Java 2 Micro Edition

F. Ricci

Content

- Why Java on mobile devices
- Three main Java environments
- Java 2 Micro Edition
- Configurations and profiles
- Optional packages
- Generic connection framework
- Application manager and MIDP applications
- Sun Java Wireless Toolkit
- Two application examples
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** (Simbian 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.
Java Editions

- The Java 2 Platform is split into three editions:
  - **Java 2 Standard Edition (J2SE)** - Desktop-based applications
  - **Java 2 Enterprise Edition (J2EE)** - Server-based applications
  - **Java 2 Micro Edition (J2ME)** - For handheld and embedded devices

- Each edition provides a complete environment for running Java-based applications including the Java Virtual Machine (JVM) and runtime classes
- What separates one edition from another, then, is primarily the set of class libraries that each edition defines
- you can think of J2ME as a subset of J2SE and J2SE as a subset of J2EE.
J2ME

- J2ME aims at providing a comprehensive application development platform for creating dynamically extensible, highly portable, small-footprint, networked devices and applications for the consumer and embedded market.

- Two broad categories of products:
  - **High end consumer devices**: (CDC) e.g., TV set-top boxes, Internet TVs, Internet enables screen phones, automotive entertainment/navigation systems
    - Large user interface, total memory starting from 2Mb, persistent and high-bandwidth network connections (TCP/IP)
  - **Low-end consumer devices**: (CLDC) cell phones, pagers, personal organizers
    - Simple user interface, memory starting from 128-256 Kb, low bandwidth, intermittent network connection, battery operated.
J2ME require flexibility

- **Different device types** and hardware configurations have different requirements
- Different **usage models** employed by the devices (key, stylus, voice)
- Constantly improving device technology
- Diverse **range of existing applications** and features
- Users/developers want flexibility: they want to choose what they want to use and what they don’t
- *Hence: J2ME provides a range of Java Virtual machine technologies, optimized for different processor types and memory footprints.*

Independence of Network Technology Standards

- There is a plethora of wireless technologies in use with varying levels of sophistication, compatibility and interoperability (GSM, TDMA, CDMA, WCDMA, UMTS, GPRS, EDGE, …)
- J2ME standardization efforts is to define **solutions that can work effectively with all these** network technologies and standards
- J2ME should be used both in today’s low-speed wireless networks and in tomorrow’s high-speed broadband wireless networks
- CLDC defines a **Generic Connection Framework**
Today’s small wireless devices (cell phones and two-way pages) have microprocessors that are similar to those used in desktop computers less than 15 years ago!

**BUT**

- Moore’s law does not apply to the battery: *(the transistor density of integrated circuits doubles every 24 months)* – low power consumption is very important – almost 80% of the power is reserved for radio transmission
- High volume production: to improve profit device manufacturer want to keep the per-unit cost of the devices as small as possible
- Specialized nature of devices: cell phones are highly customized for phone voice communication and manufacturers will add features only if justified by the market.

Here **one solution does not fit all**: consumer devices are highly specialized for the intended use

*The performance of a consumer device is not measured by the computing power but how well it serves the intended usage*

Factors differentiating consumer devices from desktop computers

- Small screen size
- Different usage models: stylus, tiny keypad, small QWERTY keyboards, voice operated
- Mobility: in traffic, while skiing, etc.
- Limited network bandwidth with intermittent connections.
A configuration is a complete Java runtime environment:

- Java virtual machine (VM) to execute Java
- Set of core Java runtime classes
- Interface to the underlying system

- Defines a minimum platform for a “horizontal” category or grouping of devices with similar requirements on memory and processing power

- A J2ME application is written for a particular profile and a profile is based upon or extends a particular configuration.
Configuration

- There are 2 basic configurations: **CDC** (Connected Device Configuration) and **CLDC** (Connected Limited Device Configuration)

**CDC** (Connected Device Configuration): high-end consumer devices (TV set-top boxes, Internet TV)
- 512KB of read-only-memory (ROM), 256 KB of random access memory (RAM), minimum
- 32-bit processor
- High bandwidth network connection
- Full-featured Java2 virtual machine (CVM)
- 17 packages
- Use for devices like Palms.

