Wireless Java with J2ME

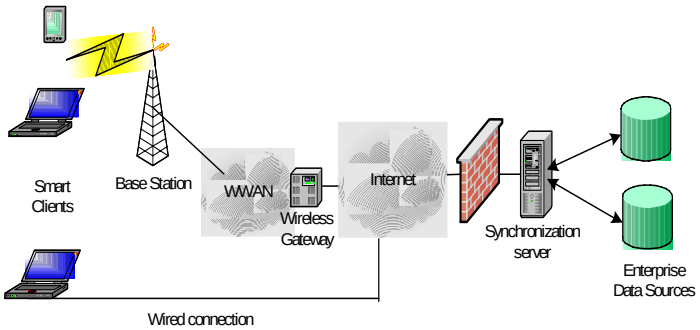
David Tipper
Associate Professor
Department of Information Science and Telecommunications
University of Pittsburgh

dipper@mail.sis.pitt.edu
http://www.sis.pitt.edu/~dtipper/2727.html

Slides 7+
+ based on material from Ola Bo at Molde College, Norway
and
F. Ricci at Free University Bozen-Bolzano, Italy

Smart Client Architecture

- Custom software executes on Devices
- Components:
  - Smart clients
  - Synchronization server
  - data/content source
  - may include proxies or gateways

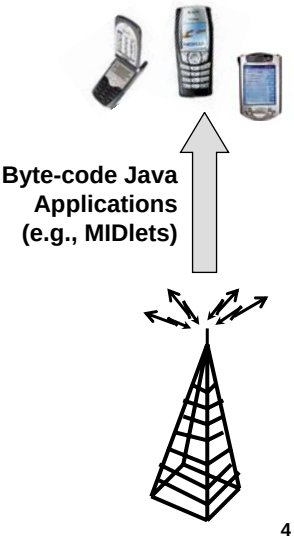


Platform Independence

- Biggest problem in smart client development is wide variety of mobile devices
- Different operating systems, different CPUs, memory, displays, etc.
- If develop application in native code (e.g., assembly language) for a specific platform (e.g., Nokia 655) will likely not work on another platform
- Use virtualization to strive for platform independence

J2ME – Java for Hand-Held Devices

- Platform independence main goal of Java
  - The same byte-code Java application can be downloaded and executed by all Java-enabled devices
  - Pre-verification at compile time to verify if an application can run on Java Virtual Machine
  - Java 2 Micro Edition (J2ME) ports this idea to handheld devices



Why Java for Wireless Devices

- The wireless Internet revolution will transform wireless devices from **voice-oriented** to **extensible Internet-enables devices**
- Devices need to **support dynamic downloading of new software** and running software written not only by the device manufactures
- Java is (becoming) a standard application development language for wireless devices
- Is **not** defining **a new operating system** (Symbian or PocketPC) but it standardizes **a portable wireless application development environment**
- The environment **can be added on top** of existing software and hardware solutions that the device manufacturer already have.

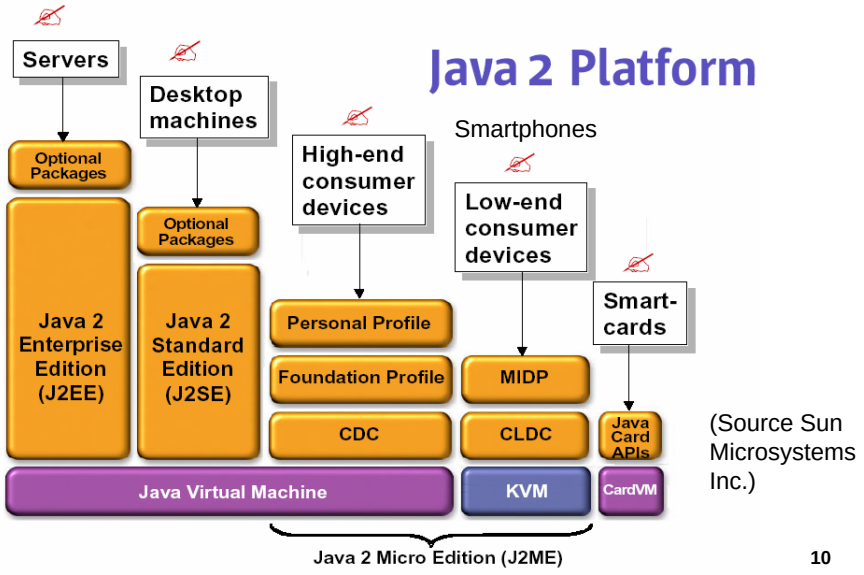
What Java Offers on Wireless Devices

- **Dynamic delivery of content:** new application, services and content can be downloaded dynamically
- **Security:** class file verification, a well-defined application programming interface, security features, ensure that applications cannot harm the device or network
- **Cross-platform compatibility:** standardized language features and libraries implies that the application can run on different devices
- **Enhanced user experience and interactive content**
- **Offline access:** applications can be used without active network connection
- **Object oriented:** good abstraction mechanisms and higher level programming constructs
- **Large developer community:** more than 3 millions Java developers worldwide.

