# Programming Mobile Devices

## *High Level User Interface*

University of Innsbruck

WS 2009/2010


STI · INNSBRUCK

thomas.strang@sti2.at

# User Interface

- Graphical User Interface API (Input and Output) is defined as part of the Profile (e.g. MIDP), not the Configuration (e.g. CLDC)
  - On pure Configuration systems, only available:
    - `System.out` e.g. `System.out.println("hello")`
    - `System.err` (usually for simple Debugging)
- MIDP adds a GUI API called "LCDUI"
  - package `javax.microedition.lcdui`
  - Input *and* Output
  - closely related to UI MIDlet event model

# MIDlet UI Event Model

- `Display` class
  - exactly one instance of `Display` class per MIDlet (Singleton Pattern)
  - MIDlet can get a reference to that instance by calling the static
    `Display display = Display.getDisplay()`
- `Displayable` interface
  - any object to be visualized (shown) on the `Display` must implement the `Displayable` interface
  - uiObject sent to Display by
    `display.setCurrent(uiObject)`

# MIDlet UI Event Model

- `Command` class
  - high level event object
  - encapsulates semantic information of a single action
  - only info about command, action is defined in listener
- `CommandListener` interface
  - links commands to command handler implementation: `Displayable.setCommandListener(implementation)`
  - requires *callback* in implementing class: `public void commandAction(Command c, Displayable d)`
  - handler should return immediately!

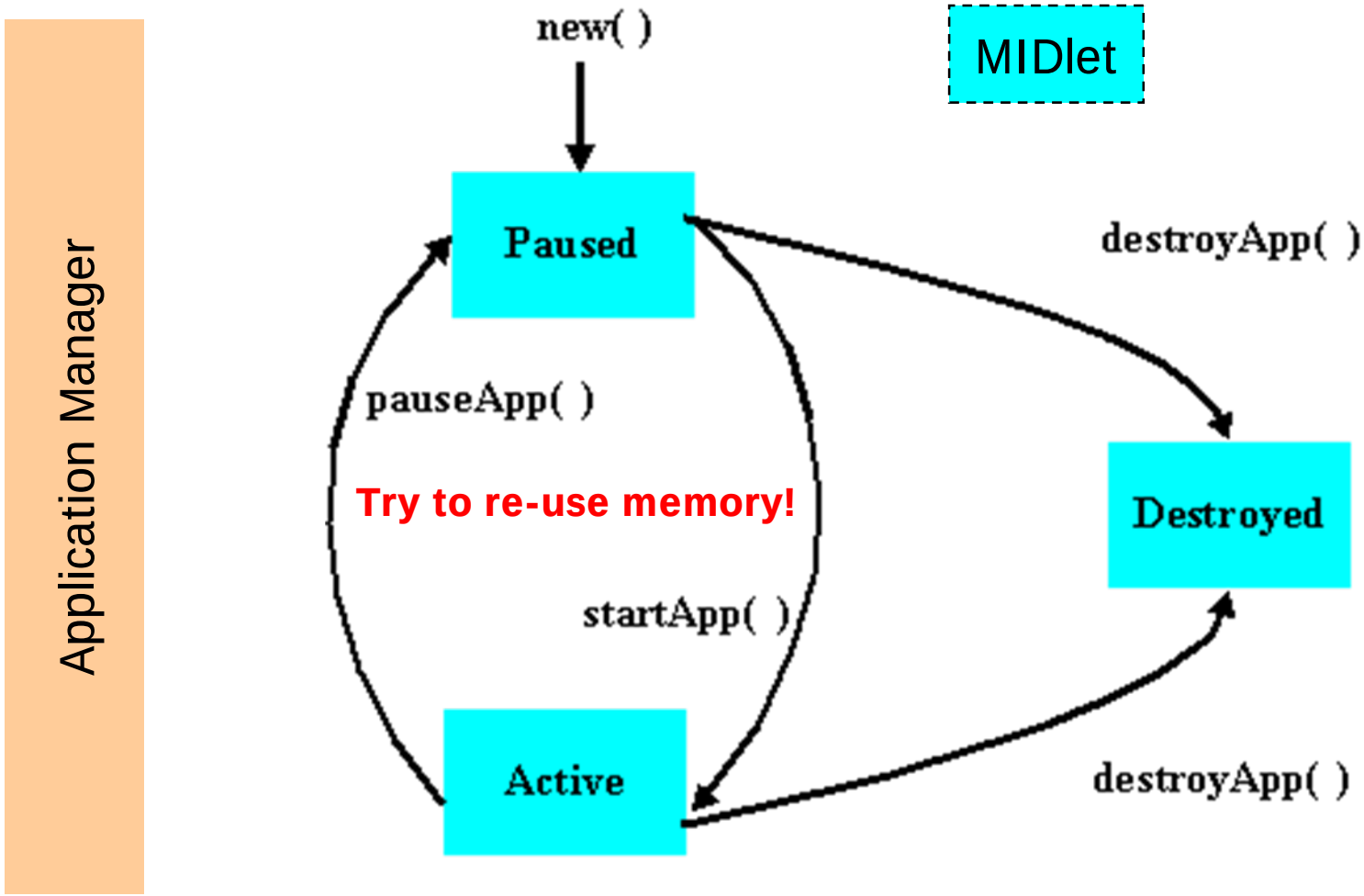
# Commands

- Label: string representation of action
- Type:
  - BACK
  - OK, CANCEL
  - EXIT
  - HELP
  - STOP
  - ITEM
  - (MENU)
  - SCREEN
  - Caution: None of the actions (beside MENU) are provided by the system automatically, the developer has to implement them!
- Priority
  - determines priority *within same type*
  - highest priority is 1, increasing with lower prios