**Configuration: CLDC**

- **CLDC** (Connected Limited Device Configuration): low-end consumer devices - Cell phones, two-way pagers, personal digital assistants (PDAs), organizers, home appliances, and point of sale terminals
  - 160 - 512 KB of total memory (160KB ROM and 32KB RAM, minimum)
  - 16-bit or 32-bit processor
  - Low power consumption and often operating with battery power
  - Connectivity with limited bandwidth
  - Selected classes from:
    - java.lang, java.io, java.util
  - Limited VM (called **KVM**) without:
    - Floating point types (in CLDC 1.0)
    - Object finalization
    - JNI or reflection
    - Thread groups or daemon threads
    - User Class loaders
Portability

- **Portability** is a hard requirement in consumer space
  - Consumer devices differ substantially in memory size, networking, user interface capabilities
- Hence, **portability in device families** is what is required: cellular telephones, washing machines, electronic toys
  - A *stock trading application is not supposed to work on an electronic toy*
  - *But should work on different cellular phones (from different manufacturers)*
- It is important to deliver right functionality at the right price!
- A single device can support several profiles
- A profile is a contract between an application and a vertical market.
Profile and Optional Packages

- The **profile** adds classes to a configuration:
  - To fill in missing functionality
  - To support specific uses of a device
  - To address the specific demands of a vertical market sector, e.g., cellular telephones, washing machines, electronic toys
  - The only one in existence is MIDP (cell phones)
- The **Optional Packages** are set of APIs that support additional and common behaviors
  - Examples of optional packages:
    - Bluetooth Optional Package
    - JDBC Optional Package
    - File connection
    - Personal Information Management (PIM)
    - ...

Profiles

- Several profiles in various stages of development:
  - **Mobile Information Device Profile (MIDP)** - CLDC-based, used for running applications on cell phones and interactive pagers with small screens, wireless HTTP connectivity, and limited memory
  - **Personal Digital Assistant Profile (PDAP)** - CLDC-based, extends MIDP with additional classes and features for more powerful handheld devices (uses a subset of Abstract Windowing Toolkit AWT)
  - **Foundation Profile (FP)** - CDC-based, extends the CDC with additional J2SE classes
  - **Personal Basis Profile (PBP)** - extends the FP with lightweight (AWT-derived) user interface classes and a new application model
  - **Personal Profile** extends the PBP with applet support and heavyweight UI classes
- Check on [http://jcp.org/jsr/detail](http://jcp.org/jsr/detail) the state of these specifications
Optional Packages for the Wireless Market

- JSR 120: Wireless Messaging API
- JSR 135: Mobile media API
- JSR 172: J2ME Web Services Specification
- JSR 177: Security and Trust Services Specification
- JSR 179: Location API for J2ME
- JSR 180: Session initiation Protocol (SIP) for J2ME
- JSR 184: Mobile 3D Graphics for J2ME
- JSR 190: Event Tracking API for J2ME

Hardware Requirements: MIDP

- **Memory:**
  - 256Kb non-volatile for MIDP components (in addition to the requirements of CLDC),
  - 8Kb non-volatile for application created persistent data,
  - 128 Kb volatile for virtual machine run time
- **Display:** 96x54, depth 1-bit, pixel shape 1:1
- **Input:** one or two-handed keypad, touch screen
- **Networking:** two-way, intermittent, with limited bandwidth
- **Sound:** play tones.
Software Requirements: MIDP

- **Minimal kernel** to manage the underlying hardware (interrupts, exceptions, and minimal scheduling)
- Mechanism for **reading and writing from non-volatile memory** (to support persistence API)
- **Read and write access to devices’ wireless networking** (to support networking API)
- A mechanism to **time-stamping the records** written in the persistence storage
- Support to write a **bit-mapped graphic display**
- Mechanism to **capture user input** from keypad or touch screen.