Overview of the Java 2 Platform

- The Java 2 Platform is split into three editions
- J2SE (Standard Edition)
  - designed for desktop computers.
- J2EE (Enterprise Edition)
  - designed for server based applications - J2SE and adds APIs for server-side computing.
- J2ME (Micro Edition)
  - designed for small/handheld devices (set-top boxes, sensors, mobile phones)
  - subset of J2SE APIs plus j2ME specific class libraries
- Each edition provides a complete environment for running Java applications including the Java Virtual Machine and runtime classes

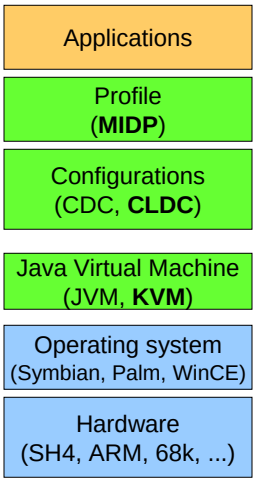
J2ME in the Java landscape



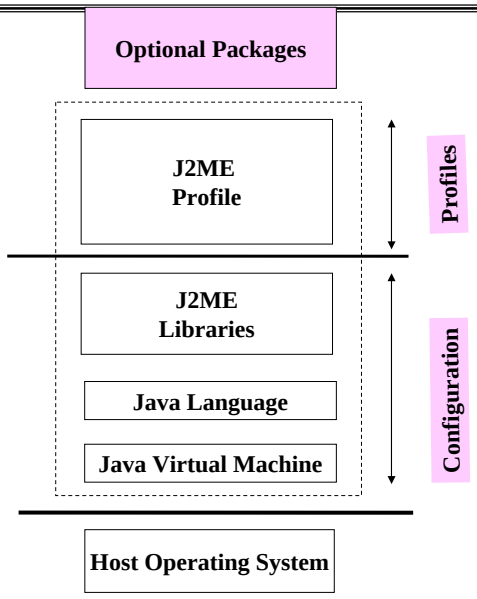
(Source Sun Microsystems Inc.)

J2ME Architecture

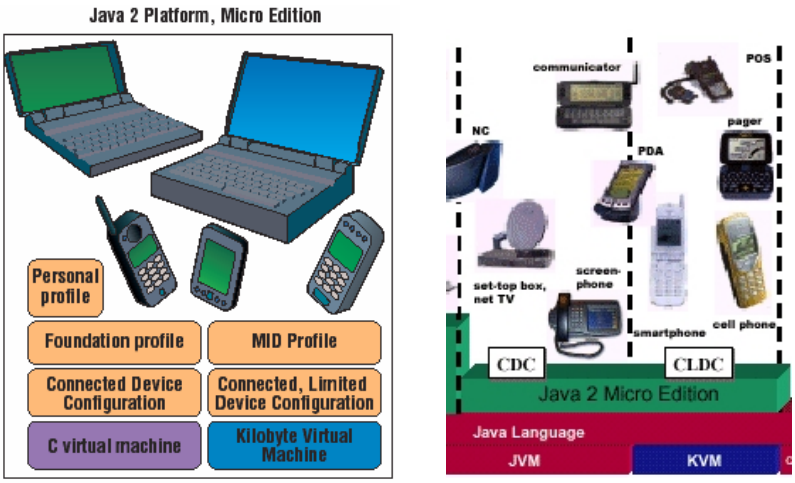
- JVM layer
  - Virtual machine – hides platform
  - Kilobyte Virtual Machine – for resource-constrained devices
- Configuration layer
  - Defines a minimum set of Java class libraries that is useful for developing applications to run on a range of devices
  - CLDC and CDC
  - E.g., java language, input/output
- Profile layer
  - Defined class libraries (supplemental to Configuration) that targets a particular type of devices
  - MIDP (Mobile Information Device Profile)
  - Class libraries, e.g., for building user interface, making network connections, and controlling application life cycles



J2ME Core Concepts



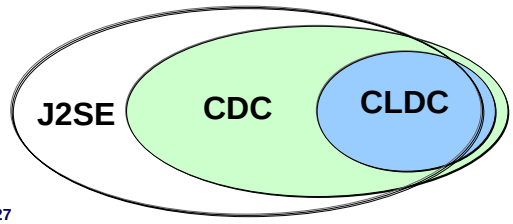
J2ME Architecture



From S. Helal, "Pervasive Java," *IEEE Pervasive Computing*, Vol. 1, No. 1, Jan.-March 2002, pp. 82-85.

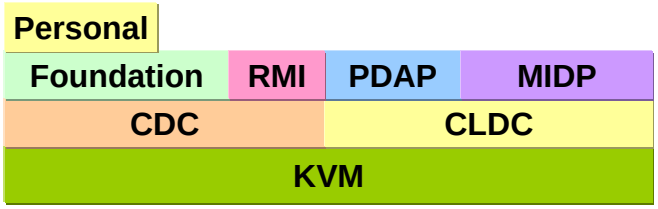
J2ME Configurations

- A Configuration defines class libraries for a particular category of devices that share similar characteristics such as memory budget, processing power, and network connectivity
  - CLDC Connected Limited Device Configuration
    - The smaller of the two configurations (subset)
    - Designed for devices with slow processor and limited memory (160-512k RAM for Java)
    - Mobile phones, two-way pagers and PDA's
  - CDC Connected Device Configuration
    - Designed for devices that have more memory, faster processors, and greater network bandwidth
    - Set top boxes, navigation system, and high-end wireless devices e.g. Nokia 9500 and some PDAs



