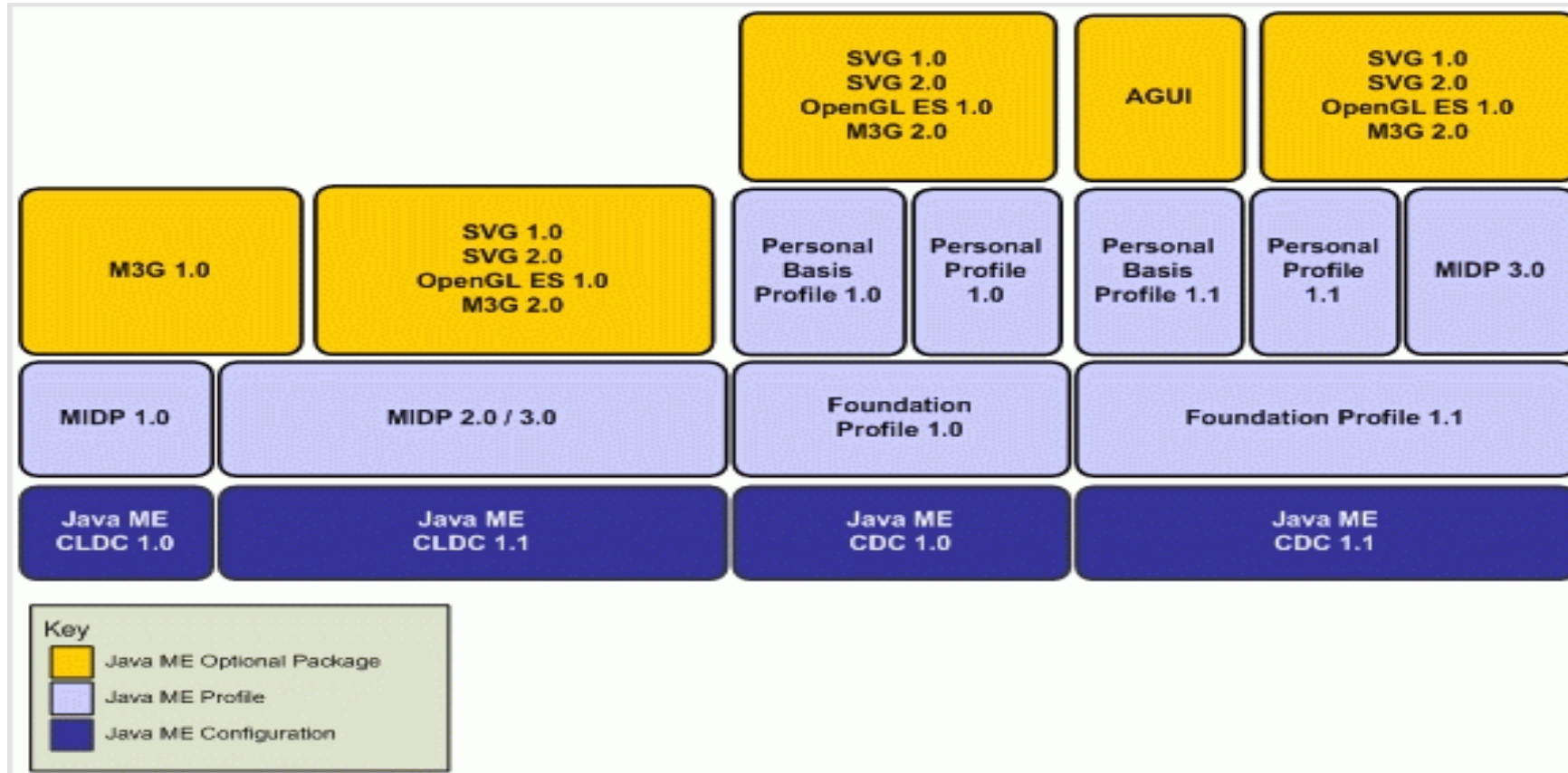


Appendix

- UI Design
 - ★MIDP 2.0 Menus and commands:
 - High-level and low-level
- Persistence Storage

➤ UI Design (Graphical APIs)