Low-level security

- **Low-level security** (virtual machine security): ensure that the application running in the JVM follows the semantic of the java prog. language (malicious classes must not harm the device)
- **Class file verifier** ensures that the bytecode:
  - cannot contain illegal instructions,
  - cannot be executed in an illegal order, and
  - cannot contain references to invalid memory locations
- *The class files loaded in the virtual machine must execute what is allowed by the JVM Specification.*
Application Security

- **Application security**: Java application running on the device can access only those libraries, system resources, and components that the device and Java environment allow to access.
- There is a closed „sandbox“
  - Class files must be properly verified
  - The downloading, installation, and management of the application is such that the programmer cannot modify the standard class loading mechanism
  - A closed set of Java APIs is available
  - The application programmer cannot download any new libraries containing native functionality not part of the specifications.

Application Management

- The **Application Manager** is a piece of device-specific software with the ability to:
  - **Download** and **install** Java applications
  - **Inspect** existing Java applications stored on the device
  - **Select** and **launch** Java applications
  - **Delete** existing applications
- A CLDC system may allow multiple Java applications to executes concurrently or restrict to one application at a time
- Application management is up to MIDP mostly.
CLDC 1.1 and MIDP 2.0 packages

### MIDP 2.0
- `javax.microedition.lcdui`
- `javax.microedition.lcdui.game`
- `javax.microedition.media`
- `javax.microedition.media.control`
- `javax.microedition.midlet`
- `javax.microedition.pki`
- `javax.microedition.rms`

### CLDC 1.1
- `java.lang`
- `java.lang.ref`
- `java.io`
- `java.util`
- `javax.microedition.io`

### Devices Evolution (Nokia)

<table>
<thead>
<tr>
<th>Device</th>
<th>Year</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>6600</td>
<td>2003</td>
<td>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)</td>
</tr>
<tr>
<td>N70</td>
<td>2005</td>
<td>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)</td>
</tr>
<tr>
<td>N95</td>
<td>2007</td>
<td>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)</td>
</tr>
</tbody>
</table>
What device?

- If you want to know what devices support what profile/configuration/package ...

CLDC 1.1. Class Library

- Classes that are a subset of standard J2SE:
  - `java.lang.*`, `java.util.*`, `java.io.*`, `java.lang.ref`
  - A class with the **same name** and package name as a J2SE class must be **identical to** or a **subset** of the corresponding J2SE class
  - The classes **cannot add** any public or protected methods or fields that are not available in J2SE
- Classes that are specific to CLDC
  - `javax.microedition.io`
CLDC Classes (subset of J2SE)

- **System Classes**
  - `java.lang.Object`
  - `java.lang.Class`
  - `java.lang.Runtime`
  - `java.lang.Thread`
  - `java.lang.runnable` (interface)
  - `java.lang.String`
  - `java.lang.StringBuffer`
  - `java.lang.Throwable`
  - `java.lang.Float`
  - `java.lang.Double`

- **Data Types Classes**
  - `java.lang.Boolean`
  - `java.lang.Byte`
  - `java.lang.Short`
  - `java.lang.Integer`
  - `java.lang.Long`

- **Collection Classes**
  - `java.util.Vector`
  - `java.util.Stack`
  - `java.util.Hashtable`
  - `java.util.Enumeration` (interface)

- **IO Classes**
  - `java.io.InputStream`
  - `java.io.OutputStream`
  - `java.io.ByteArrayInputStream`
  - `java.io.ByteArrayOutputStream`
  - `java.io.DataInput` (interface)
  - `java.io.DataOutput` (interface)
  - `java.io.DataInputStream`
  - `java.io.DataOutputStream`
  - `java.io.Reader`
  - `java.io.Writer`
  - `java.io.InputStreamReader`
  - `java.io.OutputStreamWriter`
  - `java.io.PrintStream`

- **Calendar and Time Classes**
  - `java.util.Calendar`
  - `java.util.Date`
  - `java.util.TimeZone`

- **Utility classes**
  - `java.util.Random`
  - `java.lang.Math`

- **Exception and Error classes**
  - See the specification!
The package `java.net` of JDK contains **31 classes** and interfaces and **8 exception** classes.

It is **difficult to make all this functionality fit in a small device** with only few hundreds Kbs of memory.

A significant part of standard I/O and networking is not directly applicable to small devices.

Requirements for networking depends on the device:

- **Packet-switched** networks want datagram-based communication.
- **Circuit-switched** requires stream-based connections.

