Wireless Online Game Development for Mobile Device

Lesson 1

Who am I?

Lo Chi Wing, Peter

- Email: Peter@Peter-Lo.com

Are you take the right course?

- This course is intended for experienced Java programmers who would like to learn how to develop mobile game using J2ME.
- The code examples are not particularly complex, but it is assumed that you understand how classes are created within the Java platform.

What games can you make after this course?

Where can you find the material?

- Workshop Notes and Exercises
  - http://www.Peter-Lo.com/Teaching/I123-1-A/
- J2ME Document
  - http://java.sun.com/j2me/

Class Exercise

- Who are you?
- Why you apply this course?
- Have you learn programming such as Java?
- What type of game you like to develop (Action, RPG, SLG, War game…)?
- What is your expectation for this course?

Part 1

Introduction to J2ME

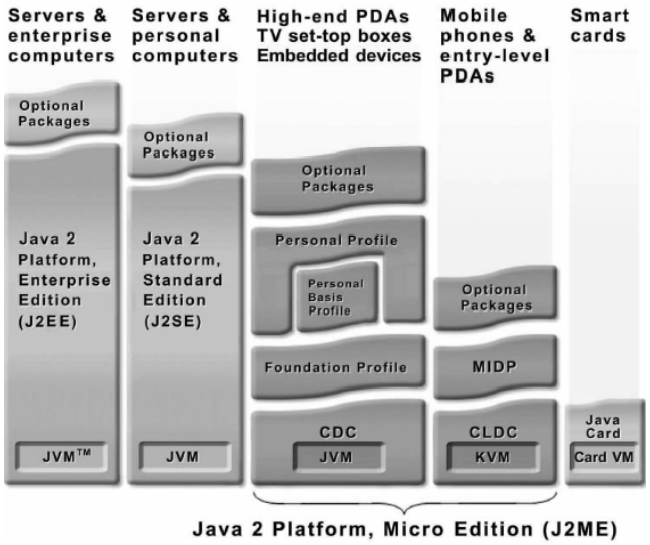
What is J2ME?

- As we enter the post-PC era, the market for network-connected consumer devices such as smart mobile phones, televisions and PDA is expected to grow at a phenomenal rate.
- In order to provide a compelling Java technology solution for these devices, Sun has introduced the latest addition to the Java platform: Java 2 Micro Edition (J2ME).

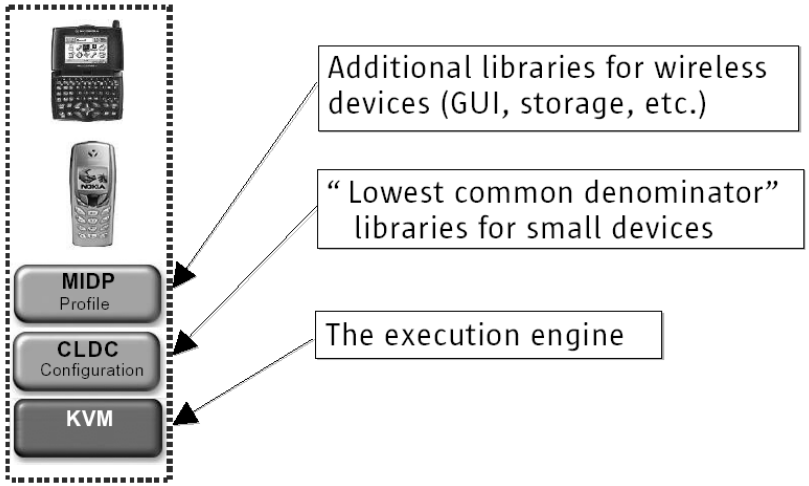
What is J2ME?

- In J2ME, the Java Runtime Environment (JRE) is adapted for devices that have limitations on what they can do when compared to standard desktop or server computers.
- For low-end devices, the constraints are fairly obvious:
  - Extremely Limited Memory
  - Small Screen Sizes
  - Alternative Input Methods
  - Slow Processors

Java Platform



Key Components of the J2ME Technology Stack



J2ME Technology

- J2ME is divided in Configurations and Profiles:
  - A **Configuration** defines the minimum Java technology that an application developer can expect on a broad range of implementing devices.
  - A **Profile** defines the set of APIs available for a particular family of devices such as cell phones, PDA, microwave ovens, etc.