Further Reading –

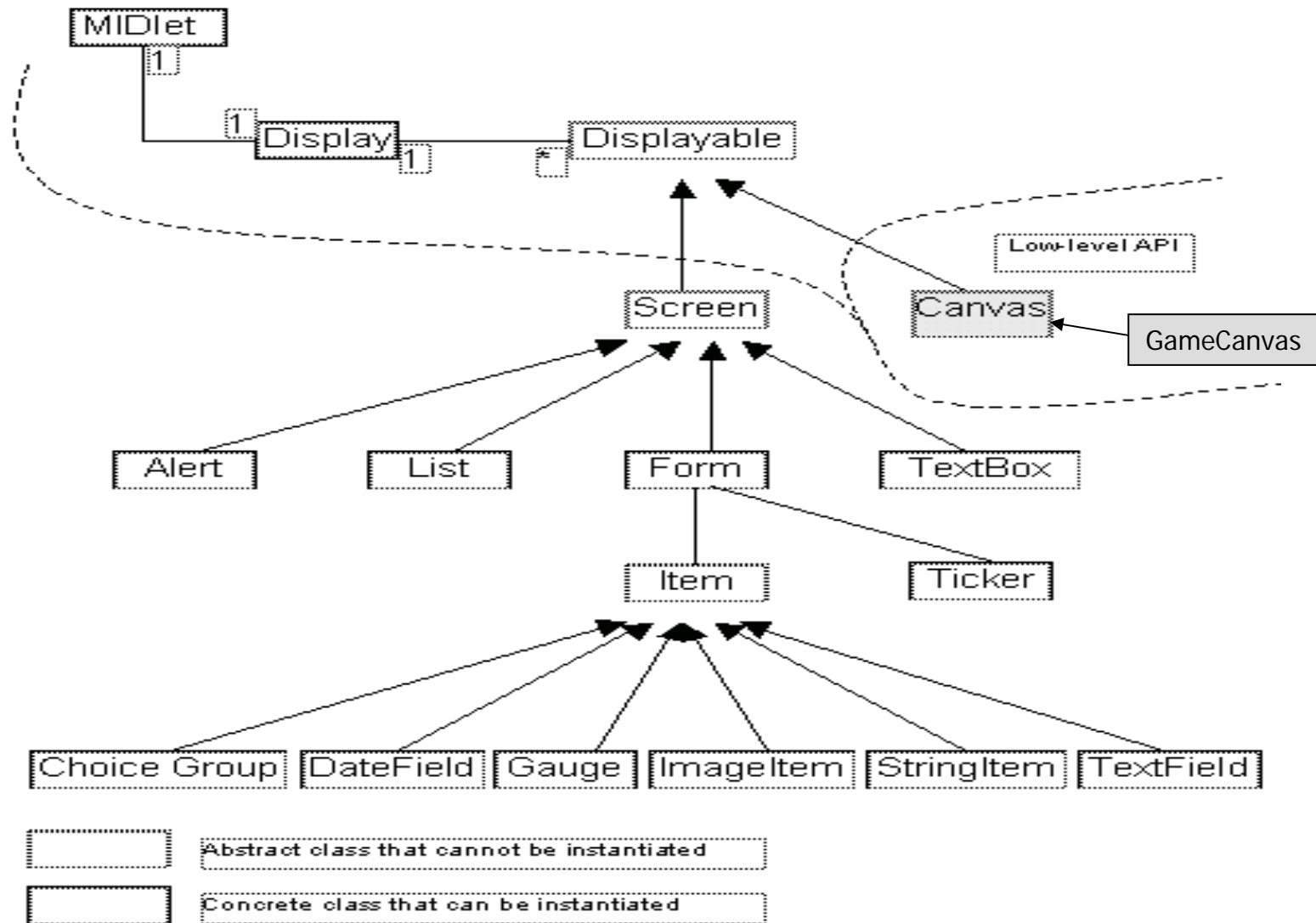
<http://developers.sun.com/mobility/midp/articles/guiapis/>



UI, Storage, AV

- CLDC 1.1/MIDP 2.0 LCDUI
 - ★ High-level UI objects
 - ★ Low level UI objects
- Persistence Storage
- Audio and Video

High-level User Interface APIs



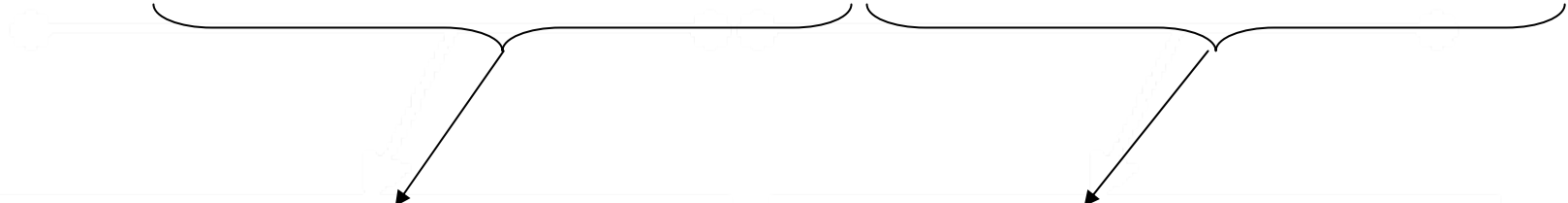
Display and Displayable classes

- The *Display* class is the **manager** of the actual device's display-screen and input.
- To create or get an instance of this *Display* class, the ***Display.getDisplay*** method is used. The **MIDlet** it self is passed as an argument to the ***Display.getDisplay*** method.
- The **displayable object** of a MIDlet is shown on the device's screen through a call to ***Display.setCurrent***.



Display and Displayable classes

```
Display.getDisplay (thisMIDlet).setCurrent(HelloForm);
```



Gets an instance of the Display object, which represents the actual display. The MIDlet itself is given as an argument for the *getDisplay* method.

Sets *HelloForm* to be current display content of the *Display* object.



MIDP Screen - Alert

- A standardized dialog displays a label, text and an optional image for a short time.
- Alert Types : **ALARM, INFORMATION, ERROR, WARNING and CONFIRMATION.**

```
Alert alertMsg = new Alert("Alert Message", "Are you  
sure?", null, AlertType.CONFIRMATION);  
alertMsg.setTimeout(alertMsg.FOREVER);  
Display.getDisplay(this).setCurrent(alertMsg);
```

- *Implicit DISMISS_COMMAND. If the application adds any other Commands to the Alert, DISMISS_COMMAND is implicitly removed.*
- *setIndicator(javax.microedition.lcdui.Gauge)*
- *AlertType.playSound()*

alertMsg.setTimeout(1000); - Display Alert for 1000ms, then Form
Display.getDisplay(this).setCurrent(alertMessage, HelloForm);



MIDP Screen - Form

- This is the only container class in MIDP on which you can place *Items* given below.
- Items can be added to the Form using *append()* method.