# Recap: Hello World MIDlet

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener {
  private Command exitCommand;
  private TextBox tbox;

  public HelloWorld() {
    exitCommand = new Command("Exit", Command.EXIT, 1);
    tbox = new TextBox("MIDletTextBox", "Hello MIDP World!", 25, 0);
    tbox.addCommand(exitCommand);
    tbox.setCommandListener(this);
  }

  protected void startApp() {
    Display.getDisplay(this).setCurrent(tbox); }
  protected void pauseApp() {}
  protected void destroyApp(boolean bool) {}
  public void commandAction(Command cmd, Displayable disp) {
    if (cmd == exitCommand) { destroyApp(false); notifyDestroyed();
} } }
```

# Recap: MIDlet lifecycle



# Typical Steps

1. create screen object (alert, form etc.)
2. create event objects (commands)
3. add event objects to screen object (<screen>.addCommand(cmd))
4. implement event handler (commandAction(cmd,disp))
5. register callback (<screen>.setCommandListener)

# LCDUI Displayables

- LCDUI classes extending `Displayable`:
  - `Screen` ("high level GUI")
    - direct subclasses:
      - **TextBox**: A screen that allows the user to enter / edit text
      - **Alert**: A screen that allows data to the user and waits for a certain period of time before proceeding
      - **Form**: A screen that contains an arbitrary mixture of items (images, text, text fields, choice groups, etc.)
      - **List**: A screen containing a list of choices
    - common to all: `Title` and `Ticker` properties
      - have moved up to `Displayable` since MIDP 2.0
  - `Canvas` ("low level GUI")
    - later …

# TextBox Screen

- `TextBox` class is a `Screen` that allows the user to enter and edit text up to `maxSize` characters
- Input can be constraint towards:
  - Since MIDP 1.0: Email adresses, Numbers, Password, Phone numbers, URL
  - Since MIDP 2.0: Floating Point Numbers, Capital letters at the beginning of each word / sentence, Text prediction (e.g. T9), sensitive data (not to be stored anywhere), uneditable data

# Alert Screen

- An `Alert` is an ordinary `Screen` that can contain text (String) and/or images, and which handles events like other screens
- used to inform the user about errors and other exceptional conditions
- alert time can be set to infinity with `setTimeout(Alert.FOREVER)` in which case the Alert is considered to be *modal,* otherwise returned to displayable after timeout
- Alert can have a specific type assigned:
  Alarm, Confirmation, Error, Information, Warning

# Form Screen

- A `Form` is a `Screen` that contains an arbitrary mixture of items: images, read-only text fields, editable text fields, editable date fields, gauges, and choice groups
- system handles layout, traversal, and scrolling – user has almost no influence!
- items contained within a `Form` may be edited using append, delete, insert, and set methods

# Form- or Alert-Items

- `Item` is a superclass for components that can be added to a `Form` or an `Alert`