J2ME Configuration

- A configuration is a specification for a certain class of device
  - Usually based on available memory and processing power
- Specifies a JVM
  - The JVM can be ported to the devices encompassed by the configuration
- Specifies a subset of J2SE APIs that will be available

Common Configuration

- J2ME Connected, Limited Device Configuration (CLDC)
  - Specifies Java environment for mobile phone, pager and PDA class devices
  - CLDC devices are usually wireless
- J2ME Connected Device Configuration (CDC)
  - Specifies Java environment for digital television set-top boxes, high end wireless devices and automotive telemetric systems.
  - CDC devices may be wired (DTV cable, etc.)

J2ME Profiles

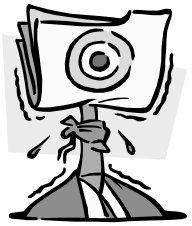
- Build on configurations
- More specific than configurations
  - Can specify available input and output facilities
- Adds APIs for user interface, persistent storage, whatever else is needed to run applications on a class of devices

Common Profiles

- J2ME Mobile Information Device Profile (MIDP)
  - Application runtime environment for wireless devices based upon CLDC
- J2ME Foundation Profile
  - Base profile for non-GUI networked devices based upon CDC
- J2ME Personal Basis, Personal, and RMI Profiles
  - Basic graphics and RMI support for CDC & Foundation Profile based devices

What will this course focus?

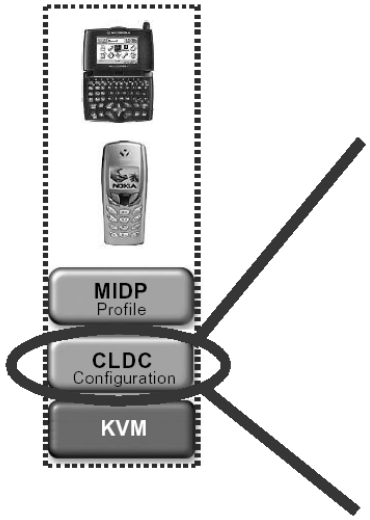
- This course focuses on using the **MIDP** and **CLDC** to create an application for a cell phone.



Part 2

Introduction to CLDC and MIDP

CLDC Scope



- Core JLS & JVMS features
- Core java.* libraries
- Additional I/O and networking libs
- Security features
- Internationalization

What devices CLDC target?

- 160KB to 512KB total memory available for Java technology
- With an intermittent network connection
- Limited power
- Small screen/User Interface

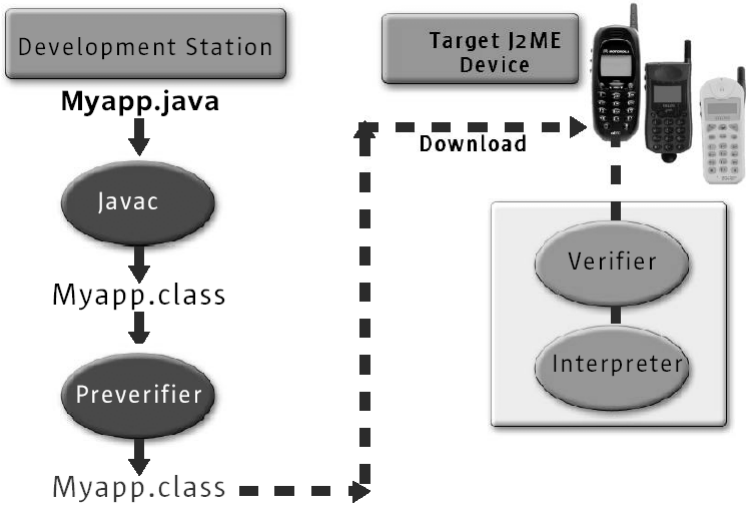
Beyond the CLDC Scope

- Specified in Profiles to be implemented on top of the CLDC:
  - User interface support
  - Event handling
  - High-level application model
  - Persistence support
- Example specification of these remaining APIs: MIDP

CLDC Libraries

- Classes inherited from Java 2 Platform, Standard Edition (J2SE) are in packages:
  - **java.lang.***
  - **java.util.***
  - **java.io.***
- Classes introduced by CLDC are in package:
  - **javax.microedition.***