Item	Description
ImageItem	Used to place an image on a Form
StringItem	Used to place a string on a Form
TextField	Used to create a text input field on a Form
ChoiceGroup	Used to create a choice group input field on a Form
DateField	Used to create a date field on a Form
Gauge	Used to create a bar graph for given integer values

- Example : [Form with imageItem and stringItem](#) (Listing 3.3)



MIDP Screen - List

- Displays a series of items for selection.
- List types : IMPLICIT, EXCLUSIVE or MULTIPLE.
 - ★ IMPLICIT: The application gets immediate notification when an item is selected.
 - ★ EXCLUSIVE: This option allows you to create an option list where the users can select only one item from the list.
 - ★ MULTIPLE: This option allows you to create check boxes where the users can select any number of items. The method *getSelectedFlags(Boolean[] index)* will return an array of Boolean representing the selection of items by the user.



MIDP Screen : TextBox

- The *TextBox* allows you to get multi-line text input from the user.
- You can control the input using constraint constants.
- A *TextBox* is constructed with a label, default text, a maximum text size, and constraints to control the type of input.

```
TextBox id = new TextBox("Enter PIN num", null, 8,  
                          TextField.NUMERIC);  
Display.getDisplay(this).setCurrent(id);
```



MIDP Screen : TextBox

➤ TextBox constraint constants

- ★ *public static final int ANY = 0;*
- ★ *public static final int EMAILADDR = 1;*
- ★ *public static final int NUMERIC = 2;*
- ★ *public static final int PHONENUMBER = 3;*
- ★ *public static final int URL = 4;*
- ★ *public static final int PASSWORD = 65536;*
- ★ *public static final int CONSTRAINT_MASK = 65535;*



MIDP Screen : TextBox

- Textbox Modifiers (Can be combined with constraint constant using **&**)
 - ★ PASSWORD
 - ★ UNEDITABLE
 - ★ SENSITIVE
 - ★ NON_PREDICTIVE
 - ★ INITIAL_CAPS_WORD
 - ★ INITIAL_CAPS_SENTENCE

- TextBox Modes- Internationalization (UniCode Blocks)
 - ★ UCB_BASIC_LATIN
 - ★ UCB_GREEK
 - ★ UCB_HEBREW
 - ★ UCB_ARABIC
 - ★ UCB_DEVANAGARI

– **`textBox1.setInitialInputMode(UCB_BASIC_LATIN)`**



Item Class Object - StringItem

FYORP

- *StringItem* is used to create a simple read-only text with a label to the user.

```
StringItem UIDlabel = new StringItem("User ID:","Anand");  
demoForm.append(UIDlabel);
```



Item Class Object - TextField

FYORP

- The **TextField** MIDP component allows constrained user input and is similar in usage to the `TextBox` object.
- The constructor takes same four parameters as *TextBox*.

```
TextField textPwd = new TextField("Enter Password", "",  
                                15,TextField.ANY);  
demoForm.append(textPwd);
```



Item Class Object - ImageItem

FYORP

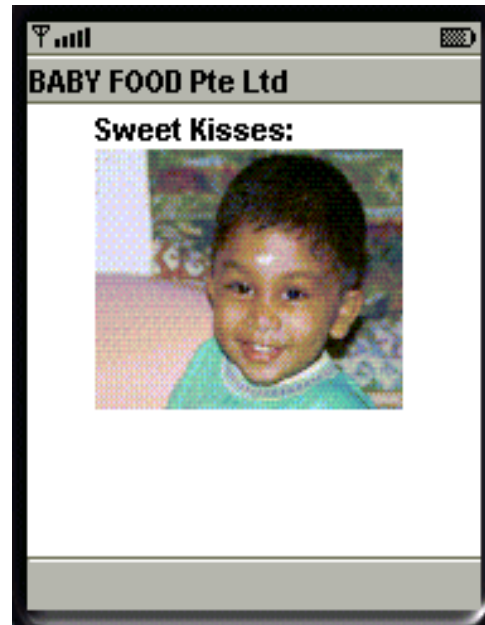
- *ImageItem* is a non-interactive item that is used to display images.
- In addition to the label, the *ImageItem* constructor takes an Image object, a layout parameter, and an alternative text string, which will be displayed when the device is not able to display the image for some reason.
- Image objects can be constructed as **mutable or immutable**. Mutable images can be changed dynamically whereas immutable or static images are loaded from some external source such as a **Portable Network Graphic (*.png)** file.



Item Class Object - ImageItem

FYORP

- Example : [Image Item](#)
- Output :



Item Class Object - ImageItem

FYORP

➤ ImageItem layout constants

Constant Name	Value	Meaning
LAYOUT_DEFAULT	0	The image is aligned according to the default formatting defined in the device.
LAYOUT_LEFT	1	Image is aligned to left.
LAYOUT_RIGHT	2	Image is aligned to right.
LAYOUT_CENTER	3	Image is horizontally aligned to center.
LAYOUT_NEWLINE_BEFORE	256	New line will be started before the image.
LAYOUT_NEWLINE_AFTER	512	New line will be started after the image.



Item Class Object - DateField

FYORP

- *DateField* allows the user to view/edit the date and time information. The presentation of date and time depends on input mode, which takes one of the three possible values.