J2ME Profiles

- A set of specific APIs available for a single device or a group of devices
- Mostly, related to GUI, buttons, local data storage
  - MIDP (Mobile Information Device Profile)
  - PDAP (PDA Profile)
  - Foundation Profile (non-GUI networked device)
    - A basis for Personal Profile
  - Personal and RMI Profiles



Examples of CDC and CLDC

Profile	Configu-ration	Virtual Machine	Device Examples
MIDP	CLDC	KVM	Cellphones, Pagers
PDAP	CLDC	KVM	PDAs
Foundation	CDC	CVM	Foundation for Personal Profile
Personal	CDC	CVM	PocketPC, Tablets
RMI	CDC	CVM	Any
Multimedia	CLDC/CDC	KVM/CVM	Any
Gaming	CLDC/CDC	KVM/CVM	Any
Telephony	CDC/CLDC	KVM/CVM	Cellphones

From J. White and D. Hemphill, *Java 2 Micro Edition*, Manning Publications Co., 2002.

CLDC 1.1 and MIDP 2.0 packages

MIDP 2.0	CLDC 1.1
javax.microedition.lcdui	java.lang
javax.microedition.lcdui.game	java.lang.ref
javax.microedition.media	java.io
javax.microedition.media.control	java.util
javax.microedition.midlet	javax.mcroedition.io
javax.microedition.pki	
javax.microedition.rms	

Devices Evolution (Nokia)

6600 (2003)

MIDP 2.0
CLDC 1.0
Bluetooth API (JSR-82 No OBEX)
Mobile Media API (JSR-135)
Nokia UI API
Wireless Messaging API (JSR-120)

N70 (2005)

MIDP 2.0
CLDC 1.1
Bluetooth API (JSR-82)
FileConnection and PIM API (JSR-75)
JTWI (JSR-185)
Mobile 3D Graphics API (JSR-184)
Mobile Media API (JSR-135)
Nokia UI API
Web Services API (JSR-172)
Wireless Messaging API (JSR-120)

N95 (2007)

MIDP 2.0
CLDC 1.1
Advanced Multimedia Supplements (JSR-234)
Bluetooth API (JSR-82)
FileConnection and PIM API (JSR-75)
JTWI (JSR-185)
Location API (JSR-179)
Mobile 3D Graphics API (JSR-184)
Mobile Media API (JSR-135)
Nokia UI API
Scalable 2D Vector Graphics API (JSR-226)
Security and Trust Services API (JSR-177)
SIP API (JSR-180)
Web Services API (JSR-172)
Wireless Messaging API (JSR-205)

For a list see http://developers.sun.com/techtopics/mobility/device/device

Developing a J2ME Application

- Define application requirements
- Application design (control)
  - May need to break into small apps
  - What type of connections
- User Interface design
- Data storage design
- Select targeted devices
- Choosing the right configuration and profile
- Write code, compile, verify, pack, test, and deploy.

J2ME for mobile and wireless devices

Specific libraries for mobile and wireless devices: GUI, storage
A subset of the Java libraries adapted to a lowest common denominator for mobile devices
The small memory footprint virtual machine – corresponding to JVM in standard Java

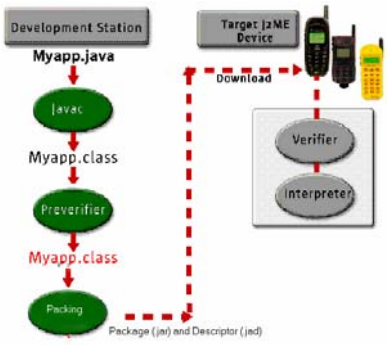
CLDC

- CLDC library APIs can be divided into two categories:
  - Classes that are a subset of the J2SE APIs
    - These classes are located in the following packages: java.lang.*, java.util.*, java.io.*
  - Classes specific to the CLDC
    - These classes are located in the javax.microedition package

- Java Specification Request–JSR 139 (CLDC1.1)
- CLDC **not** contain:
  - UI, application life cycle, Persistence, and special interfaces for the device.

CLDC-based class special handling

- Verification: identify and reject invalid class files
- In J2SE, verification is performed by JVM on device at runtime
  - This class file verification process is expensive and time consuming, and therefore not ideal for small, resource-constrained devices
- CLDC proposes two-phase class file verification process
  - Preverification process performed at development station – to move most of the verification work off device
  - Supports faster start up of CLDC based apps
- Preverification and packaging may be done using
  - Command line tools
  - KToolbar in the WTK

