Africa Information
Technology Initiative

Lecture 18: Introduction to J2ME

AITI 2009

Java 2 Micro Edition (J2ME)

- A version of Java designed for mobile computing
- Pros:
  - Its Java!
  - Portable
  - Application development is fast
  - Many new phones come with an interpreter
- Cons:
  - Slow (it's interpreted)
  - Hard to access device specific features
  - Limited as compared to J2SE

J2ME

- Two broad hardware configurations:
  - Connected, Limited Device Configuration (CLDC): mobile phones
  - Connected Device Configuration (CDC): PDAs

- Profile is a specific type of configuration
  - Mobile Information Device Profile (MIDP)

Course Mobiles

- Nokia N70’s support:
  - CLDC 1.0 (newest version is 1.1)
  - MIDP 2.0 (newest version is 2.1)
- Nokia N95's support:
  - CLDC 1.1
  - MIDP 2.0
- Nokia 6300 supports:
  - CLDC 1.1
  - MIDP 2.0

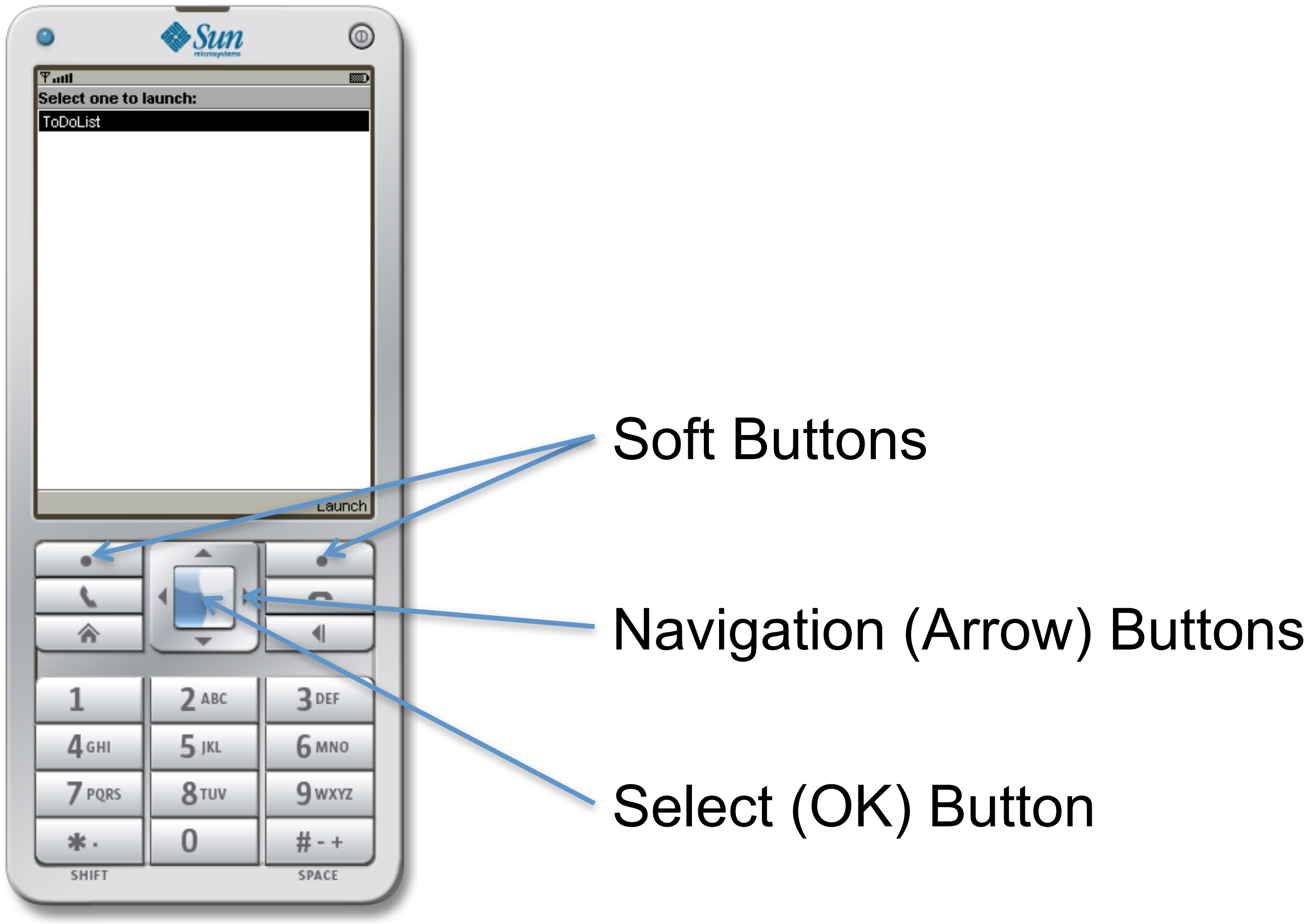
Differences Between J2SE and CLDC/MIDP

- No floating point for CLDC 1.0
- System.out.print/println don't do anything!
  - In the WTK the print to console
- Subset of java.lang
  - Limited implementation of many classes
- Very limited java.util / java.io
- Make sure you are reading the JavaDoc for the J2ME MIDP when you are developing!

Compilation for J2ME

- Extra steps versus desktop Java:
  - Compilation using Java compiler
    - Must include the J2ME Java libraries
  - Pre-verification of bytecode
  - Package the classes application for deployment
    - Create a *jar* archive of the class files
- All this is done for you in the *Java Wireless Toolkit*