### Connection Interface Hierarchy

- **Connection**
  - **InputConnection**
  - **OutputConnection**
  - **DatagramConnection**
    - **StreamConnection**
    - **UPDDatagramConnection**
  - **CommConnection**
  - **ContentConnection**
  - **SocketConnection**
  - **StreamConnectionNotifier**
    - **HttpConnection**
    - **SecureConnection**
    - **ServerSocketConnection**
    - **HttpsConnection**
Generic Connection Framework

- **General form**
  - `Connector.open("<protocol>:<address>;<parameters>")`

- **HTTP**
  - `Connector.open("http://www.sun.com")`

- **Sockets**
  - `Connector.open("socket://129.144.111.222:2800")`

- **Communication port**
  - `Connector.open("comm:comm0,baudrate=9600")`

- **Datagrams**
  - `Connector.open("datagram://129.144.111.222:2800")`

- These calls will return an object that implements one of `javax.microedition.io.Connection` interface (see previous slide)

- Hence a binding of a protocol in J2ME can be done at runtime!

MIDlets – The heart of J2ME

- MIDP does not run in the “regular” Java fashion using: `main()`, `System.exit()`

- Instead, we use **MIDlet** applications - which are subclasses of `javax.microedition.midlet.MIDlet`

- The application must extend this class to allow the **application management** software to control the MIDlet:
  - control the MIDlet installation
  - be able to retrieve properties from the application descriptor
  - respond to a request for state change

- MIDlets are installed moving its class files to a device

- The class files are packaged in a Java Archive (JAR), and an accompanying descriptor file (.jad extension) describes the contents of the JAR.
MIDP Application Lifecycle

- **MIDlets** move from state to state in the lifecycle – it is the application manager that changes its state:
  - **Pause**
    - After the constructor is called or,
    - `pauseApp()` called by AM or,
    - The midlet has called a `notifyPaused()`
  - **Active**
    - The AM has called `startApp()`
    - The midlet has called `resumeRequest()`
  - **Destroyed**
    - The AM has called `destroyApp()`
    - The midlet has called `notifyDestroyed()`.

MIDlet Suite

- One or more MIDlets are packaged together into a **MIDlet suite**, composed of:
  - **JAR (Java archive) file**
    - Contains **Java classes** for each MIDlet in the suite and Java classes that are shared between MIDlets
    - Contains **resource files** (e.g. an image) used by the MIDlets and a **manifest file**
  - **JAD (Java Application Descriptor) file**
    - Contains a predefined set of attributes that allows the device application management software to identify, retrieve, and install the MIDlets
  - Eventually the JAR / JAD files are **uploaded to the device** in order to run the application.
**Wireless Development Tutorial Part I**

- **What do we need**
  - **Java Platform, Standard Edition version** 1.4.2 or higher. (version 1.5.0 is now available to download)
  - **Sun Java Wireless Toolkit** This is a package of tools for building and testing MIDlets
  - **Text editor.** This can be something as rudimentary as Notepad (on Windows) or something more elaborate (IDE environment as NetBeans)

- Following example is from

---

**Sun Java Wireless Toolkit**

- Execute the installation file
- The files for the toolkit will go into `c:\WTK25`
- To run the toolkit itself, select the **KToolbar** shortcut
- There is a very good [user guide](http://java.sun.com/products/sjwtoolkit/download-2_5.html)
Project

- The Sun Java Wireless Toolkit works with *projects*, where the end result of each project is one **MIDlet suite**
- Click on New Project in the button bar

HelloWorld MIDlet

```java
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class HelloWorld
    extends MIDlet
    implements CommandListener {
    private Form mMainForm;

    public HelloWorld() {
        mMainForm = new Form("HelloMIDlet");
        mMainForm.append(new StringItem(null, "Hello, MIDP!");
        mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
        mMainForm.setCommandListener(this);
    }

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

    public void pauseApp() {}
    public void destroyApp(boolean unconditional) {}
    public void commandAction(Command c, Displayable s) {
        notifyDestroyed();
    }
}
```
Hello World