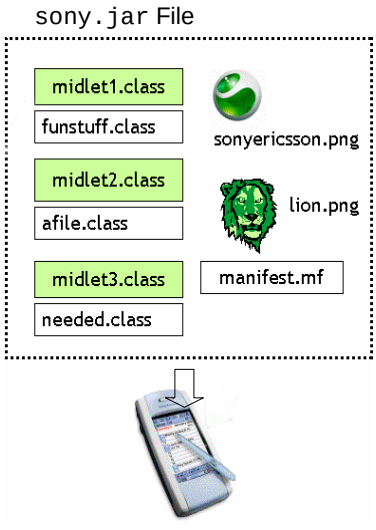


MIDP

- The first J2ME profile, and the most mature and widely deployed one
- Primarily deployed on PDAs and cellphones
- Complying devices are readily available
  - http://developers.sun.com/techtopics/mobility/device/device
- MIDP APIs can be divided into two parts:
  - Two classes inherited directly from the J2SE API
    - Java.util.Timer, Java.util.TimerTask
  - MIDP-specific classes.
    - These classes are located in the following packages:
      - javax.microedition.midlet, javax.microedition.lcdui, javax.microedition.io, and javax.microedition.rms
    - New standard MIDP released Nov 2002 (JSR 118)
      - Add new packages: javax.microedition.lcdui.game, javax.microedition.media., javax.microedition.media.control, and javax.microedition.pki

MIDP applications AKA MIDlets

- A MIDP application is called a MIDlet
  - Must extend the MIDlet class
- A MIDlet suite is a bundle of MIDlets.
  - can access same persistent data
- MIDP Packaging
  - Midlet suites are packaged in compressed Java Archive (JAR) format (JAR) files
    - The preverified class files in the suite
    - Resource files (for example icons, sounds)
    - A manifest file describing a JAR comtent
  - Each JAR file comes with a Java Application Descriptor (JAD) metafile containing instructions describing its contents for deployment (name, description, version etc.)



JAD (Java Application Descriptor)

- A text file that lists important information about a set of MIDlets packaged together into a single JAR file
- Description includes many attributes:
  - Name of MIDlet suite, location and size of JAR file, and configuration and profile requirement
- It is used by the Application Management Software (AMS) to download and install the application
- AMS is a software on a device responsible for downloading, (un) installing, and managing life-cycle of MIDlets

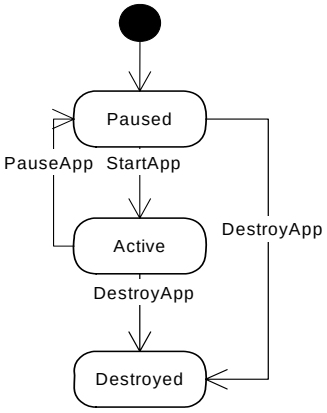
sony.jad

```
MIDlet-Name: SonyMenu
MIDlet-Version: 1.0.0
MIDlet-Vendor: University of Pittsburgh
MIDlet-Description: A sample MIDlet suite
MIDlet-Info-URL: http://www.tele.pitt.edu
MIDlet-Jar-URL: http://localhost/sony.jar
MIDlet-Jar-Size: 3000
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-1: funstuff, midlet1
MIDlet-2: afile, midlet2
MIDlet-3: needed, midlet3
```


Provisioning

- Provisioning is getting the application into the device. Two basic ways to install a MIDlet suite:
  - Direct method:
    - involving some direct connection between the device and the development platform (with synchronization software), commonly USB cable, IR, or bluetooth link
  - "Over The Air (OTA) " using HTTP over wireless protocols
    - Link to the application .jad file from a wap compatible –page (WML, XHTML-MP) using a device's browser
- OTA not standardized in MIDP 1.0,
- OTA provisioning standardized in MIDP 2.0

Application Life Cycle



Application State Description
Initialized and ready to run
Pause: Release some resources if allocated and become inactive
Active: Resources acquired, application executing
Destroyed: All resources released, All threads stopped

Application LifeCycle methods

(source javadoc for class javax.microedition.MIDlet, consult documentation for details)

Called by device (AMS)	Called by midlet
*`startApp()`** AMS invokes the method *startApp*() to move Midlet from Paused state to Active state	`resumeRequest()` Midlet asks AMS to reactivate the MIDlet
*`pauseApp()`** AMS invokes the method *pauseApp* () to move MIDlet from Active state to Paused state	`notifyPaused()` MIDlet calls this method to shift itself to Paused state.
*`destroyApp(boolean)`** AMS invokes the method *destroyApp* () to move MIDlet to Destroyed state	`notifyDestroyed()` MIDlet calls this method to shift itself to Destroyed state

** abstract methods, to be implemented by programmer when subclassing MIDlet.*

MIDlet Application Skeleton

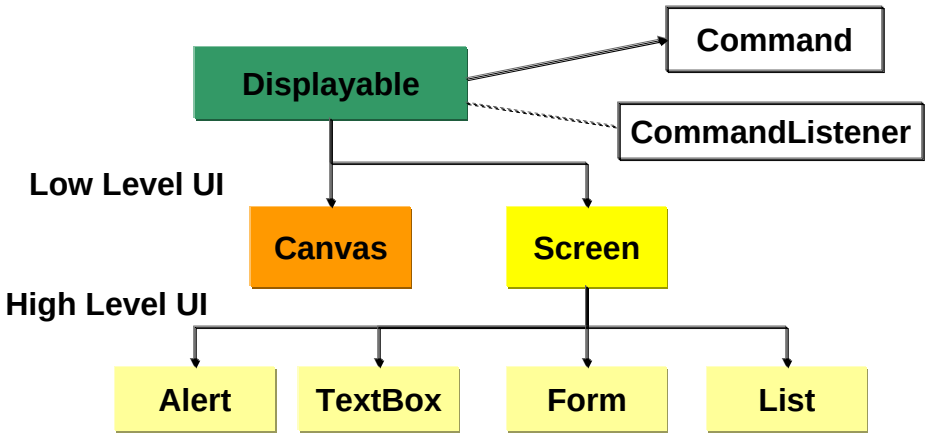
- All MIDlets have this common skeleton
  - Constructor called once
    - E.g. for initialization
  - `startApp()` called when system active MIDlet
  - `pauseApp()` called when system pauses MIDlet
  - `destroyApp()` called when system destroys MIDlet