Terminology



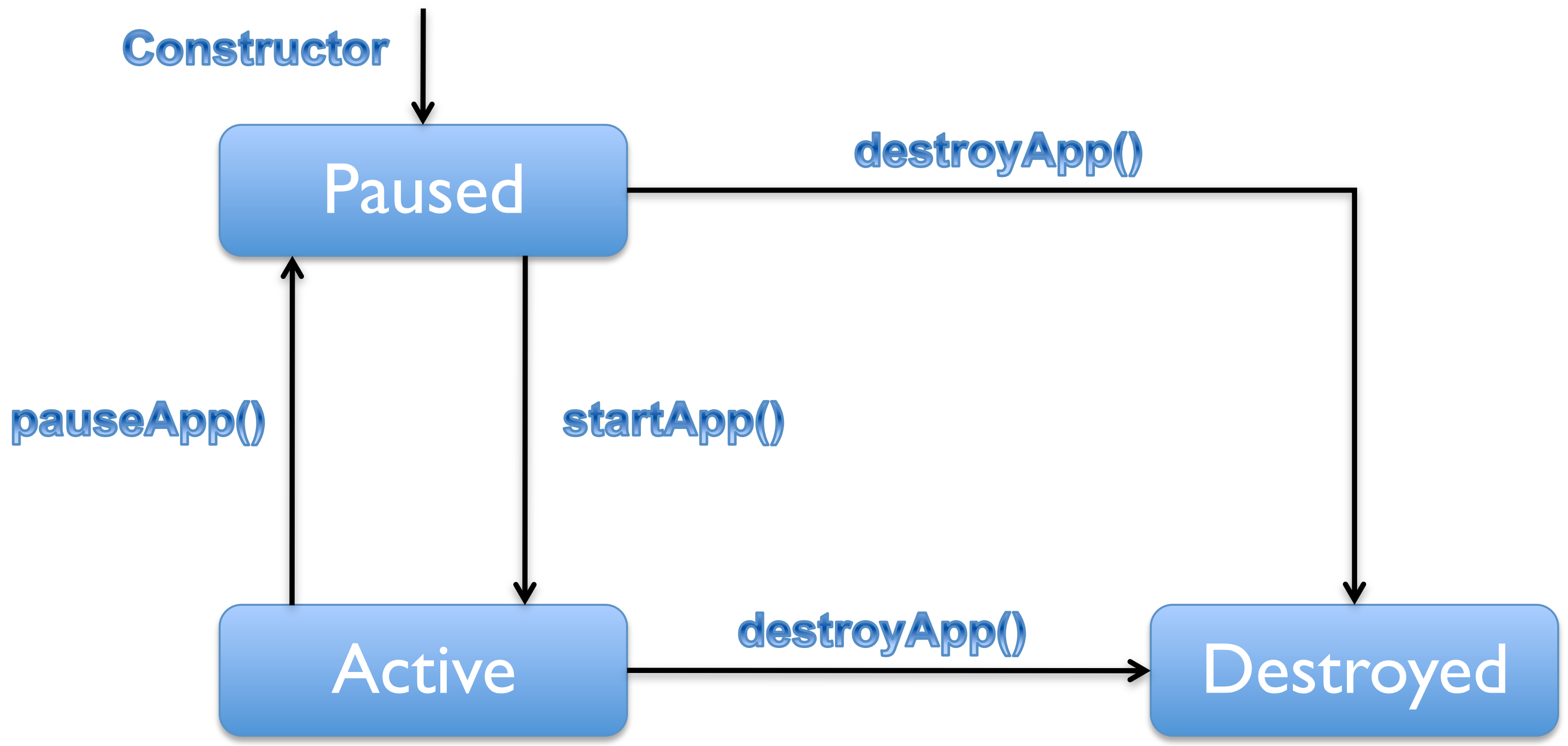
CLDC/MIDP Applications

- All cell phone applications inherit from the MIDlet class
  - javax.microedition.midlet.MIDlet
- The MIDlet class defines 3 abstract methods that the cell phone app must override:
  - `protected abstract void startApp();`
  - `protected abstract void pauseApp();`
  - `protected abstract void destroyApp(boolean unconditional);`

MIDlets

- These methods are called by the J2ME runtime system (interpreter) on your phone.
  - When an application is started, startApp() is called.
  - When an application is paused, pauseApp() is called.
  - When an application is exited, destroyApp(boolean) is called.

Life Cycle of a MIDlet



Constructor versus startApp()

- In the constructor you should create and initialize objects.
  - These are done once per run
- startApp() might be called multiple times for a single run
  - The app is woken from paused
  - In startApp(), you should set the display and be ready for execution

Pausing a MIDlet

- Your application might be paused
  - A call is accepted while the your application is running
  - The runtime will call pauseApp() before your application is paused
- You can pause your app by calling notifyPaused() from within the app
  - Your app is still memory-resident, but the user is taken back to the menu

Exiting a MIDlet

- The runtime system can kill your application
  - User presses hangup command
  - Before it does, it will call destroyApp(true)

- You can kill your app by calling notifyDestroyed()
  - You still have to call destroyApp(true) explicitly

pauseApp() and destoryApp()

- pauseApp()
  - Called when app is paused
  - Close connections / stop threads
- destroyApp(boolean unconditional)
  - Called when an application is about to exit
  - You can ignore the exit if unconditional == false
  - Clean up code goes here
  - Close connections / stop threads
  - Save state if necessary

The MIDlet Philosophy

- Abstraction:
  - Specify the user interface in abstract terms
  - Just specify the components to add
  - A limited set of predefined components
  - Let the MIDP implementation decide on the placement and appearance
  - Ex: add a “done” command somewhere on the screen

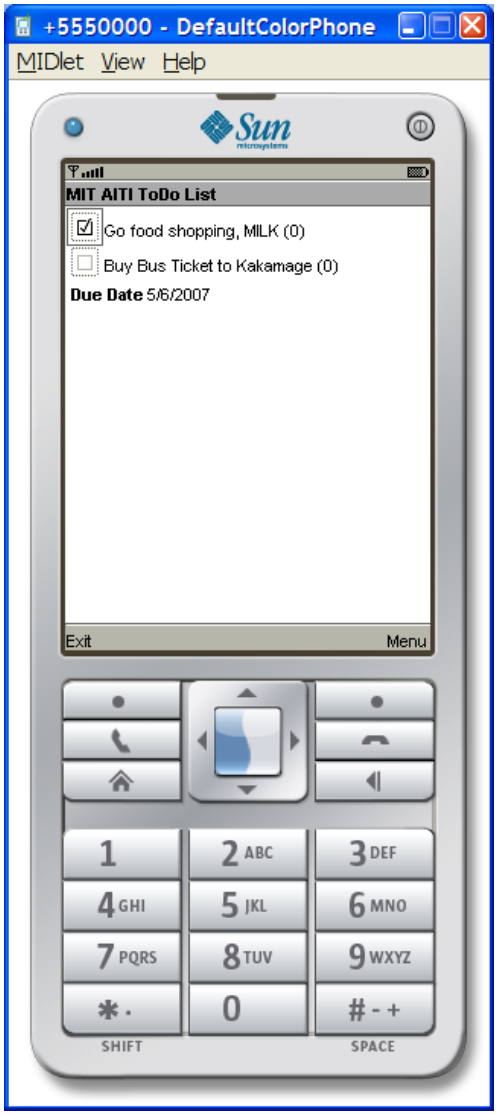
The MIDlet Philosophy

- The device's display is represented by an object of the `Display` class
  - Think of it as an easel
- Objects that can be added to a Display are subclasses of `Displayable`
  - Canvas on the easel
- MIDlets change the display by calling `setCurrent(Displayable)` in `Display`

The MIDlet Philosophy

1. Show a `Displayable` with something on it
2. Wait for input from user
3. Decide what `Displayable` to show next and what should be on this `Displayable`.
4. Go to 1.

