



Mobile Information Device Programming (9)

Lecturer: Alireza Mousavi
School of Engineering & Design
www.brunel.ac.uk/~emstaam



Choice and ChoiceGroup

- **Choice** interface consists of methods that have the capability to manipulate various types of predefined selections.
- Two classes in MIDP that implement the **Choice** interface

```
public class ChoiceGroup extends MIDlet implements Choice {
```

```
...
```

```
}
```

```
public class List extends MIDlet implements Choice {
```

```
...
```

```
}
```



ChoiceGroup

Multiple (Check Boxes)

- Exclusive (Radio Buttons)





Event Handling for ChoiceGroups

There are two ways to realise the status of selections within a **ChoiceGroup**:

1. *ItemStateListener*

2. *CommandListener*



ChoiceGroup

ItemStateListener

- If the status of ChoiceGroup has changed and the form registers an *ItemStateListener* the *ItemStateChanged()* method is called.
- Within this group you can find out which item has been selected with *getSelectedFlags()* or *getSelectedIndex()*



ChoiceGroup CommandListener

- If a Form contains a ChoiceGroup and there exists a *CommandListener* registered with the form, you can add a Command to prompt your program to query the selection status.
- The method for this is *commandAction ()*.



Exclusive ChoiceGroup

Exercise 9-1

- Create a ChoiceGroup using append () method
- Create an application where one item can be selected from a list of available items (exclusive)

- In the Constructor declare an exclusive ChoiceGroup, Example:

```
cgEmail = new ChoiceGroup("You Have Email Message ...", Choice.EXCLUSIVE);
```

- Choose a default ChoiceGroup, Example:

```
replyIndex = cgEmail.append("Reply", null);  
... cgEmail.setSelectedIndex(replyIndex, true);
```



Result





MultipleChoice Exercise 9-2

- Preferences

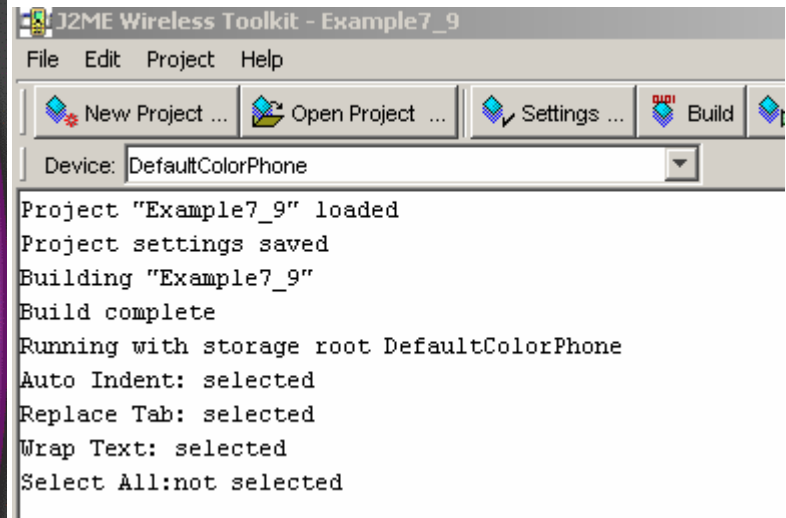




Image and ImageItem

- An **ImageItem** class allows to display images on your Form.
 - It has its own associated methods, API and layout constructs
- It is down to the device on how it would display or place the image



Image Class

- **Immutable:** Is a fixed image that cannot be changed during an application cycle time. You can normally place it in your /resources folder.
- **Mutable:** Is a flexible method of displaying images using the *paint ()* method. In this case you spare a chunk of memory that you will draw the image into



Image Layout Example

Creating an Image:

```
Image myImage = Image.createImage("/blue.png");
```

Specify layout

```
myForm.append(new ImageItem(null, myImage, ImageItem.LAYOUT_CENTER, null))
```

String label

String altText



Exercise 9.3

- Write an application with an associated immutable image.
- Create Image using Fireworks 30X30 pixel save in resources directory and provide the address for the MIDlet
- Add Strings
- Run application



Result

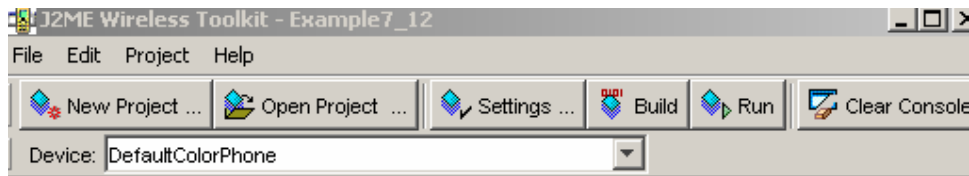
```
Command Prompt - midp -classpath . ImmutImage
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

h:\>d:
D:\>cd midlets\example7_10
D:\midlets\Example7_10>javac -bootclasspath c:\j2me\midp2.0fcs\classes ImmutImage.java
D:\midlets\Example7_10>preverify -classpath c:\j2me\midp2.0fcs\classes;. -d . ImmutImage
D:\midlets\Example7_10>midp -classpath . ImmutImage
Layout Directives are: 771
D:\midlets\Example7_10>midp -classpath . ImmutImage
Layout Directives are: 771
```





ChoiceGroup & Immutable Image 9-4 Assignment



```
Project "Example7_12" loaded
Project settings saved
Building "Example7_12"
Build complete
Running with storage root DefaultColorPhone
Red Circle: selected
Green Circle: not selected
Blue Circle: not selected
```





Mutable Image

- Self reading – see E9.5