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

public class ExampleMIDlet extends MIDlet{
    public ExampleMIDlet() {
    }
    public void startApp() {
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```


MIDP 1.0 User Interface

- Low Level User Interface API (Canvas)
  - Designed for applications that needs precise placement and control of graphic elements
  - Offer higher flexibility to implement a much better graphics
  - E.g. needed for interactive game applications
- High Level User Interface API (Screen)
  - Provide a series of predefined graphical elements
  - Provide very little control over look and feel
  - Provide abstraction from low-level graphic management

MIDP UML `Displayable` Class



MIDP 1.0 High Level UI

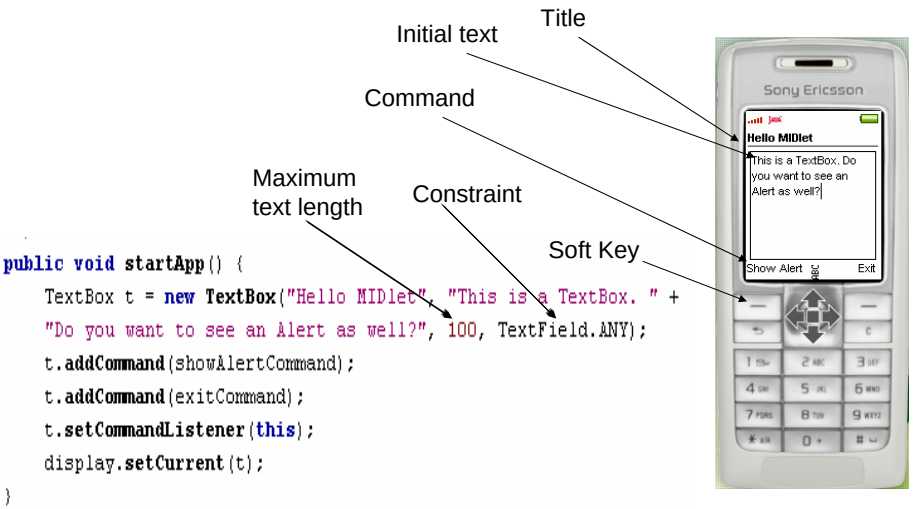
- Using whole Screens – four kinds of Screen
  1. `Alert` - message
  2. `TextBox` – Editable or non-editable textbox
  3. `Form`
  4. `List`
  - Screens have a Title
  - Commands can be added to screens
- How commands appear is device dependent (Soft key, Menu item, ...)
  - Commands must be handled by class implementing a `CommandListener` Interface
  - (for further details see javax microedition.lcdui.Command)
  - Show a screen by using `display.setCurrent(myScreen)`

TextBox

- A screen that allows the user to enter or edit text
- Input **constraints** specifies content type and other aspects of the TextBox (i.e., defines a specific set of characters that are valid to be entered)
  - ANY
  - EMAILADDR
  - NUMERIC
  - PHONENUMBER
  - URL
  - DECIMAL
  - PASSWORD

new constraints from MIDP 2.0
- UNEDITABLE
- SENSITIVE
- NON_PREDICTIVE
- INITIAL_CAPS_WORD
- INITIAL_CAPS_SENTENCE

TextBox Code




```
public void startApp() {
    TextBox t = new TextBox("Hello MIDlet", "This is a TextBox. " +
    "Do you want to see an Alert as well?", 100, TextField.ANY);
    t.addCommand(showAlertCommand);
    t.addCommand(exitCommand);
    t.setCommandListener(this);
    display.setCurrent(t);
}
```

Alert

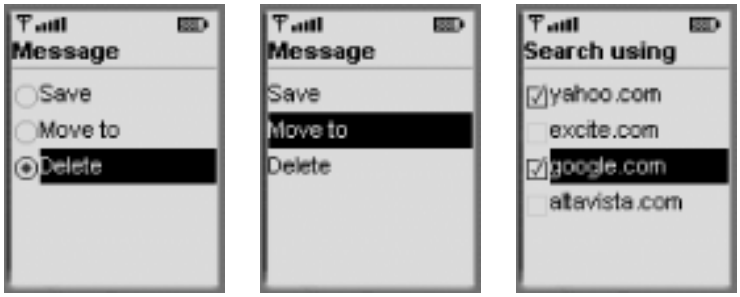
- A screen that shows a message and an optional emage to the user for a certain period of time before proceeding to the next screen.
  - A timeout can be set
  - An image can be added
  - Is displayed by a special version of display.setCurrent(Alert,Screen)
    - The second parameter is what to display when alert screen gone. (display.getCurrent can be used here)

Alert and Command Listener Code

```
public void commandAction(Command c, Displayable s) {
    if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    }
    if (c==showAlertCommand){
        Alert al=new Alert("Alert Message",
        "You wanted to see an alert? \nThis is one",null,AlertType.INFO);
        display.setCurrent(al, display.getCurrent());
    }
}
```