Example Application: ToDoList



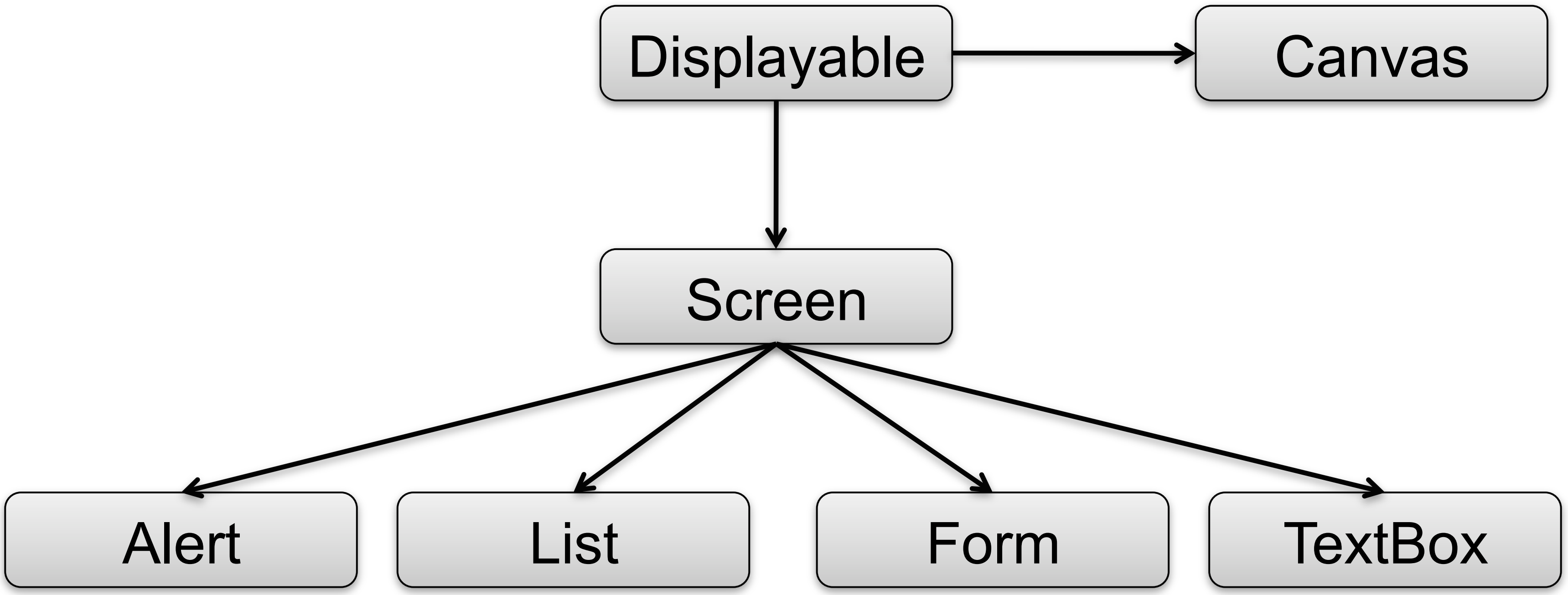
MIT AITI ToDo List

☑ Go food shopping, MILK (0)

☐ Buy Bus Ticket to Kakamage (0)

Due Date 5/6/2007

The Displayable Hierarchy



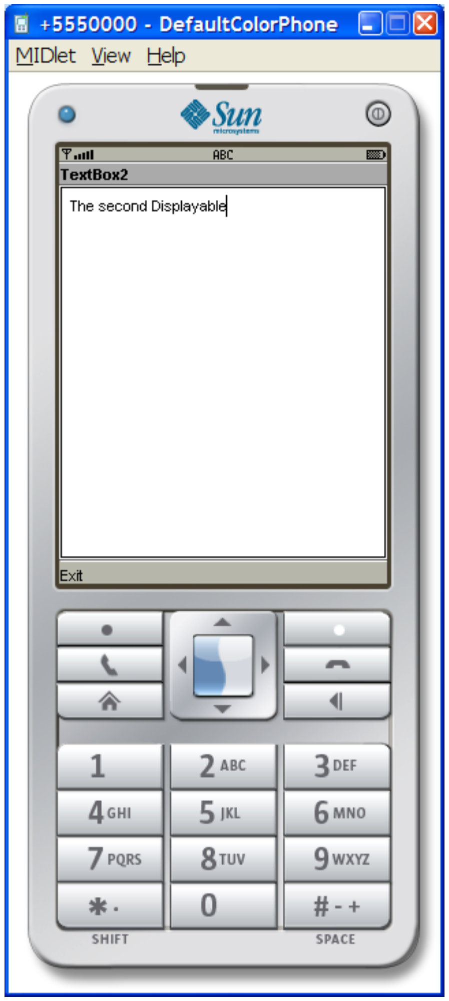
- The appearance of the `Screen` sub-classes are device-dependent
- All these classes are defined in javax.microedition.lcdui

Getting the Display

- The Display object representing the screen is created for you
- You can access it by calling the static method Display.getDisplay(MIDlet)
- Example (inside a subclass of MIDlet):

```
Display display = Display.getDisplay(this);
```