Constant name	Value	Meaning
DATE	1	Allows the user to enter date only in the <i>DateField</i>
TIME	2	Allows the user to enter time only in the <i>DateField</i>
DATE_TIME	3	Allows the user to enter date and time information in the <i>DateField</i>



Item Class Object - DateField

FYORP

```
DateField appointTime = new DateField("Exam Date:",  
DateField.DATE_TIME);
```



Item Class Object - ChoiceGroup

FYORP

- ChoiceGroup item is similar to the List screen used to create a list of items on a Form for selection by the user.
- These elements consist of simple Strings, but can display an optional image per element as well. The *ChoiceGroup* can be of either two types: EXCLUSIVE or MULTIPLE.

Constant	Meaning
EXCLUSIVE	To create option list where the users can select only one item from the list.
MULTIPLE	To create check list where the users can select multiple items
IMPLICIT	Valid for List screen only. It lets the List to send the state changes immediately.



Item Class Object - ChoiceGroup

FYORP

- A simple *ChoiceGroup* constructor takes a label and choice type constant parameters to create an empty *ChoiceGroup*. Additionally you can provide an array of Strings and Images to be used as its initial contents.

```
private ChoiceGroup ch = new  
ChoiceGroup("Select Service:", ChoiceGroup.EXCLUSIVE,  
arr, null);
```

- Elements can be added dynamically by using *append()* method. The Append method takes two parameters: a text and an image.
- Example : [ChoiceGroup](#)



Item Class Object - Guage

FYORP

➤ Output

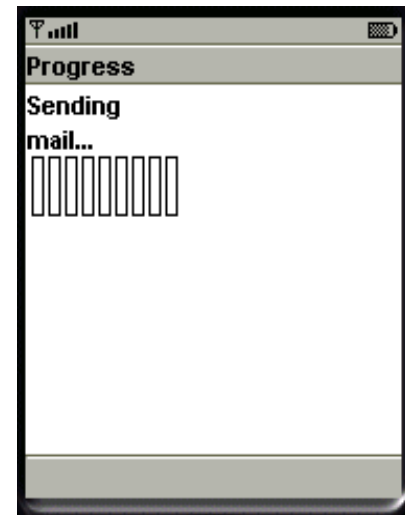


Item Class Object - Gauge

FYORP

- Gauge provides horizontal bar for a given integer value. Gauge can be used to show progression of a task visually or as a sliding 'track bar' that allows user interaction.

```
public void startApp() {  
    Form progress = new Form("Progress");  
    Gauge G1 = new Gauge("Sending mail...", false, 100, 1);  
    progress.append(G1);  
    Display.getDisplay(this).setCurrent(progress);  
}
```



Item Class Object - Ticker

FYORP

- Tickers are used to scroll the text across the screen continuously. Tickers can be associated with several screens at a time. You can use the Screen object's *setTicker()* method to associate the ticker with any Screen (Form, Canvas, TextBox, Alert...).

```
Form mainForm = new Form("Anuflora International");  
Ticker grt = new Ticker("Welcome to Anuflora  
International");  
mainForm.setTicker(grt);
```



Receiving Changes form Interactive UI Items

- To receive the users response to the high-level user interface objects within a *Form* screen, MIDP defines *ItemStateListener* interface.
- *itemStateChanged()* is the only method in the *ItemStateListener* interface.
- *itemStateChanged(Item item)* is called when the internal state of an Item has been changed by the user.



Receiving Changes form Interactive UI Items

- User actions that causes state changes
 - ★ Changes the set of selected values in a *ChoiceGroup*
 - ★ Adjusts the value of an interactive *Gauge*
 - ★ Enters or modifies the value in a *TextField*
 - ★ Enters a new date or time in a *DateField*

- Coding steps
 - ★ Add the ***ItemStateListener*** interface in the class declaration;
 - ★ **Implement the *itemStateChanged()*** method in your class;
 - ★ **Register the *ItemStateListener*** at the *Form* from which you need to receive the events.

- Example : [ItemStateListener](#) (3.10)



Commands & CommandListener

- Commands allow the user to initiate some tasks.
- Creating and using Command objects.

Constructing the *Command*. (Line number 4 in listing 3.11c)

```
Command qt = new Command("Quit", Command.EXIT, 1);  
Command bk = new Command("Back", Command.BACK, 1);
```

Adding the command to *Displayable*. (Line numbers 23,24 in listing 3.11c)

```
this.addCommand(qt);  
this.addCommand(bk);
```

Defining your class with *CommandListener* interface (Line number 3 in listing 3.11c)

```
public class subForm extends Form implements  
CommandListener
```



Commands & CommandListener

- Creating and using Command objects.

Implementing the *commandAction()* method. (Line numbers 27 to 34 in listing 3.11c)

```
public void commandAction(Command command,
Displayable displayable) {
    if (command == qt) {
        Choices.quitApp();
    }
    else if (command == bk) {
        Choices.display.setCurrent(lastScreen);
    }
}
```



Commands & CommandListener

- Creating and using Command objects.