List

- A screen that contains a list of choices
- There are three types of lists (`Choice.type`)
  1. `EXCLUSIVE` selecting one deselects previous
  2. `IMPLICIT` selecting invokes command
  3. `MULTIPLE` selecting more than one possible



From: Q. H. Mahmoud, *Learning Wireless Java*, O'Reilly, Inc., 2001

Exclusive List Code

```
List list = new List("Border Style",
                     Choice.EXCLUSIVE);
list.append("None", null);
list.append("Plain", null);
list.append("Fancy", null);
list.addCommand(backCommand);
list.addCommand(okCommand);
...

public void commandAction(Command cmd,
                          Displayable d) {
    if (cmd == okCommand) {
        int i = (List)d.getSelectedIndex();
        // Use the index of selected list element...
    } else if (cmd == backCommand) {
        // handle the back command
    }
}
```

Implicit List Code

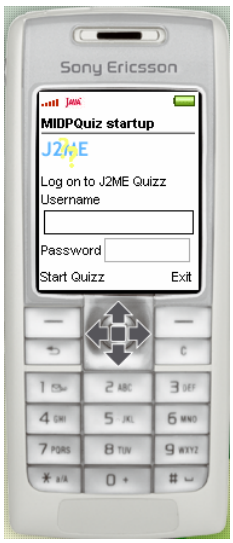

```
List list = new List(null,
                     Choice.IMPLICIT);
list.append("Canvas", null);
list.append("Form", null);
list.append("Alert", null);
list.append("TextBox", null);
list.append("Exclusive List", null);
list.append("Multiple Choice", null);
list.setCommandListener(this);
list.addCommand(exitCommand);
  .
  .
  .
public void commandAction(Command cmd, Displayable d) {
    if (cmd == Command.SELECT_COMMAND) {
        int i = (List)d.getSelectedIndex();
        // Act on the item selected
    } else if (cmd == ...) {
        // Check for and handle other commands
    }
}
```


Multiple List Code

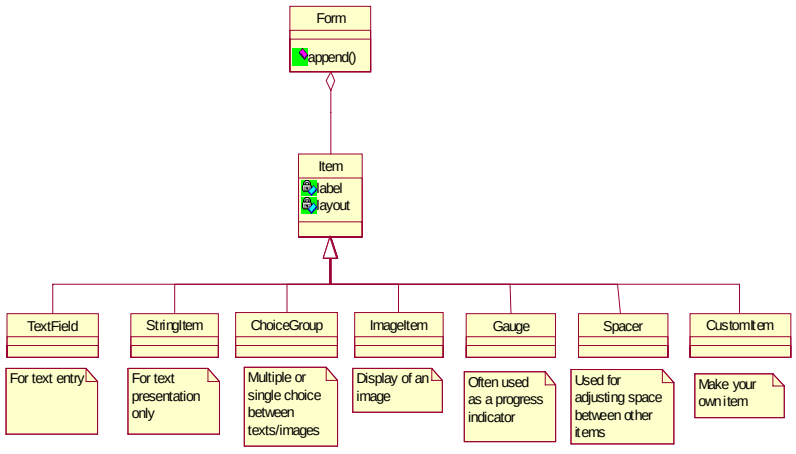

```
List list = new List("Colors to mix",
                     Choice.MULTIPLE);
list.append("Red", null);
list.append("Green", null);
list.append("Blue", null);
text.addCommand(backCommand);
text.addCommand(okCommand);
...

public void commandAction(Command cmd,
                          Displayable d) {
    if (cmd == okCommand) {
        List list = (List)d;
        for (int i = 0; i < list.size(); i++) {
            boolean selected = list.isSelected(i);
            // If selected, take action...
        }
    }
}
```

Form

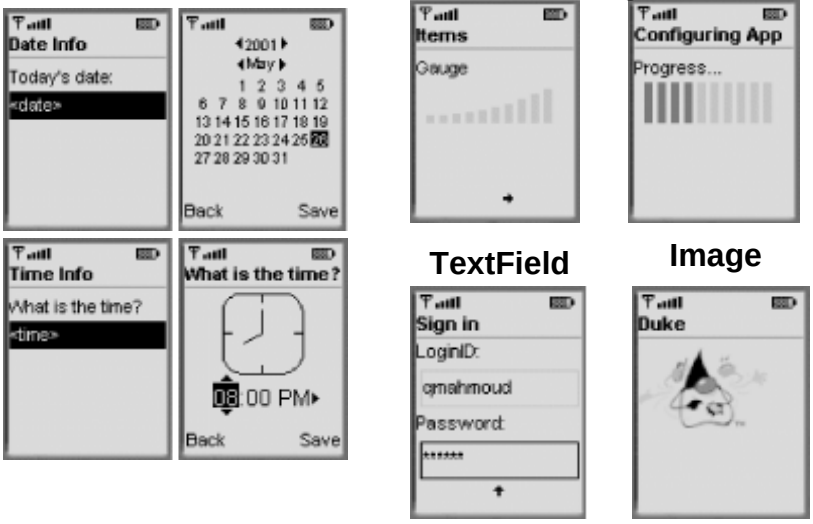
- A form displays an arbitrary number of Items
- Form is used in cases where a screen with single function is not sufficient
- Items can be:
  - **`TextField`** for flexible text entry
  - **`StringItem`** for text display
  - **`ImageItem`** for image display
  - **`DateField`** for date display and entry
  - **`ChoiceGroup`** selectable list of text and/or images
  - **`Gauge`** a bar graph display of a numeric quantity
  - **`CustomItem`** an item you can develop
  - **`Spacer`**, a layout assistance item
- Commands can be connected to Items
- If form too high for display, it will scroll
- What Items where used on the form shown?
  - `ImageItem`, `StringItem` and two `Textfields`