Simplest Displayable: Textbox



The second Displayable

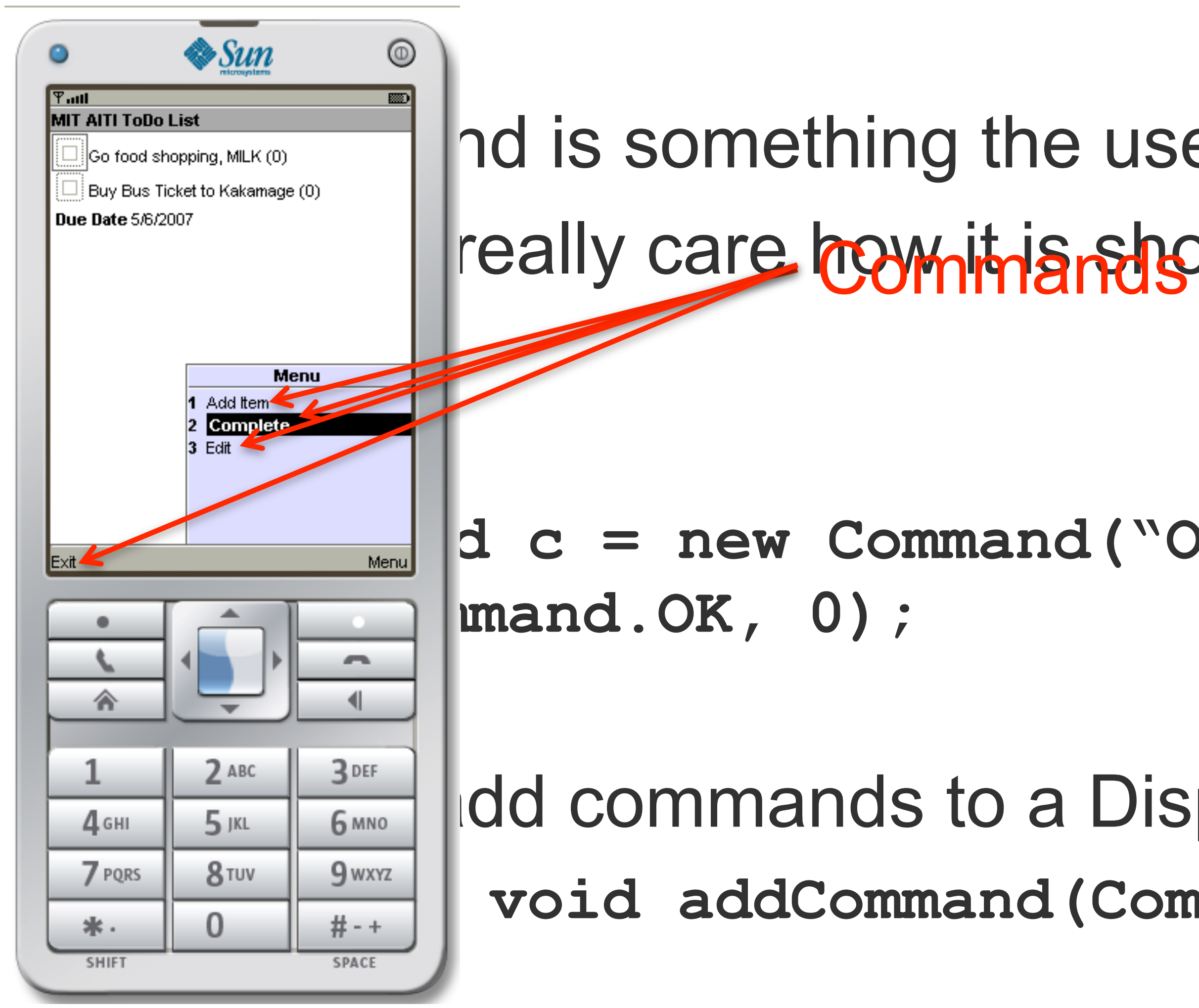
- Show text or allow user to input text

- Creating a TextBox:

```
TextBox textBox2 =
 new TextBox("TextBox2",
  "The Second Displayable",
  32, 0);
```

(has not been displayed yet, just created)

Commands


MIT AITI ToDo List

Go food shopping, MILK (0)

Buy Bus Ticket to Kakamage (0)

Due Date 5/6/2007

nd is something the user can invoke

really care how it is shown on the

```
d c = new Command("OK",
mand.OK, 0);
```

dd commands to a Displayable using:

```
 void addCommand(Command)
```

Commands

To Create a command, you need a name, type and also a priority.

Ex:

Command Text

Command Type

```
Command c = new Command("OK", Command.OK, 0);
```

Priority

• Command text is display on the screen

• Type does not affect the action of a command, only how it is displayed.
  Ex: Command.BACK is placed on left soft-button

• If more than 2 commands on a screen, lowest priority number command may not be grouped

Command Types

There are different types of commands available for you to use:

- Command.OK – Confirms a selection
- Command.CANCEL – Cancels pending changes
- Command.BACK – Moves the user back to a previous screen
- Command.STOP – Stop a running operation
- Command.HELP – Shows application Instructions
- Command.SCREEN – indicates generic type for specific application commands

```
Command launch = new Command("Launch", Command.OK, 0);
Command back = new Command("Back", Command.BACK, 0);
```

Example of Adding Command

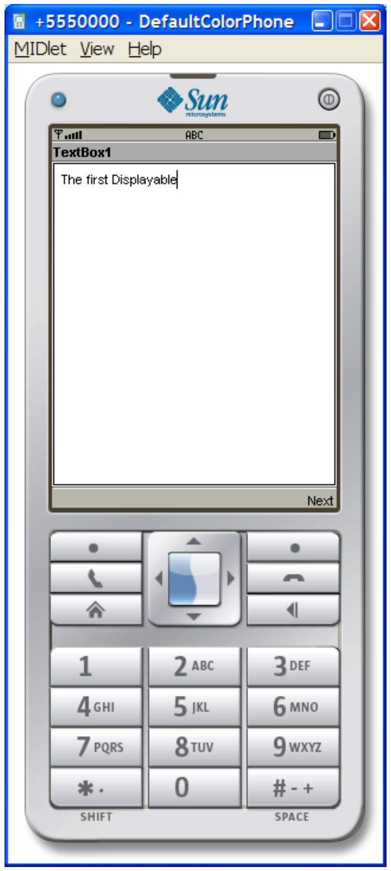
```
Command CMD_NEXT = new Command("Next", Command.OK, 0);

TextBox textBox1 = new TextBox("TextBox1",
    "The first Displayable", 30, TextField.ANY);
  textBox1.addCommand(CMD_NEXT);
```

- You can add as many commands to a display as you want.
- If more than 2, some will be grouped into a "Menu" command
  - Use priority argument of Command constructor

Example of Displaying TextBox

```
Display.getDisplay(this).setCurrent(textBox1);
```

- Get the Display object for the mobile's screen
- Set the current Displayable to textBox1
- The TextBox will be displayed, and the Command will be mapped to a soft-button.



The first Displayable

Responding to Command Events

- When a Command is invoked by the user, a method is called to service the command

- The exact method is:
  - `public void commandAction(`
    `Command c, Displayable d)`
  - c is the Command invoked and d is the Displayable the Command was added to.

Responding to Command Events

- We need to tell the Displayable the object in which to call **`commandAction()`**
- Two Steps:
  1. The class of the object must implement the interface **`CommandListener`**
     - **`CommandListener`** defines **`commandAction()`**
  2. You tell the Displayable which object by calling **`setCommandListener(CommandListener)`** on the Displayable

Example

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

public class HelloWorld extends MIDlet implements
   CommandListener {

    private static Command CMD_EXIT = new
      Command("Exit", Command.EXIT, 0);
    private static Command CMD_NEXT = new
      Command("Next", Command.OK, 0);

    private TextBox textBox1;
    private TextBox textBox2;
```

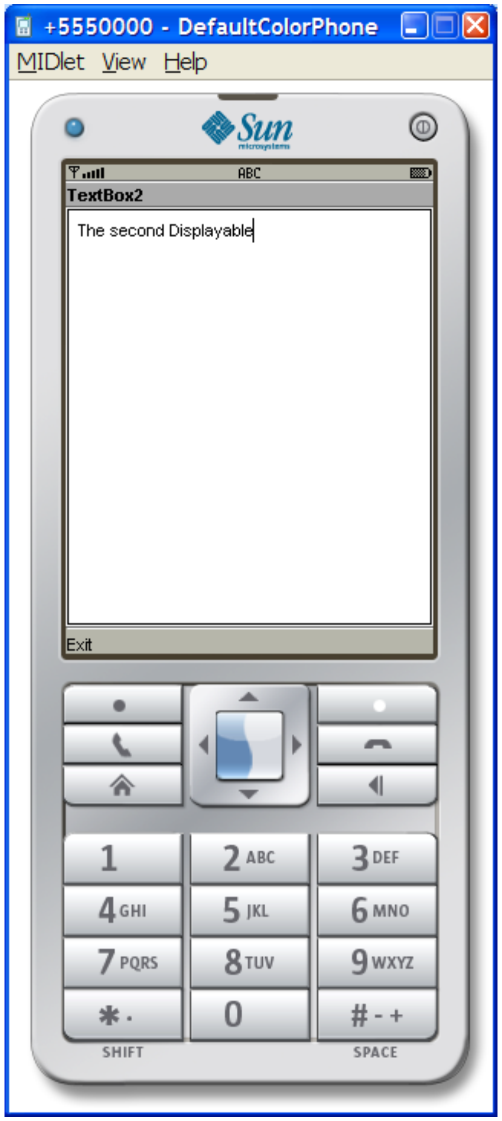
Example