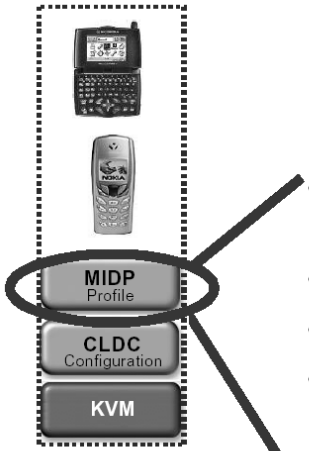
CLDC Classfile Verification



Preverification and Application Representation

- Classfiles must be preverified
  - You will receive a runtime error if you attempt to execute an application which has not been preverified
- Whenever a Java application intended for a CLDC device is stored publicly, the JAR format must be used
  - Alternate formats may be used internally

MIDP Scope



- Application model (lifecycle, packaging
- Persistent storage (RMS APIs)
- Networking (HTTP, etc.)
- User interface (High and low level APIs)

What devices MIDP target?

- MIDP is more specific than CLDC about device requirements:
  - 128 kB for the MIDP implementation
  - 32 kB for the runtime heap
  - 8 kB for persistent data
  - Minimum screen: 96 x 54, 1 bit color
  - Some method of input
  - Two-way network connection (intermittent)

Beyond the MIDP Scope

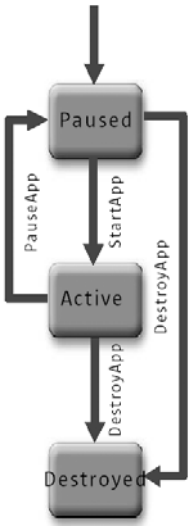
- To be implemented by device manufacturers, operators, or third party developers:
  - How an application actually gets on the device ("provisioning")
  - The end-to-end security model
  - System- or OEM-specific technologies
- Refer to MIDP 2.0 for more on provisioning and security

Part 3

MIDlet Life Cycle

The MIDlets Life Cycle

- MIDP applications, or “MIDlets”, move from state to state in their lifecycle according to a state diagram
  - **Paused** – initialized and waiting
  - **Active** – has resources and is executing
  - **Destroyed** – has released all resources, destroyed threads, and ended all activity



notifyDestroyed()

- Used by an MIDlet to notify the application management software that it has entered into the Destroyed state.
- The application management software will not call the MIDlet's destroyApp method, and all resources held by the MIDlet will be considered eligible for reclamation.
- The MIDlet must have performed the same operations (clean up, releasing of resources etc…) it would have if the ***MIDlet.destroyApp()*** had been called.

destroyApp()

- Signals the MIDlet to terminate and enter the Destroyed state.
- In the destroyed state the MIDlet must release all resources and save any persistent state.
- This method may be called from the Paused or Active states.
- MIDlets should perform any operations required before being terminated, such as releasing resources or saving preferences or state.

notifyPaused()

- Notifies the application management software that the MIDlet does not want to be active and has entered the Paused state.
- It may be invoked by the MIDlet when it is in the Active state (Invoking this method will have no effect if the MIDlet is destroyed, or if it has not yet been started)
- If a MIDlet calls notifyPaused(), in the future its startApp() method may be called make it active again, or its destroyApp() method may be called to request it to destroy itself.
- If the application pauses itself it will need to call resumeRequest() to request to reenter the active state.

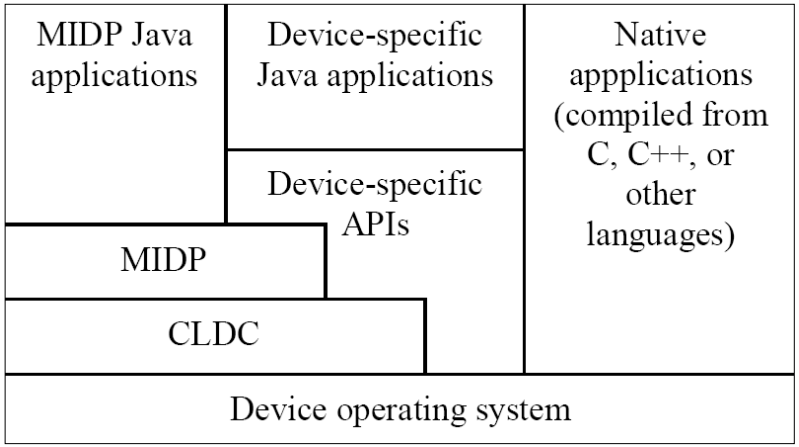
pauseApp()

- Signals the MIDlet to enter the Paused state.
- In the Paused state the MIDlet must release shared resources and become quiescent.
- This method will only be called called when the MIDlet is in the Active state.
- If a Runtime exception occurs during pauseApp the MIDlet will be destroyed immediately.
- Its destroyApp will be called allowing the MIDlet to cleanup.