High level UI – Form An Overview



Form Items

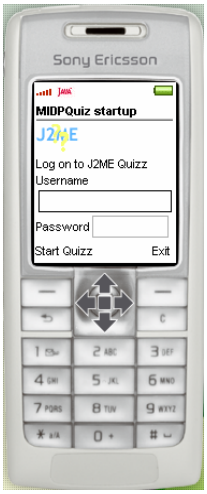


From: Q. H. Mahmoud, *Learning Wireless Java*, O'Reilly, Inc., 2001 44

Form Code

```
35          introForm=new Form("MIDPQuiz startup");
36          Image logo=null;
37          try{
38              logo=Image.createImage("/quizz/images/logo.png");
39          }catch(IOException ioe){
40              logo=null;
41          }
42          ImageItem im=new ImageItem(null,logo,ImageItem.LAYOUT_RIGHT,"? J2ME");
43          introForm.append(im);
44          StringItem stri=new StringItem("","\nLog on to J2ME Quizz");
45          introForm.append(stri);
46          uname=new TextField("Username","",15,TextField.ANY);
47          introForm.append(uname);
48          passwd=new TextField("Password","",15,TextField.PASSWORD);
49          introForm.append(passwd);
50          startCommand=new Command("Start Quizz",Command.OK,2);
51          introForm.addCommand(startCommand);
52          exitCommand=new Command("Exit",Command.EXIT,3);
53          introForm.addCommand(exitCommand);
54          introForm.setCommandListener(this);
55          display.setCurrent(introForm);
56          return introForm;
57      }
58
59      public void commandAction(Command c, Displayable d) {
60          String commandstr=null;
61          String username=uname.getString();
62          mainMIDlet.username=username;
63          String password=passwd.getString();
```

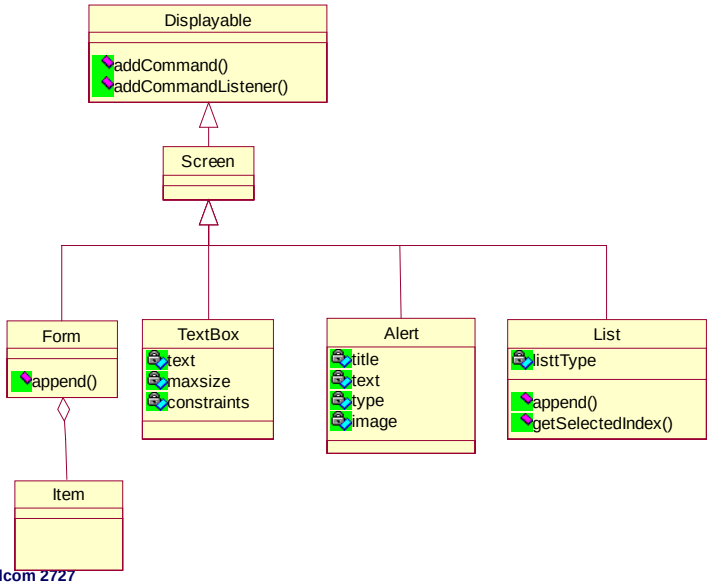
One Form, different devices, different results



Find five differences!

Where can you find "Start Quiz" on the Nokia device?

High Level Interface – A Summary



Guidelines for UI Design

- HCI on mobile and wireless devices a challenge!
  - Why?
- Use simple forms with few Items
  - Remember: Small screen size, mobile context of use
  - Can you avoid scrolling?
- Minimize input, use RMS to store user settings
- Use static form content
  - Small screen means small changes not easy to spot
- Use uniform appearance for easy learning
- Always provide commands like OK, BACK and EXIT on screen
  - use uniform command configuration on all forms to ease learning and use
  - Use soft keys rather than on-screen keys
- Use Threads and Gauges to avoid UI-blocking during network IO
  - improves user experience

Canvas

- Canvas is a displayable that is completely controlled by program
- An application can combine canvases and screens
  - A list to choose racing track
  - A canvas to do the racing
- Methods to handle game actions and pointer events
- A Canvas must be subclassed!
- `paint` method for painting must be implemented
- The paint method provides a Graphics object having methods for painting

Canvas with Graphics

```
import javax.microedition.lcdui.*;
class CanvasDemo extends Canvas {
  protected void paint(Graphics g){
    g.drawString("Canvas Demo",1,1,Graphics.TOP|Graphics.LEFT);
    g.fillRect (20,30,30,20);
    g.drawLine(50,50,75,50);
    g.drawLine(75,50,75,75);
    g.drawLine(75,75,50,75);
    g.drawLine(50,75,50,50);
  }
}
```