```
 public HelloWorld()
 {
   textBox1 = new TextBox("TextBox1",
     "The first Displayable", 30, TextField.ANY);
   textBox1.addCommand(CMD_NEXT);
   textBox1.setCommandListener(this);


   textBox2 = new TextBox("TextBox2",
     "The second Displayable", 30, TextField.ANY);
   textBox2.addCommand(CMD_EXIT);
   textBox2.setCommandListener(this);
}
```

Example

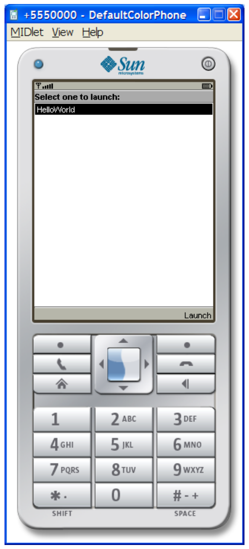
```
public void startApp() {
   Display.getDisplay(this).setCurrent(textBox1);
}

public void commandAction(Command c, Displayable d)
{
   if (d == textBox1 && c == CMD_NEXT)
        Display.getDisplay(this).setCurrent(textBox2);
   else if (d == textBox2 && c == CMD_EXIT) {
       destroyApp(true);
       notifyDestroyed();
   }
}