Registering the *CommandListener* class to a *Displayable*. . (Line number 21 in listing 3.11c)

```
setCommandListener(this);
```



Handling Timer Events

- *Timer* can be used to trigger tasks at a given interval for one or more repetitions.
- *Timer* object is used to create *Timer* and the *TimerTask* object is used to define a task to be done when the timer expires.

```
Timer T = new Timer();  
TimerTask task = new progerssTask();
```



Handling Timer Events

- The *TimerTask* object defines a abstract method *run()* that should be implemented by the application.
- The *run()* method will be executed when ever the *Timer* triggers the *TimerTask*.

```
class progressTask extends TimerTask {  
    public void run() {  
  
        /* Add codes that should be executed when the Timer triggers this  
           Task.*/  
        -----  
        -----  
        -----  
        T.cancel(); // cancel the timer to stop triggering this task  
    }  
}
```

T.schedule(task, 3000);

T.schedule(task, 3000, 1000);



Third-Party UI objects for MIDP

➤ www.j2mepolish.org



UI, Storage, AV

- CLDC 1.1/MIDP 2.0 LCDUI
 - ★ High-level UI objects
 - ★ Low level UI objects
- Persistence Storage

Canvas Screen

- Canvas
 - ★ Pixel level access to screen
 - ★ Keyboard and pointer events
 - ★ Abstract method `paint()`
 - ★ `repaint()` method
 - ★ Example : [Simple Drawing – Listing 4.1b](#)
- Graphics class
 - ★ Graphics object provides simple 2-D geometric rendering capability. Drawing primitives are provided for text, images, lines, rectangles, and arcs. Rectangles and arcs may also be filled with a solid color.
 - ★ ***Double buffering***, also known as screen swapping, is a common graphics technique to reduce flicker in animations.



Canvas Screen

FYORP

- Output for listing 4.1b

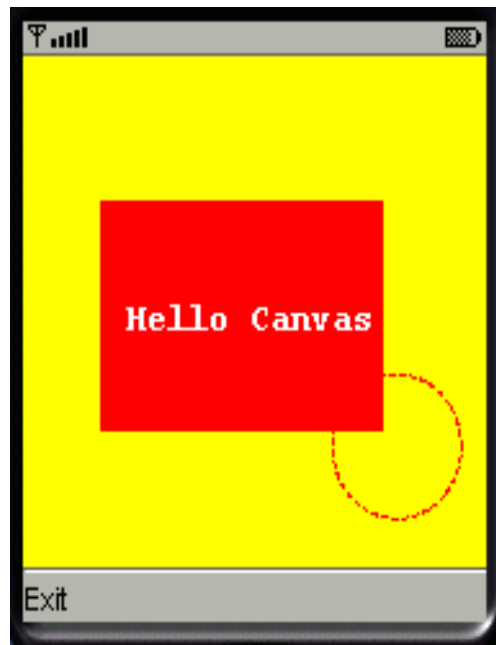


Figure 4.2 Simple Drawing



Drawing Methods

FYORP

<i>Method</i>	<i>Purpose</i>
<i>drawString(String text, int x, int y, int anchor)</i>	Draws the specified String at the given position using the current font and color.
<i>drawImage(Image image, int x, int y, int anchor)</i>	Draws the specified image at the given position.
<i>drawLine(int x1, int y1, int x2, int y2)</i>	Draws a line between the coordinates (x1,y1) and (x2,y2) using the current color and stroke style.
<i>drawRect(int x, int y, int width, int height)</i>	Draws the outline of the specified rectangle using the current color and stroke style
<i>fillRect(int x, int y, int width, int height)</i>	Draws a filled rectangle with the current color.



Drawing Methods

FYORP

<i>Method</i>	<i>Purpose</i>
<i>setColor(int red, int green, int blue)</i>	Sets the current color to the specified RGB values
<i>setFont(Font font)</i>	Sets the font for all subsequent text rendering operations
<i>setGrayScale(int value)</i>	Sets the current grayscale to be used for all subsequent rendering operations
<i>setStrokeStyle(int style)</i>	Sets the stroke style used for drawing lines, arcs, rectangles, and rounded rectangles

[Refer table 4.2 for other methods](#)



Coordinate System

- The coordinate system represents locations between pixels, not the pixels themselves.

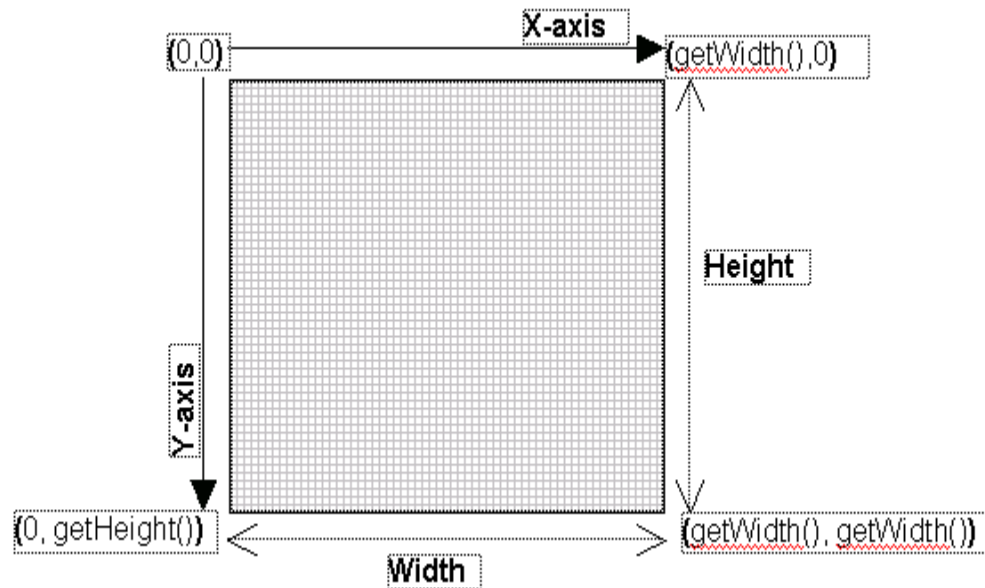


Figure 4.3 Coordinate System



Coordinate System

- The origin of the coordinate system can be changed using the `translate (int x, int y)` method. It will add the coordinates `(x,y)` with all the subsequent drawing operations automatically. For example,

`g.translate(getWidth()/2,getHeight()/2)`

- will cause the center point of the screen to be the origin for the subsequent drawings.