Part 4

MIDlet Implementation

MIDP Applications

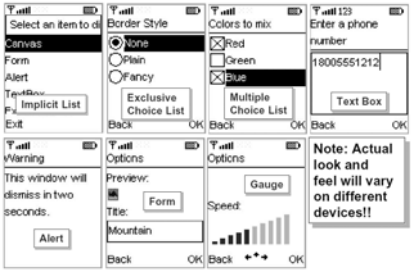


UI Design Principles

- MIDlets must be usable in all devices
  - One handed, two handed, stylus operation
  - Not all devices have a pointing device
- Must constantly think of end users
  - Mobile Information Devices are consumer products, not desktop computers
  - MIDP applications and native apps should look and behave consistently on any given device

High-level UI API

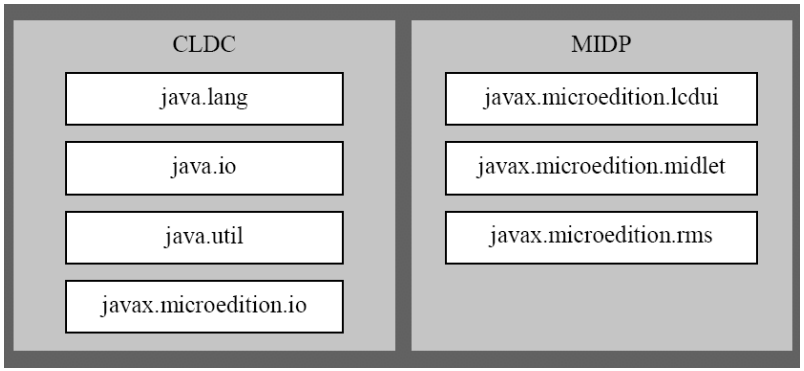
- Use the MIDP High-level UI APIs for portability
  - Apps run in all MIDP compliant devices
  - No direct access to native device features
  - Based upon ***javax.microedition.lcdui.Screen***



Low-level UI API

- Provide access to native drawing primitives, device key events, native input devices, etc.
- Allow developers to choose to compromise portability for user experience, if required
- Handle low level events and drive graphics via ***javax.microedition.lcdui.Canvas***

Implementation Model



javax.microedition.lcdui

- The User Interface API
- Provides a set of features for implementation of UI
- The central abstraction of the MIDP's UI is a screen.
- A screen is an object that encapsulates device-specific graphics rendering user input.
- Only one screen may be visible at the time, and the user can only traverse through the items on that screen.
- The screen takes care of all events that occur as the user navigates in the screen, with only higher-level events being passed on to the application.

javax.microedition.midlet

- The MIDlet package defines MIDP applications and the interactions between the application and the environment in which the application runs.
- Core API of MIDP
- An application of the Mobile Information Device Profile is a MIDlet.

javax.microedition.rms

- The MIDP provides a mechanism for MIDlets to persistently store data and later retrieve it.
- This persistent storage mechanism is modeled after a simple record oriented database and is called the **Record Management System**.

Part 5

MIDlet Development

Development Tools

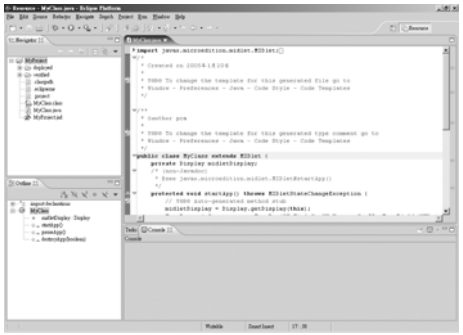
- J2ME Wireless Toolkit
- Nokia Developer's Suite for J2ME
- Sony Ericsson J2ME SDK
- Motorola iDEN SDK for J2ME Technology