1. Displaying a string at position 1,1
2. Drawing a filled rectangle
3. Drawing a rectangle using four lines

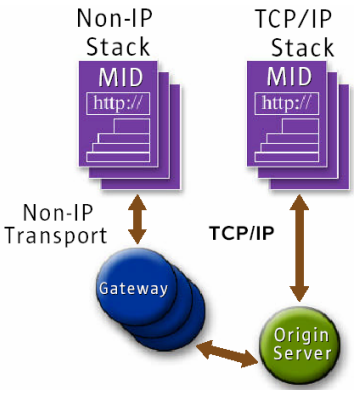

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class CanvasMidlet extends MIDlet{

  Display d;
  Canvas c;

  protected void startApp() {
    d = Display.getDisplay(this);
    c = new CanvasDemo();
    d.setCurrent(c);
  }

  protected void pauseApp() {
  }

  protected void destroyApp(boolean unconditional) {
  }
}
```

1. Creating an instance of the Canvas object
2. Setting the current Displayable to the Canvas object

From J. White and D. Hemphill, *Java 2 Micro Edition*, Manning Publications Co., 2002.

MIDP 1.0 Communication

- Use a standard framework called Generic Connection Framework (GCF) to connect to the Internet, a socket, a file, or etc.
- Device must implement a HTTP 1.1 client interface
- Transport is not necessarily over TCP/IP



Source: Bill Day J2ME

Networking using GCF

- provide a more compact solution, than the standard libraries
  - In GCF all protocols can be specified using an URL-approach:
    `Connector.open("<protocol>:<address>;<parameters>")`
  - But no network protocol is mandatory in CLDC
  - What protocol can actually be used depend on the device and the profile
  - Examples

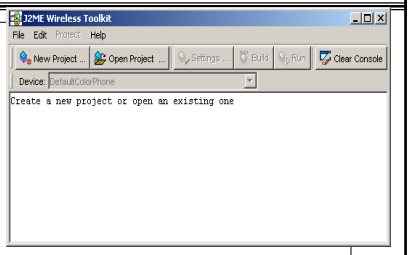
```
Connector.open("http://www.sis.pitt.edu/mwap/test.html")
Connector.open("file:/pictures/picture12.jpg")
Connector.open("comm:0;baudrate=9600;parity=8N1")
Connector.open(“socket://localhost:8080”)
```


GCF Code Sample

```
try {
  InputStream is = Connector.openInputStream("socket://127.0.0.1:8888");
  int ch;
  while ((ch = in.read()) > 0) {
    //do something with the data read
  }
  is.close();
} catch (IOException x) {
    //Handle Exception
}
```

- Use `openInputStream` (from java.io package) to receive data stream
- Only a subset of java.io stream classes of J2SE is available for J2ME
  - ByteArrayInput/OutputStream
  - DataInput/OutputStream
  - PrintStream

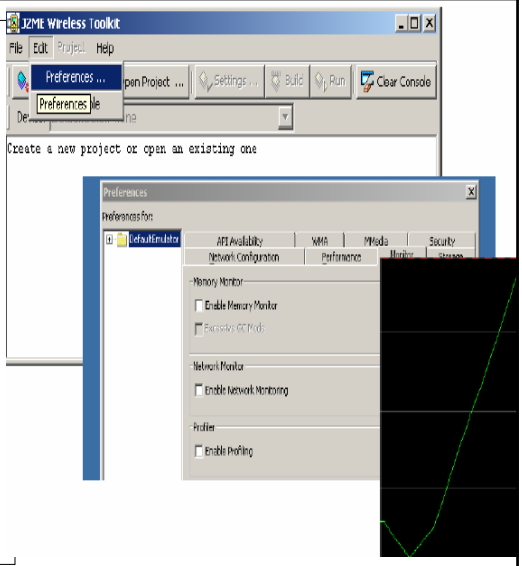
J2ME Development



- Steps in software development
  - edit (code)
  - compile
  - preverify,
  - emulate
  - test on device
  - deploy
- Several tools to help process J2ME Wireless Toolkit (WTK) – compiles, verifies, emulates, can monitor performance
- Most manufacturers have SDKs that build on WTK (Nokia, Sony/Ericcson, Motorola, etc.) see class web page for links
- Sun has Wireless Blueprints for common J2ME applications
  - Games, End-to-End, etc.

MIDP Development with J2ME WTK

1. Write your Java application (midlets) using any text editor or IDE (e.g., netbeans, JEDIT, etc.)
2. Use J2ME Wireless Toolkit (WTK), Create a Project
3. Click on Build
4. Run against a built in emulator.



MIDP Development with J2ME WTK

- Wireless Toolkit (WTK) Features
  - Can incorporate specific device emulators from manufacturer
  - Automatic JAD verification
  - Can monitor performance
  - Wireless Messaging APIs
  - Mobile Media APIs
  - Integrated Over the Air emulation
  - Integrated Security Tools
  - Push emulation

OTA Development with J2ME WTK

Select Project | Run via OTA

In the emulator, select Apps | Install Application (Notice in the emulator that the WTK has generated an HTML page that references the JAD file)

Select Menu | Go

Emulator shows the JAD file to install (next page)

OTA Development with J2ME WTK

OTA Development with J2ME WTK