Clipping

- A clip is a rectangle region in the destination of the *Graphics* object that responds to the subsequent drawing operations. There can be one clip per *Graphics* object.

Method	Purpose
setClip (int x, int y, int width, int height)	Sets a new rectangle clip region specified by the coordinates. Subsequent drawings will be effective only inside this region. Any drawing outside this region will be ignored.
getClipX() , getClipY() , getClipHeight() , getClipWidth()	Returns the X offset, Y offset, height and width of the current clipping area.

Table 4.3 Methods of Graphics class for clipping



Clipping

➤ Listing 4.2 Clipping Demo

```
28. /** Required paint implementation */
29. protected void paint(Graphics g) {
30.     g.setColor(255,255,255);
31.     g.fillRect(0,0,getWidth(),
32.         getHeight());
33.     g.setColor(0,0,0);
34.     g.drawArc(30,30,130,100,0,360);
35.     g.setStrokeStyle(g.DOTTED);
36.     g.drawRect(40,40,110,80);
37.     g.setClip(40,40,110,80);
38.     g.setColor(0,255,0);
39.     g.fillArc(30,30,130,100,0,360);
40. }
```

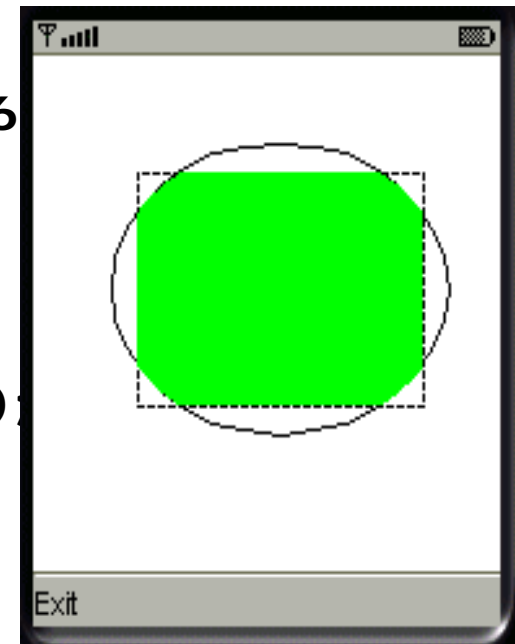


Figure 4.4



Drawing Texts

- The method *drawString()* is used to draw text on the screen.
- *setFont()* sets the font for subsequent text rendering to the Font passed as parameter to the *setFont()* method.
- Font class
 - ★ To create a Font object, the *lcdui* defines a **Font class** with the *getFont()* method which takes three parameters: Size, Style and Face.
 - ★ *getFont(int face, int style, int size)*

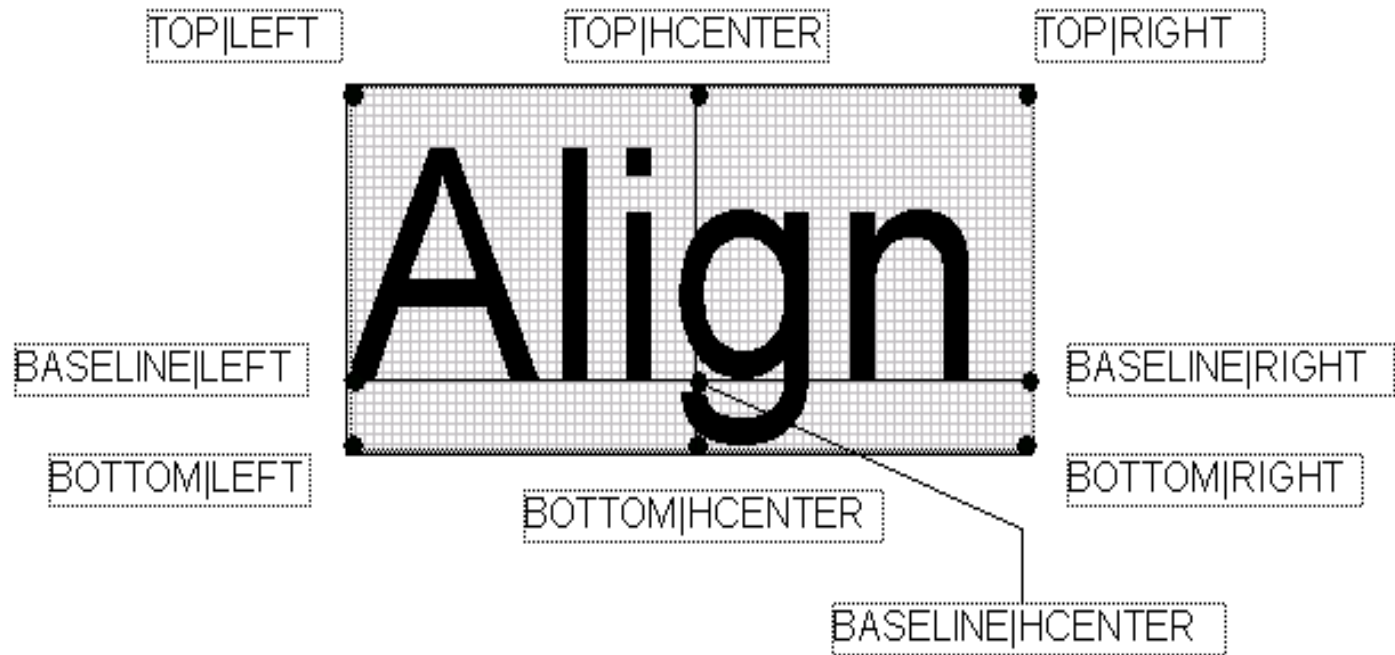
Parameter	Constants
Face	FACE_SYSTEM, FACE_MONOSPACE, FACE_PROPORTIONAL
Style	STYLE_PLAIN, STYLE_ITALIC, STYLE_BOLD, STYLE_UNDERLINED
Size	SIZE_SMALL, SIZE_MEDIUM, SIZE_LARGE

