Java for Small Devices

Java 2 Microedition & Blackberry
Agenda

Introduction, Motivation, Overview
- The Mobile World
- Problems implied by Mobile Devices
- Programming Challenges
- Best Practices & Design Patterns

Infrastructure, Platforms, Programming Models
- Available Platforms
- J2ME

Blackberry Programming Practise
- Blackberry Platform Characterization & API Introduction
- Communication & Synchronization
- Portability
- Security
- Additional Features
The Mobile World

Motivation for Applications on Mobiles
- Ubiquitous computing
- Stay connected (with business or friends)
- Diversify communication methods
- Retrieve Information
- Personal Information Management (PIM)

Use cases
- Mail, News, Calendar, Contacts
- Business Applications (access intranet applications …)
- Check, Buy and Sell stock (time critical application)
- Play games, Download & Play music
The Mobile World – Development Environment

**Major differences**

- Development Device ≠ Target Device
  => Development Architecture ≠ Target Architecture
- Demand for Cross-Compiler
- Demand for deployment facility (e.g. IrDA, Bluetooth, Provision over air)
- Simulators may be useful
- Remote debugging facility may be useful
Problems implied by Mobile Devices

Resource-poor compared to Desktops, Servers

<table>
<thead>
<tr>
<th></th>
<th>Genuine Cell Phone</th>
<th>Smartphone or PDA</th>
<th>Average Desktop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Power</td>
<td>12-30MHz</td>
<td>50-400MHz</td>
<td>2.x GHZ</td>
</tr>
<tr>
<td>Volatile Memory (RAM)</td>
<td>512Kb-1MB</td>
<td>2-64MB</td>
<td>256MB-1GB</td>
</tr>
<tr>
<td>Non volatile Memory</td>
<td>2-4MB (mostly Flash)</td>
<td>16-512MB (depending on add-on memory card)</td>
<td>60-100GB</td>
</tr>
</tbody>
</table>

- Usually limited access to additional hardware or interfaces (e.g. Bluetooth, IRDA, RS232)
Problems implied by Mobile Devices (cont.)

Connectivity variability in performance and reliability
- Lack of reliability due limited coverage areas
- Bandwidth variations due to different (wireless) connections

<table>
<thead>
<tr>
<th></th>
<th>GSM</th>
<th>GPRS</th>
<th>UMTS</th>
<th>WLAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switched</td>
<td>Circuit Switched</td>
<td>Packet Switched</td>
<td>Packet Switched</td>
<td>Packet Switched</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>14.4 kBit</td>
<td>56 kb/s</td>
<td>2 MBit</td>
<td>54 Mbit</td>
</tr>
</tbody>
</table>

Finite energy sources
- Power consumption / duration of battery lifetime is key marketing criterion
- Applications should not consume much power
- Applications may help in managing power
- Applications should be aware of power consumption of hardware in use (Bluetooth, IRDA, RS232, Screen lighting, Network)
Problems implied by Mobile Devices (cont.)

Demand for synchronization
- Limited connectivity, bandwidth and power
- Mobile not be accepted as storage platform for information
  → Information needs to be stored locally which requires synchronization
- Synchronization demand crosses application domains
  (e.g. Mail & News, Contacts, Web Pages, Calendar, Business Data, Files)

Security threats
- Malicious Code
  - May secretly use the device and incur costs
  - May make the device unusable
- Networking related threats
  - Eavesdropping attacks
  - Replay attacks
  → used to gain secret information
Problems implied by Mobile Devices (cont.)

Variety of hardware and APIs (Portability and Ergonomics)

- Different screen dimensions
- Different screen colour depth
- Different input means (ranging from knobs and wheels to pointing devices)
- GUI constraints (e.g. senseless: variable window sizes)
- Different APIs (e.g. native, Java, .NET)

-> Contradicts application portability across different devices
Programming Challenges - Categorization

Communication
- Synchronization
- Network imposed problems

Device Limitations
- Power consumption
- Memory restrictions
- Storage capacity

Security Requirements
- Malicious code
- Eavesdropping attacks

Heterogeneity
- Platform diversity
- Screen and input device diversity
Popular available Platforms