- Save this code as HelloMIDlet.java in the src directory of your project
- c:\WTK25\apps\HelloSuite\src\HelloMIDlet.java
- Press BUILD
- The toolkit will attempt to compile your project
- Sun Java Wireless Toolkit has created classes, tmpclasses, and tmplib

---

Running

- Click on the Run button
- You should see a phone emulator
- The emulator is showing a list of MIDlets in the MIDlet suite (here only one)
- the name you see here is HelloSuite, the class that will be run is HelloMIDlet
- To see where this mapping occurs, go to KToolbar and select Settings....
- Back in the emulator, click on the soft button below Launch
Result

- Click on **Exit** to leave the MIDlet
- Close the emulator window or hit the Escape key to end the emulator session

1) Build

- What happens when you press the **Build** button?
- The toolkit finds all the .java files in the src directory of your project and 1) **compiles** them
- Source files must be compiled in a MIDP environment rather than a J2SE 5.0 environment
- For instance a MIDlet that uses the `java.lang.System` class: this class has different APIs in J2SE 5.0 and MIDP
- When the toolkit compiles your MIDlet class it uses the MIDP `java.lang.System`, not J2SE 5.0 version of the class
- You could make this selection yourself (if you installed the MIDP reference implementation), using the command `javac` and the `-bootclasspath` option
  - `javac -bootclasspath hellomidlet.java`
2) Preverifying Class Files

- The toolkit performs an initial verification at build time (preverifying)
  - Certain checks are performed and the class file is modified in such a way that the second-step (runtime) can be easily handled
- The device's runtime system performs a second verification when it loads the classes
  - If a class file has not preverified it is rejected
- You could perform the first verification yourself using the command line `preverify` tool.

3) JARing

- Finally, MIDlets are bundled into MIDlet suites for distribution to actual devices
  - Bundling entails JARing the MIDlet suite class files and the resource files, and putting some extra information in the JAR manifest
  - Finally the files are 4) deployed on the device
- The above steps are not required for running the application in the Wireless Toolkit
- But are required if you want to deploy the midlet suit on a mobile device.
Deploying MIDlets

- MIDlets can be deployed on a phone in two ways:
  - Transfer the jar and jad files to the phone from the computer via an **external connection**: serial cable, USB cable, IRDA, Bluetooth.
  - **Over the Air (OTA) provisioning**: download the midlet suite from a server
- Installation is specific to the device!
- Check the documentation of your device to see how to install a MIDlet suite.

Manifest Information

- Every JAR includes a manifest file `META-INF/MANIFEST.MF`

  MIDlet-1: Hellosuite, Hellosuite.png, HelloMIDlet
  MIDlet-2: HitMIDlet, , HitMIDlet
  MIDlet-Name: Hellosuite
  MIDlet-Vendor: Unknown
  MIDlet-Version: 1.0
  MicroEdition-Configuration: CLDC-1.1
  MicroEdition-Profile: MIDP-2.0

- It describes the content of the archive
- It may contain extra information that is important to the MIDP runtime environment (e.g. a URL to connect).
MIDlet Suite Descriptor

- Before a midlet can be deployed an additional file is required: an application description, a **.jad** file
- The **.jad** file contains a lot of the same information that’s in the **manifest** file
- The application descriptors contains information that help the device and/or the user to decide whether or not to load a MIDlet suite
- It can be downloaded and examined before downloading the **.jar**
- Useful in OTA provisioning – the server returned MIME type for the **.jad** file must be `text/vnd.sun.j2me.app-descriptor`

Hellosuite.jad

- HitMIDlet.URL: http://localhost:8080/midp/hits
- MIDlet-1: Hellosuite, Hellosuite.png, HelloMIDlet
- MIDlet-2: HitMIDlet, , HitMIDlet
- MIDlet-Jar-Size: 3016
- MIDlet-Jar-URL: Hellosuite.jar
- MIDlet-Name: Hellosuite
- MIDlet-Vendor: Unknown
- MIDlet-Version: 1.0
- MicroEdition-Configuration: CLDC-1.1
- MicroEdition-Profile: MIDP-2.0
MIDlet Properties