Table 4.5 getFont() parameters



Drawing Texts

- Figure 4.5 Anchor points



Drawing Images

- drawImage() – displays mutable images.
- Creating mutable image from immutable images

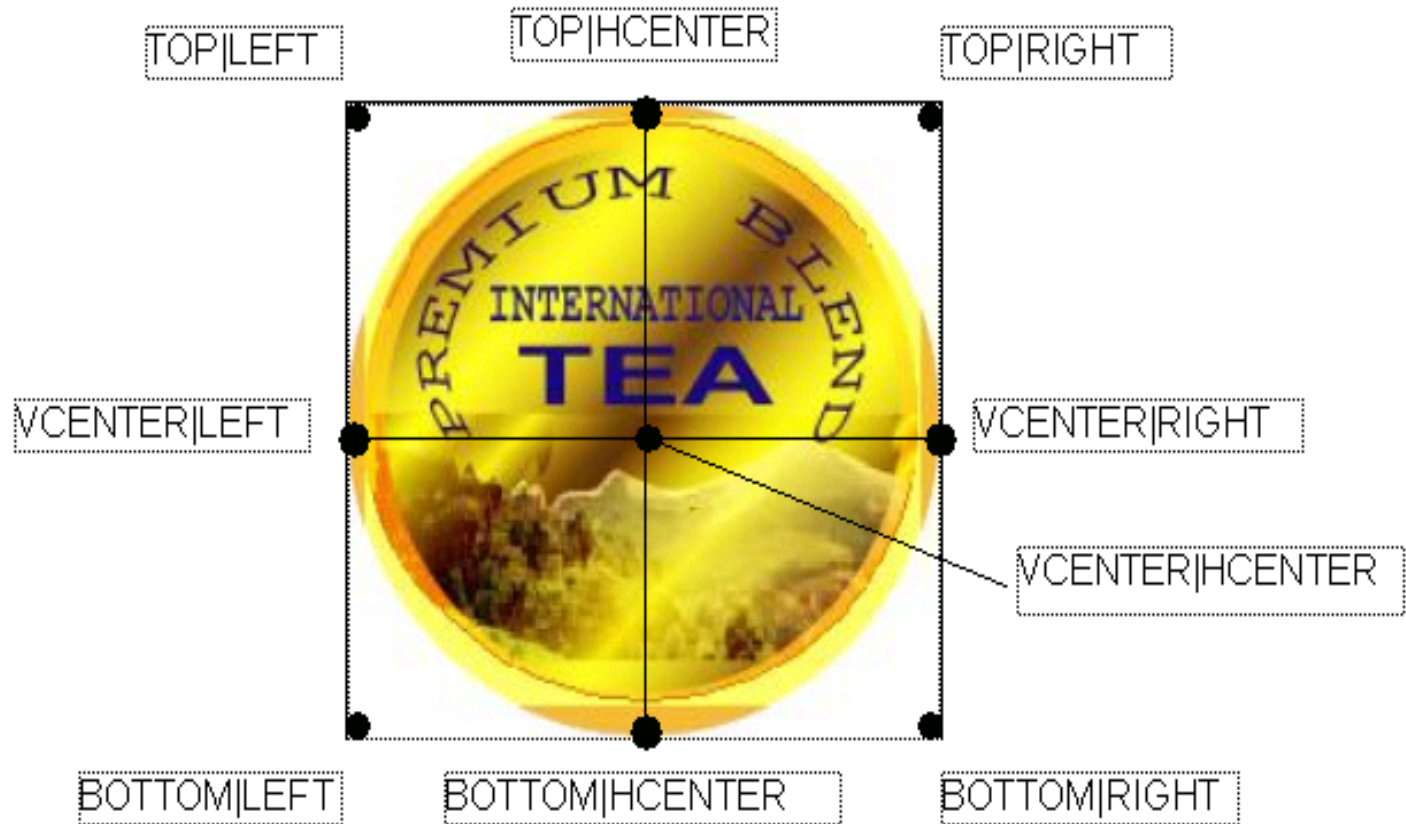
```
Image logo;  
logo =  
Image.createImage("/graphicsDemo/logo.gif");  
Image mutableLogo =  
Image.createImage(logo.getWidth(),  
logo.getHeight());  
Graphics g = mutableLogo.getGraphics();  
g.drawImage(logo, 0, 0, TOP|LEFT);
```

- Example: Refer Listing 4.3 and Figure 4.6 in Text book. Page 94.



Drawing Images

- Anchor points of an Image (Figure 4.7)



Event Handling : Keyboard

- Key events return a *Key Codes*, which are directly bounded to the physical keys. The mapping from key to key code is device dependent.
- MIDP defines the following key codes, which represents the keys on a ITU-T standard keypad: KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_STAR, KEY_POUND.
- *Canvas* defines another method ***getKeyName()*** which returns the name of the key for the given key code.
- *Canvas* provides the following key event callback methods: ***keyPressed()***, ***keyReleased()***, and ***keyRepeated()***.