Java (J2ME, Blackberry)
- VM Approach
- Mobile Phones
  - Native software plus VM
- Blackberry
  - Only minor native functionality (OS)
  - All user interacting software is written in Java

Symbian OS
- Native Code
- Some Symbian Platforms support J2ME

.NET (Compact Framework, Smart phone)
- VM Approach, Managed code

Embedded Linux (e.g. Montavista Linux)
- Native Software
J2ME – Java 2 Platform, Micro Edition

Java for small devices
- Simplifies Portability
- Standardization at high level allows broad range of hardware platforms
- Streamlines Application development

J2ME is a framework
- Designed to support different device classes like
  - Parking meter,
  - Mobile phone,
  - Watch
  
These impose different requirements

- Thus a J2ME runtime environment consists of multiple building blocks

http://java.sun.com/j2me
J2ME - Platform

- Java 2 Platform, Enterprise Edition (J2EE)
  - JVM™

- Java 2 Platform, Standard Edition (J2SE)
  - JVM

- High-end PDAs
  - End devices
  - Optional Packages
  - MIDP

- Mobile phones & entry-level PDAs
  - CLDC
  - KVM
  - Java Card
  - Card VM

- Servers & enterprise computers
  - Optional Packages

- Servers & personal computers
  - Optional Packages

- Smart cards
  - Optional Packages

HPI, SS2004: Components Programming & Middleware, Mobile Devices/12
Einar Lück, Konrad Hübner, Robert Mitschke
J2ME – Architecture

**J2ME environment:**
- Profile + Configuration + Virtual Machine = Java Technology Stack
“A J2ME configuration shall only define a minimum complement or the “lowest common denominator” of Java technology. All the features included in a configuration must be generally applicable to a wide variety of devices. Features specific to a certain vertical market, device category or industry should be defined in a profile rather than in a configuration. This means that the scope of a configuration is limited and must generally be complemented by profiles.”
J2ME – Configuration cont.

Defines:
- Application Management (Retrieving, Loading & Executing)
- Low Level Security Measures (e.g. Byte Code Verification)
- Available Byte Code Instructions (Usually Subset of J2SE)
- Class File Format
- Mandatory Libraries
  - This is just a basic set, additional libraries are specified in Profile

May not define optional features!!

Example Configuration:
CLDC (Connected Limited Device Configuration)
J2ME – Profile

Profile

- Based on a Configuration

Profile based Applications

OEM Specific Applications

Native Apps

Profile

OEM specific classes

Configuration

Native System Software (MID OS)

Mobile Information Device
J2ME – Profile cont.

- Completes runtime environment specification
  - Specifies additional hardware constraints
  - Specifies API available additionally to that specified by Configuration
  - Intensely narrows down specification of target device

- Still even though target platform is narrow, a Profile is targeted at a whole class of devices, not at a specific platform
  - E.g. MIDP is used on all mobile phones supporting Java, even on some Smart Phones and/or PDAs (Mobile Information Devices)

- Example Profile:
  MIDP (Mobile Information Device Profile)
CLDC – Connected Limited Device Configuration

- “…standards-based technologies for developing applications that run on small mobile devices“
- Currently two versions
  - CLDC 1.0.4 (JSR 30)
  - CLDC 1.1 (JSR 139)
  - 1.1 is backward compatible (1.0 Applications will run on 1.1 System)

CDC - Connected Device Configuration

- “…a standards-based framework for building and delivering applications that can be shared across a range of network-connected consumer and embedded devices“
- CDC (JSR 36)
CLDC Virtual Machines

KVM (CLDC 1.0)
- the smallest possible “complete” Java virtual machine
- hence the name K, for kilobytes (40 kilobytes to 80 kilobytes)
- Clean and highly portable
  - C-language
- Modular and customizable

Hotspot VM (CLDC 1.1)
- dynamic, adaptive compiler
  - most frequently used &
  - time-critical parts
- optimized interpreter
  - Assembly-language
- 8-10x faster than KVM
J2ME CLDC 1.0

Hardware requirements
- 128KB non volatile memory for Virtual Machine and libraries
- 32KB RAM for java runtime and object memory
  → Almost no sensible application will run with that amount of memory though