public void pauseApp(){}    public void destroyApp(boolean u) {}   }
```

Example Run



The second Displayable

Flow of Execution



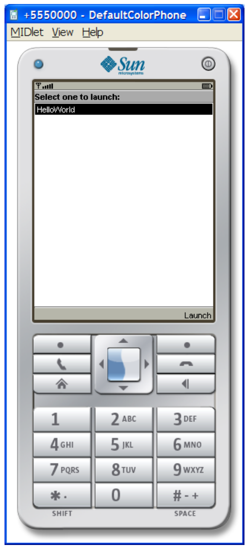
J2ME
Runtime

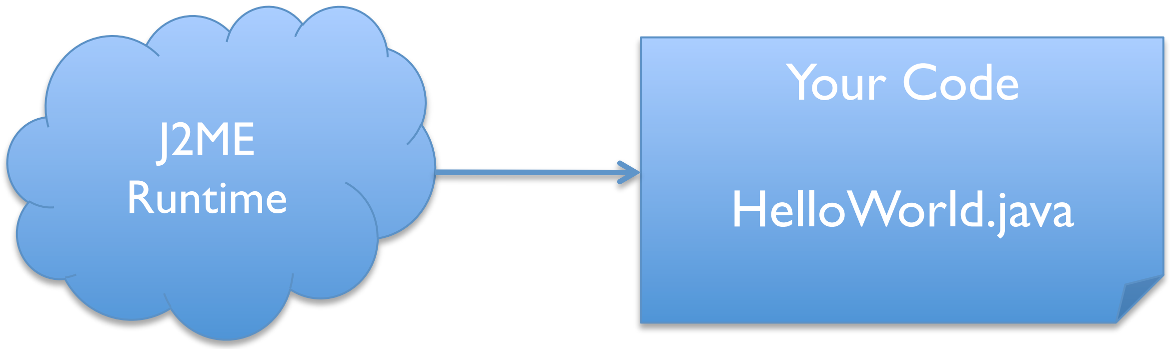
Your Code

HelloWorld.java

User starts application

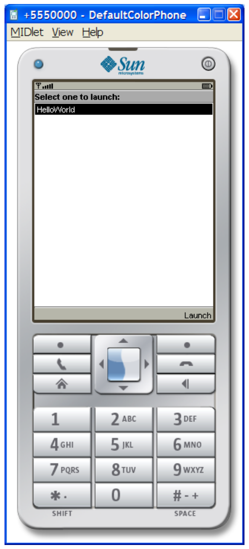
Flow of Execution

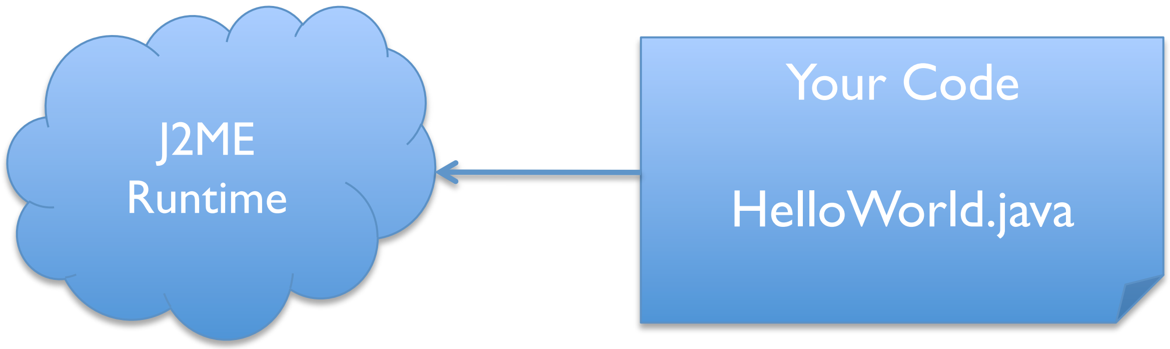




J2ME runtime is invoked
Calls HelloWorld()
constructor

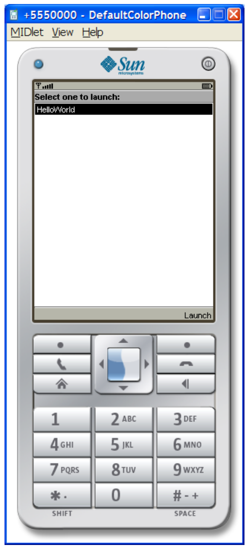
Flow of Execution

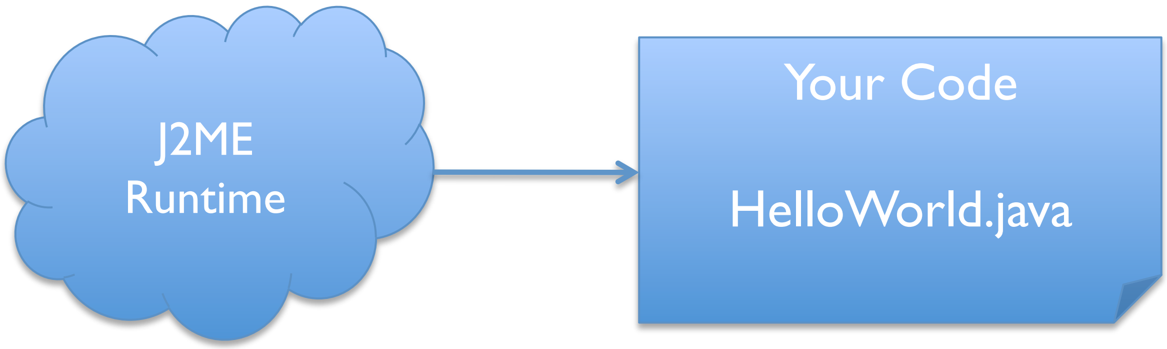




HelloWorld() constructor is executed and returns

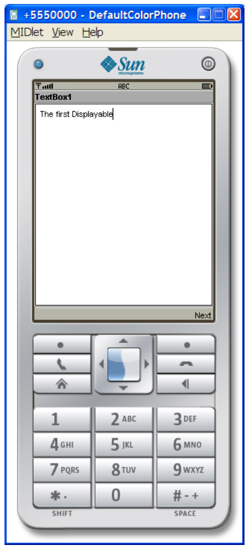
Flow of Execution

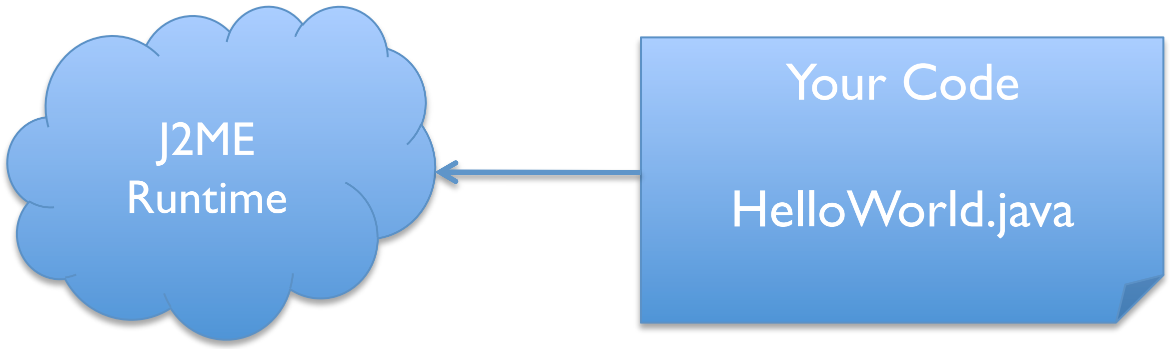




J2ME runtime calls
HelloWorld.startApp()

Flow of Execution

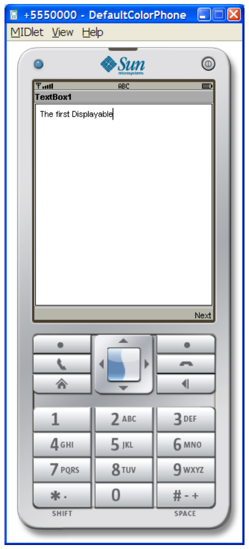


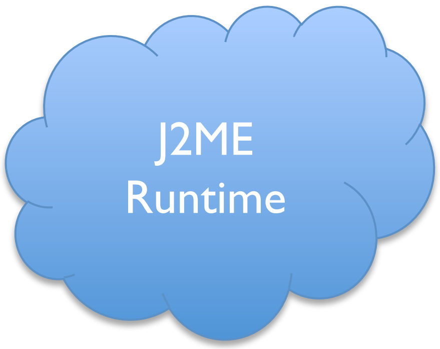


HelloWorld.startApp is called:

Displays textBox1 and returns

Flow of Execution

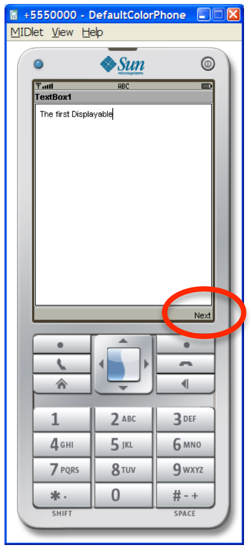




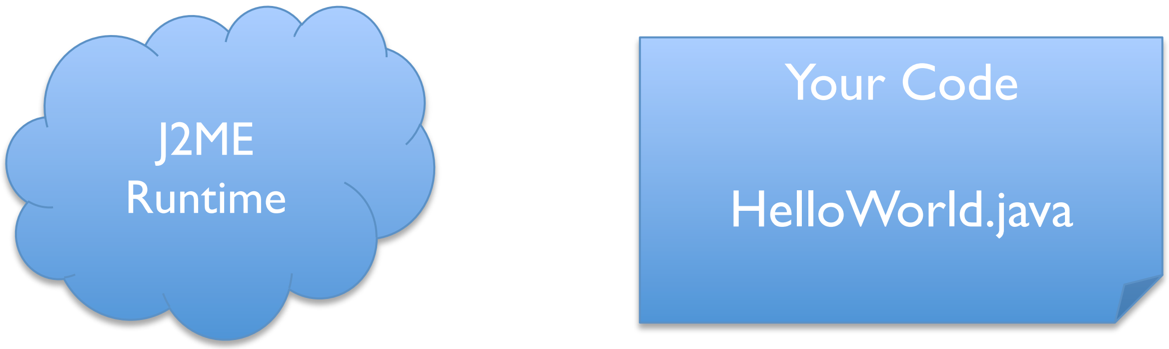
Your Code
HelloWorld.java

J2ME Runtime is waiting for user input

Flow of Execution

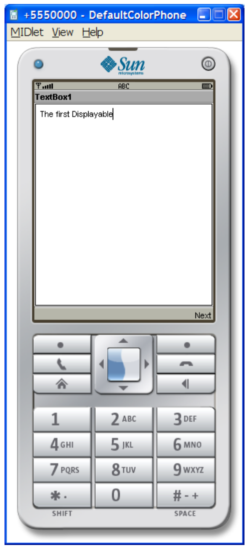


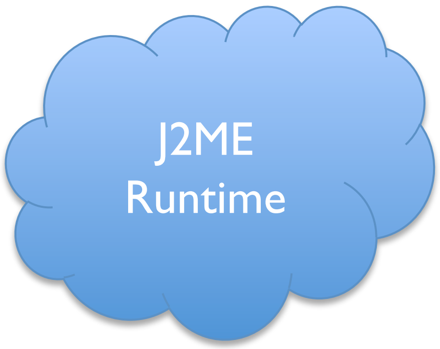
The first Displayable



User presses "Next"

Flow of Execution



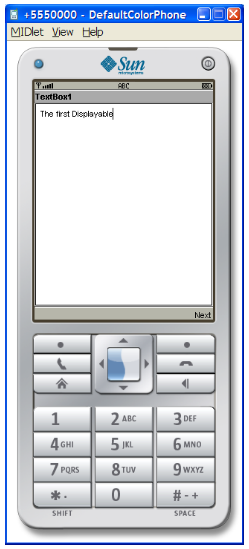


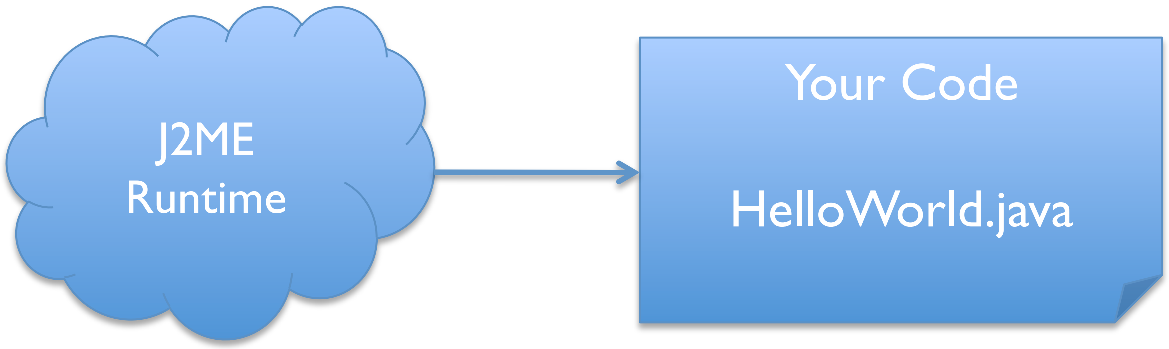
Your Code
HelloWorld.java

J2ME Runtime catches the key press.

Finds HelloWorld obj is registered as Listener for textBox1

Flow of Execution



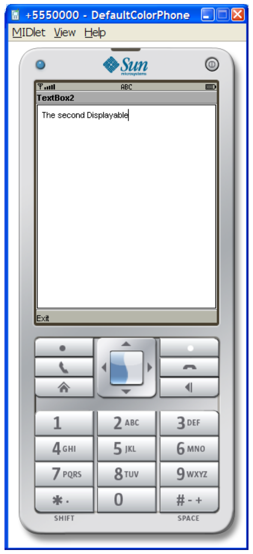


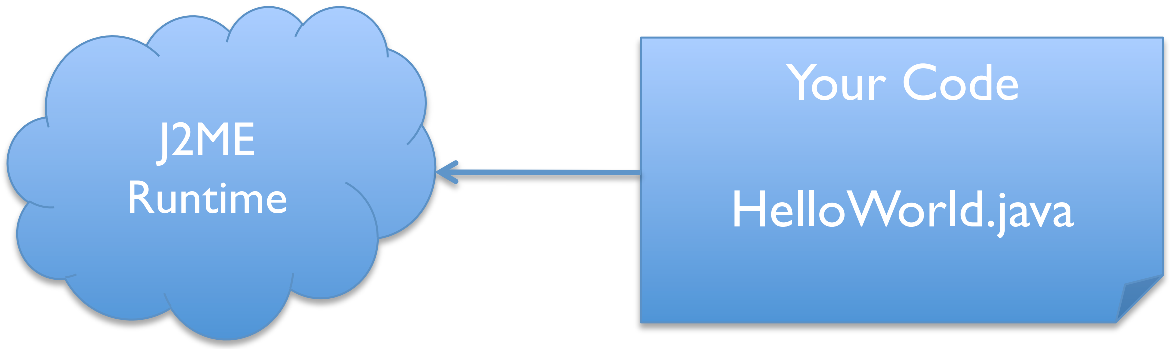
J2ME Runtime calls