MIDlet Development Steps

1. Create source code for your J2ME application
2. Compile it
3. Preverify it
4. Package it into a JAR file
5. Create the application descriptor
6. Deploy and run your application in the J2ME Wireless Toolkit or your device of choice

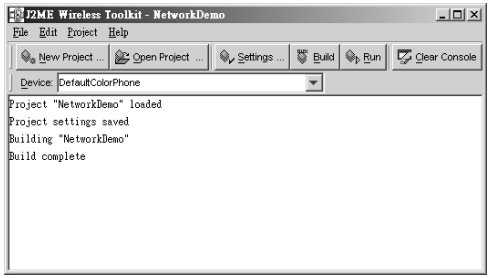
Creating Source Code

- Use the development kit of your choice
- Or use the text editor of your choice



Compiling Source Code

- Use J2SE tools to create classes for MIDP

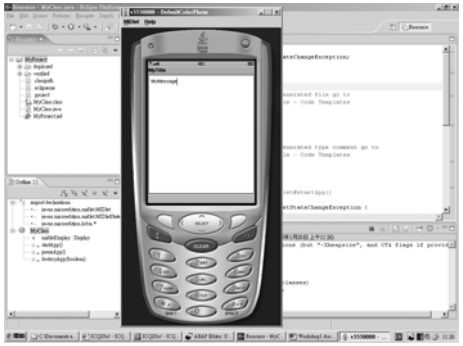


Preverifying

- MIDP's bytecode verifier is much smaller than the J2SE verifier
- Bytecode verification is split into two parts:
  - The first part is performed off the device
  - The second part is performed on the device
- Use the preverifier supplied with the development kit.

Using an Emulator

- Most development tools go along with an emulator for testing and debug



Application Packaging

- MIDlets are packaged in a JAR file including
  - Class files of the MIDlet(s)
  - Resource files
  - Manifest with application properties
- Application Descriptors (JAD files) accompany MIDlet JARs and provide deployment information (name, version, size, etc.)

Deploying on a Real Device

- Devices are assumed to have some kind of application manager
  - Installs new applications, via the Internet, a serial cable, IR, or some other means
  - Manages applications in the device's memory
  - Launches applications as needed

Sample JEME Program

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

public class ClassName extends MIDlet {
  public ClassName() {
    ...
  }
  public void startApp() {
    ......
  }
  public void pauseApp() {
  }
  public void destroyApp(boolean unconditional) {
 }
}
```

Part 6

Considerations for Mobile Game Development

Programming Approach

- Write your game in the simplest manner:
  - Separating tasks into several objects and heavily rely on patterns provides maintainability and extensibility
  - It is best to avoid using several objects unless truly necessary where memory is expensive in the mobile world
  - Merge classes that do not provide much gain in functionality.

Memory Limitations

- Three types of memory need to concern:
  - **Working Memory** is the memory where the game is executed during runtime. If the game is too big, an out of memory error will occur.
  - **Storage Memory** is used to store necessary information (high scores, …) for the game.
  - **Application Memory** is the memory needed to hold all the games and applications in the mobile.

Memory Limitations (cont')

- After knowing what memory constraints, consult the manufacturers specification for the mobile handsets that are planning to deploy the game to.
- Check for the maximum size a game can be during runtime and memory storage available.

Screen Limitation

- Displays on J2ME enabled mobile handsets vary not only from manufacturer to manufacturer; but also from model to model produced by the same manufacturer.
- Things to consider when adjusting the display of the mobile game are screen size, frame rate and color.
- Make a decision which handsets want to support.
- It is ideal to write the game generically then make alternate versions for certain handsets.
- Consult the manufacturers specification and make the appropriate adjustments.

Some Solutions to Improve Performance

- Use the least amount of classes, stick to shorter variable, method and class names.
- Try to reuse objects rather then instantiating new objects over and over again.
- If it is a network game then reduce calls to the server
- Try to obtain all resources in one request at the beginning rather then several requests to reduce the chance of latency problems to occur.
- Optimize the graphics and merge the graphics into one sheet instead of spreading them out into several separate graphics.

Limitation of KVM

- Because the KVM has to be so much smaller than a J2SE JVM, some features are missing:
  - No native methods
  - Limited bytecode verifier
  - No floating point support (i.e. no float or double)
  - No object finalization

How to find Help for API?

- The document for MIDP API is located in "C:\WTK22\docs\api"