- Attributes that have meaning in a MIDlet can be added to the manifest and/or the application descriptor files
- It is more convenient to add an attribute to the application descriptor – it may be changed without touching the application files
- If an attribute is listed in both files the value in the application descriptor will be used
- A MIDlet can retrieve the values of these attributes using `getAppProperty()`
- Example:
  - `HitMIDlet.URL: http://localhost:8080/midp/hits` in the `Hellosuite.jad`
  - `String url = getAppProperty("HitMIDlet.URL") // in the code`

Connection with a Servlet

- **Installing and Running Tomcat**
  - Tomcat is distributed as a ZIP archive `http://tomcat.apache.org`
  - Unzip the download file, for instance into a root-level directory: `C:\apache-tomcat-5.5.20`
  - To run Tomcat you'll need to tell it where to find your J2SE SDK installation
    - Set the `JAVA_HOME` environment variable to `C:\Program Files\Java\jdk1.5.0_08`
  - To run Tomcat, open a command window. Change directories to Tomcat's bin directory. Type `startup`
HitServlet

```java
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class HitServlet extends HttpServlet {
    private int mCount;

    public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
        String message = "Hits: " + ++mCount;

        response.setContentType("text/plain");
        response.setContentLength(message.length());
        PrintWriter out = response.getWriter();
        out.println(message);
    }
}
```

save the source code in a file under the Tomcat root directory
named webapps/midp/WEB-INF/classes/HitServlet.java

Compiling and deploying

- C:\>set CLASSPATH=\apache-tomcat-5.5.20\common\lib\servlet-api.jar
- C:\>javac HitServlet.java
- C:\>Deploy the following file (called web.xml) in C:\apache-
tomcat-5.5.20\webapps\midp\WEB-INF

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "/-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <servlet>
        <servlet-name>bob</servlet-name>
        <servlet-class>HitServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>bob</servlet-name>
        <url-pattern>/hits</url-pattern>
    </servlet-mapping>
</web-app>
```
public class HitMIDlet
    extends MIDlet
    implements CommandListener {
    private Display mDisplay;
    private Form mMainForm;
    private StringItem mMessageItem;
    private Command mExitCommand, mConnectCommand;

    public HitMIDlet() {
        mMainForm = new Form("HitMIDlet");
        mMessageItem = new StringItem(null, "");
        mExitCommand = new Command("Exit", Command.EXIT, 0);
        mConnectCommand = new Command("Connect", Command.SCREEN, 0);
        mMainForm.append(mMessageItem);
        mMainForm.addCommand(mExitCommand);
        mMainForm.addCommand(mConnectCommand);
        mMainForm.setCommandListener(this);
    }
private void connect() {
    HttpConnection hc = null;
    InputStream in = null;
    String url = getAppProperty("HitMIDlet.URL");
    try {
        hc = (HttpConnection)Connector.open(url);
        in = hc.openInputStream();
        int contentLength = (int)hc.getLength();
        byte[] raw = new byte[contentLength];
        int length = in.read(raw);
        in.close();
        hc.close();
        // Show the response to the user.
        String s = new String(raw, 0, length);
        mMessageItem.setText(s);
    } catch (IOException ioe) {
        mMessageItem.setText(ioe.toString());
    }
    mDisplay.setCurrent(mMainForm);
}

Adding the new MIDLET

- Put the midlet source file in c:\WTK25\apps\HelloSuite\src
- Open the Ktoolbar and Click on Settings..., then select the MIDlets icon
- Enter Name and class name as HitMIDlet
Properties

- In the toolkit, click on Settings..., then select the User Defined tab.
- Click on the Add button.
- Fill in the property name as HitMIDlet.URL; the value should be the URL that invokes HitServlet.
- And finally, build.

Running
Exercises

- Follow the slides and install, compile, and run the two midlets: HelloMIDlet.java, HitMIDlet.java (the code is on the course web site)
- Write a MIDlet that displays the current date and time – use the HelloMIDlet.java code and the classe Calendar (consult the MIDP API, C:\WTK25\index.html
- Write a new MIDlet that asks a servlet to return the IPAddress of the server, the name of the student, and a timestamp (use the Java classes InetAddress and Calendar).