```
CommandAction(CMD_NEXT,
              textBox1)
```

on HelloWorld obj.

Flow of Execution

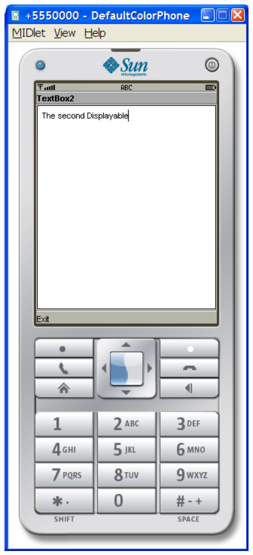


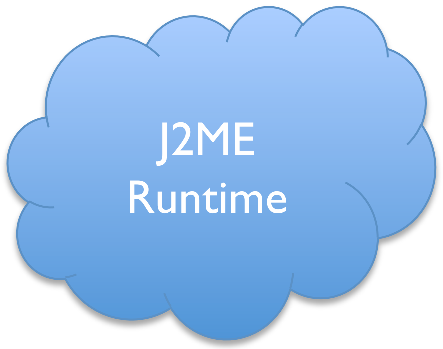


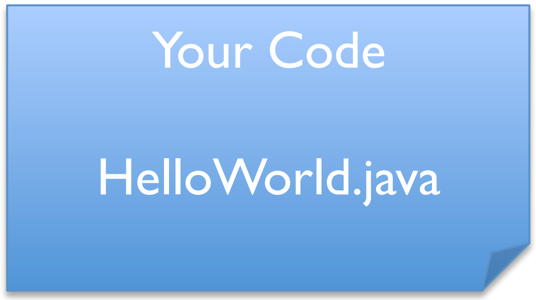
In
`CommandAction(CMD_NEXT,
              textBox1)`

first `if` statement is true:
Display textBox2

Flow of Execution

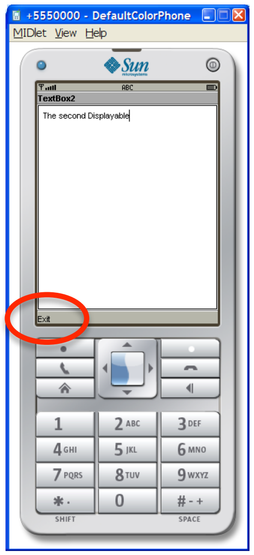




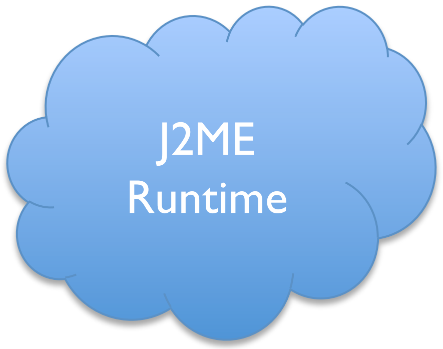


J2ME Runtime is
waiting for user input

Flow of Execution



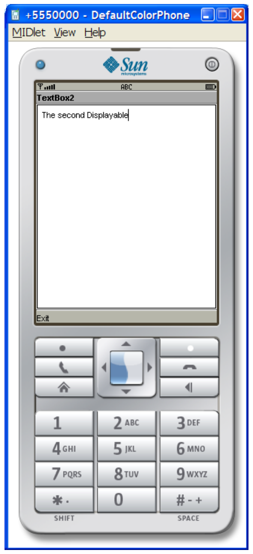
The second Displayable

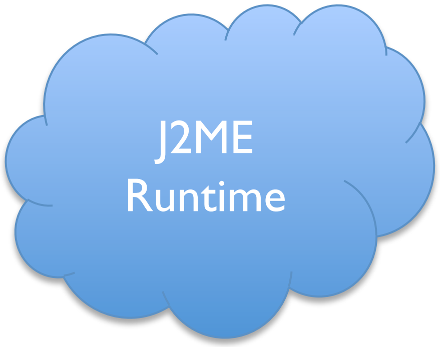


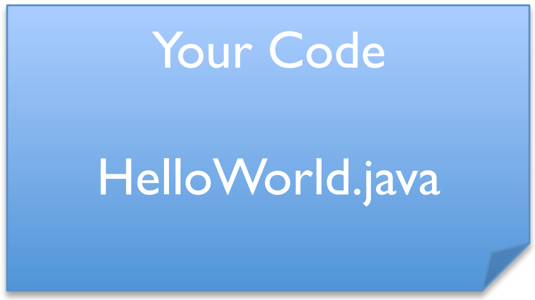
Your Code
HelloWorld.java

User presses exit

Flow of Execution



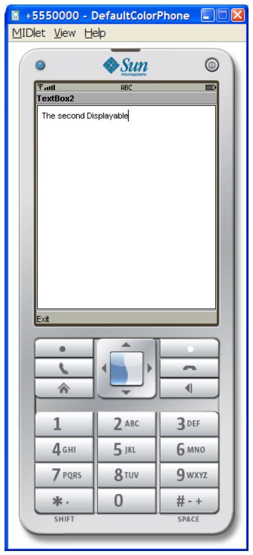


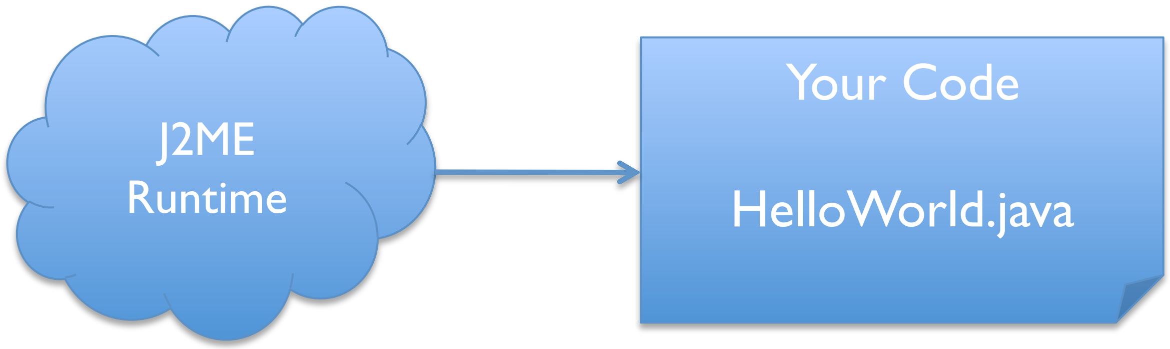


J2ME Runtime catches the key press.

Finds HelloWorld obj is registered as Listener for textBox2

Flow of Execution



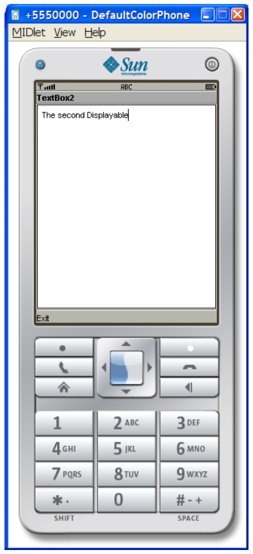


J2ME Runtime calls

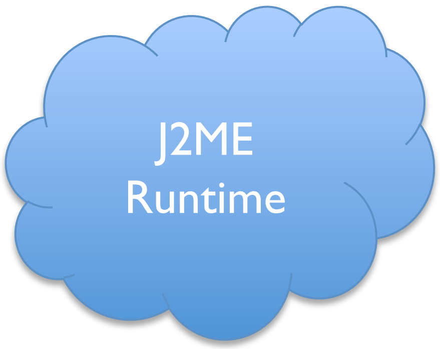
```
CommandAction(CMD_EXIT,
                    textBox2)
```

on HelloWorld obj.

Flow of Execution



The second Displayable



Your Code
HelloWorld.java

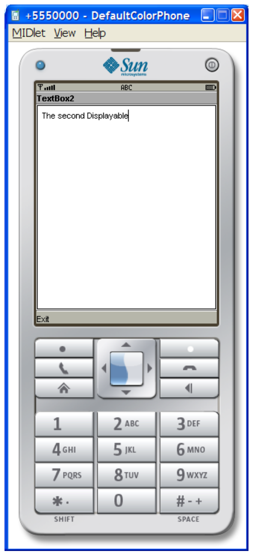
In

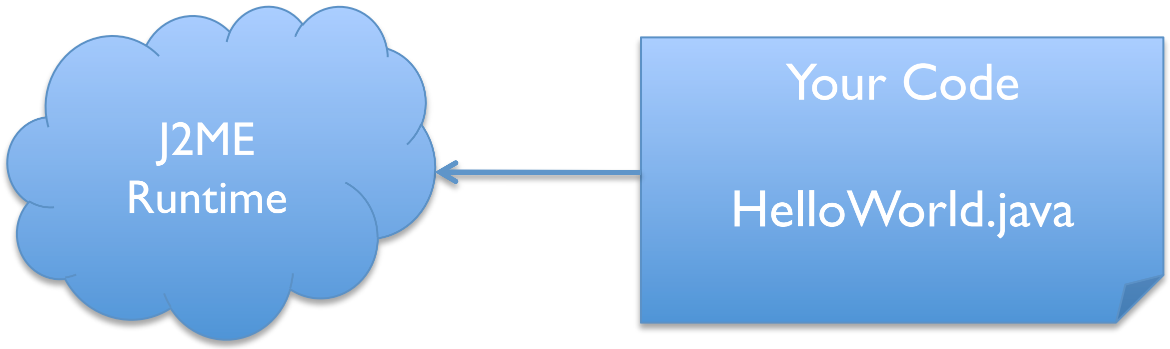
```
CommandAction(CMD_NEXT,
              textBox2)
```

second `if` statement is true:

```
    destroyApp(true);
```

Flow of Execution

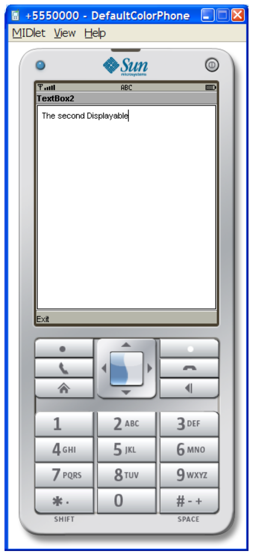


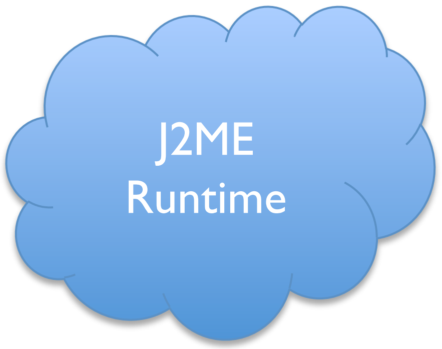


In
```
CommandAction(CMD_NEXT,
              textBox2)
```

second `if` statement is true:
```
    destroyApp(true);
    notifyDestroyed();
```

Flow of Execution

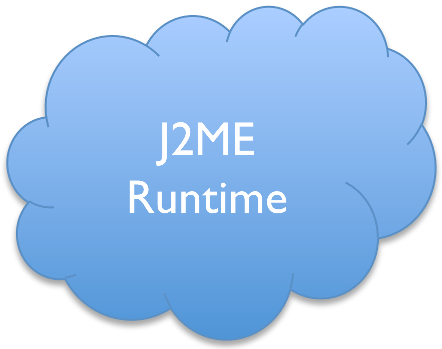




Your Code
HelloWorld.java

J2ME Runtime frees HelloWorld's memory and exits application.

Flow of Execution



J2ME Runtime frees HelloWorld's memory and exits application.