Event Handling : Keyboard

- Applications, which need only game related events and arrow key events, can use the game actions rather than key codes to maximize portability.
- MIDP has defined the following game actions: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, and GAME_D. The game actions are mapped to one or more keys.
- Portable applications can call the ***getGameAction()*** method to get the game action represented by the given key code.
- Example: [Listing 4.4 events and game actions](#)



Event Handling : Pointing Device

- Pointing devices – mouse, touch screen, stylus and trackball, etc.
- Canvas class provides three methods to handle pointer events: ***pointerPressed()***, ***pointerDragged()***, and ***pointerReleased()***.
- Example : [Listing 4.5 Pointer events demo.](#)



UI, Storage, AV

- CLDC 1.1/MIDP 2.0 LCDUI
 - ★ High-level UI objects
 - ★ Low level UI objects
- Persistence Storage

RecordStore & Records

- A record store is a **collection of records** that will remain persistent in the local storage of the device.
- A record store **can be shared** across multiple MIDlets in a same MIDlet suite and must have a unique name in the MIDlet suite.
- The record store is **time stamped** with the last modification date/time. The record store also maintains a version. These data will be useful for the synchronization of engines and applications.
- Records are **arrays of bytes of variable length**.



Creating a Record Store

- To create a RecordStore use the *openRecordStore()* method that takes a record store name and a Boolean. If the Boolean is true, then a new RecordStore will be created if the RecordStore is not available.
- The method *addRecord()* is used to add a record to the RecordStore. The *addRecord()* method takes a byte array, starting index, and number of bytes to copy into the record. The method returns a unique ID for the record, which is the record ID.



Creating a RecordStore

➤ Example

```
1.  try {  
2.  RecordStore rs =RecordStore.openRecordStore(" PhoneBook", true);  
3.  rs.addRecord(" Mikhy=96709990" .getBytes(),0, 14);  
4.  rs.closeRecordStore();  
5.  } catch (RecordStoreException e) {  
6.  e.printStackTrace();  
7.  }  
8.  }
```



RecordStore Methods

➤ Static RecordStore ManipulationMethods

(Table 5.1)

➤ Records Manipulation Methods (Table 5.1)

➤ Record Store Metadata (Table 5.1)



RecordEnumeration

- RecordEnumeration logically maintains the sequence of record-Ids in a new record store. RecordEnumeration Interface is defined in **rms**. (**javax.microedition.rms**)
- Functions :
 - ★ Traverse through the records. (forward and backward.)
 - ★ Retrieve records by criteria. (search.)
 - ★ Sort records in an application defined order.



Simple RecordEnumeration

- Example: RecordEnumeration without Filtering and Sorting.

```
RecordStore rs = null;
```

```
RecordEnumeration enum = null;
```

```
String recordContents;
```

```
rs = RecordStore.openRecordStore("PhoneBook", true);
```

```
enum = rs.enumerateRecords(null, null, true);
```

```
while (enum.hasNextElement()) {
```

```
    recordContents = new String(enum.nextRecord());
```

```
}
```

```
rs.closeRecordStore();
```



RecordEnumeration With Filtering

- RecordEnumeration methods.

[\(Table 5.3\)](#)

- To **Filter records** based on a criteria
 - ★ Define a class that implements the *RecordFilter* Interface.
 - ★ Hook your *RecordFilter* to your enumerator by instantiating the *RecordFilter* class in the call to `enumerateRecords`.



RecordEnumeration With Filtering

➤ Example

★ Defining RecordFilter class

```
public class recordSelectFilter implements RecordFilter {  
    public boolean matches (byte[] data) {  
        return (data[0] == txtfldSelect.getString( ).getBytes( ) [0]);  
    }  
}
```

★ Hooking *RecordFilter* to your enumerator

```
enum = rs.enumerateRecords(new recordSelectFilter(), null, true);
```



Sorting Records

- Example:
 - ★ Defining a class that implements *RecordComparator* Interface

```
public class alphabeticalOrder implements RecordComparator {  
    public int compare(byte [] byArrRecord1, byte[] byArrRecord2) {  
        int compareResult = new String (byArrRecord1). compareTo (new String  
                                                                    (byArrRecord2));  
  
        int result = EQUIVALENT;  
        if (compareResult < 0) {  
            result = PRECEDES;  
        } else if (compareResult > 0) {  
            result = FOLLOWS;  
        }  
  
        return(result);  
    }  
}
```



Sorting Records

- Example:
 - ★ Hooking *RecordComparator* to your enumerator

```
enum = rs.enumerateRecords (null,  
    new alphabeticalOrder(), true);
```



Record Change Notification

- RMS defines the ***RecordListener* interface** for receiving Record Change events in an application from a record store.
- By creating a class that implements the methods of *RecordListener* interface, you can receive Record Change events.
- ***RecordListener* abstract methods** are described below:
 - *recordAdded*([RecordStore](#) recordStore, int recordId)
 - *recordChanged*([RecordStore](#) recordStore, int recordId)
 - *recordDeleted*([RecordStore](#) recordStore, int recordId)



Record Change Notification

- Example : [Listing 5.2](#)
- Add the recordListener to the RecordStore

```
pals.addRecordListener(new  
recordStoreListener());
```



Record Change Notification

- Figure 5.2: Output for [Listing 5.2](#)