- `javax.microedition.lcdui.Item`, direct subclasses:
  - ChoiceGroup, DateField, Gauge, ImageItem, StringItem, TextField
- items are referred to by their indexes, which are consecutive integers in the range from zero to size()-1
- If item content is changed, e.g.
  - the set of selected values in a ChoiceGroup is changed
  - the value of an interactive Gauge is adjusted
  - the value in a TextField is entered or modified
  - new date or time in a DateField is entered

  `itemStateChange` callback is signalled

# Images

- Only one high level image format is supported: PNG
- `Images` are either *mutable* or *immutable* depending upon how they are created:
  - immutable images are created by loading the image data from resource bundles, from files, or from the network and may not be modified once created
  - mutable images are created in off-screen memory and may be modified by "painting" into them using a Graphics object (`getGraphics()`)
- `Images` to be placed within `Alert`, `Choice`, `Form`, or `ImageItem` objects are required to be immutable

# Images

- Images can be loaded from
  - white area:
    ```
    public static Image createImage(int width, int height)
    ```
  - other image (copy):
    ```
    public static Image createImage(Image image)
    ```
  - named resource (JAR):
    ```
    public static Image createImage(String name)
    ```
  - byte array:
    ```
    public static Image createImage(byte[] imagedata,
    int imageoffset, int imagelength)
    ```
- Since MIDP 2.0, the developer has access to a (maybe modified) Image data using `getRGB()`

# List Screen

- The `List` class is a `Screen` containing a list of choices
- three types of Lists:
  - IMPLICIT, where select causes immediate notification of the application through CommandListener. The focused element is implicitly and immediately selected when the user traverses the list.
  - EXCLUSIVE, where select operation changes the selected element in the list. Application is not notified.
  - MULTIPLE where select operation toggles the selected state of the focused Element. Application is not notified.

# List Item Selection

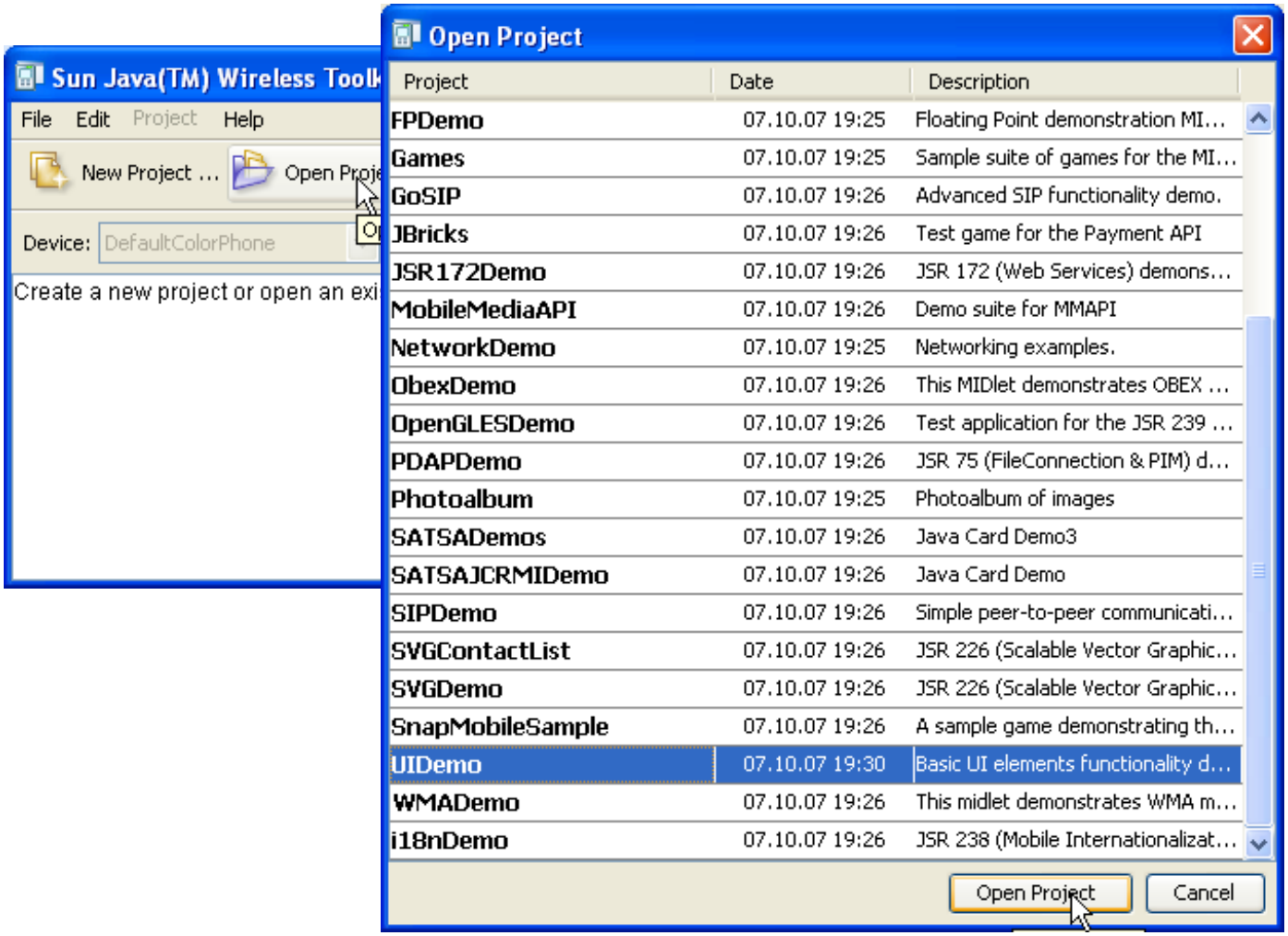
```
\\ create events
Command cmdSelect =
  new Command("Select", Command.ITEM, 2);

\\ …

\\ implement event callback
public void commandAction(Command c, Displayable d) {
  if (d == myContactListScreen) {
    if (c == cmdSelect || c == List.SELECT_COMMAND) {
      // perform appropriate action
      // access element with getSelectedIndex()
      }
    // ...
  }
}
```

# Now it's your turn..

- WTK installed under Linux (later also on Windows):
  - fedora -> siteapps -> start WTK
- If you want to try at home, see http://java.sun.com/products/sjwtoolkit/

# Exercise 2:

- Explore the UIDemo at the WTK



# Exercise 3:

- Load the following two PNGs and list them:

http://www.flaggen-server.de/printflags/Deutschland.png

http://www.flaggen-server.de/printflags/Oesterreich.png