Software requirements
- One schedulable entity
  - Thus underlying system does not need to provide multi-threading/processing
- Not covered
  - Real time
  - Separate address spaces
  - Latency behaviour

Scope
- Virtual machine features
- Core java libraries (java.lang.*, java.util.*, java.io.*)
- Input/Output
- Classfile format
J2ME – CLDC – Application Management

Application Management

- No persistent storage required for applications
  - However it is strongly suggested
  - Without storage apps could be loaded on demand

- Management of storage and application start up is considered out of scope
  - Underlying platform has to supply means for the user to execute apps
  - No assumptions about possible file systems etc.
J2ME – CLDC compared to J2SE

**Subset of J2SE**

- No floating point support
  - Byte code instructions removed
  - was reintroduced in CDLC 1.1
- No Java Native Interface (JNI, calling of native c-functions)
- No user defined class loaders (security)
- No reflection
- No Thread groups
  - Multithreading is supported, thread groups may be implemented by application developer
- No finalization
- No weak references
  - references not counting as references for Garbage collector
  - Reintroduced in CLDC 1.1
- Limited error handling - exceptions do exist though
Classes outside J2SE may not use the java.* name space
J2ME – CLDC – Connection Framework

Connection Framework of CLDC

Connection

StreamConnection
Notifier

Input
Connection

Output
Connection

Datagram
Connection

Stream
Connection

Content
Connection
J2ME – CDLC – Connection Framework

Key ingredient of CLDC

```java
Connector.open("<protocol>:<address>;<parameters>");
```

- May potentially be used for
  - HTTP
  - Bluetooth, RS232
  - TCP, UDP
  - Any other unidirectional or bidirectional stream- or datagram-based communication

No implementation of any protocol contained in CLDC → Task of Profile

Allows broad functionality

- Any known protocol or communication technique may be supported

Saves non volatile memory (flash)

- Abstract interface in Java potentially covers any protocol
- Lightweight implementation in platforms own native code
J2ME – CLDC - Security
Security

- Byte code Verification divided into two step – process
- Out of device verification at compile time
  - Then code to be compiled is pre verified
  - Slightly enlarges code size
  - binary code is restructured and attributes are added
    - Stackmap attribute
  - Pre verification is semantically verified by runtime verifier
    - Exploit using fake pre verification should not work

- Shortened verification at runtime
  - Due to attributes in binary code only a single run is required
  - This is not memory intensive due to pre verification
    - Simple linear scan
    - 10kbyte of x86 Code and 100bytes of RAM required on device

 Roughly: No illegal byte code or code violating type safety can be executed
J2ME – MIDP a sample Profile

MIDP – Mobile Information Device Profile

- Two Versions available
  - MIDP 1.0 (JSR 37)
  - MIDP 2.0 (JSR 118)
- Based on CLDC
  - MIDP 1.0 is supposedly based on CLDC 1.0
  - MIDP 2.0 is supposedly based on CLDC 1.1
- Created for cellular phones, two way pagers, wireless enabled PDAs
- Created by MIDPEG (MIDP Expert Group)
- Instead of Applet or Servlet, Applications are called MIDlet
- MIDP Style Guide
J2ME – MIDP Requirements

Requirements on top of CDLC

- **Hardware**
  - 96x54 pixel screen dimension, 1 Bit colour depth
  - ITU-T Keyboard (Phone keyboard), QWERTY Keyboard and/or touch screen
  - 128kb additional non volatile memory for MIDP components
  - 8kb non volatile memory for application created persistent data
  - 32kb volatile memory for Java runtime
  - Two way, wireless, potentially unstable, low bandwidth connectivity

- **For comparison:**
  - .Net Compact Framework requires multiple Megabytes of volatile and non volatile memory
J2ME – MIDP Requirements

- Operating System must support

  - Writing to persistent storage (for app created data, not necessarily apps)
  - Access to networking hardware
  - Time base
  - Access to bit mapped display
  - Access to user input
public class Hello extends MIDlet{

    public void startApp(){
    }

    public void pauseApp(){
    }

    public void destroyApp(boolean b){
    }

}
J2ME – MIDP Connection Framework support

Connection Framework

CLDC did not require any protocols, just supplied the Framework

MIDP requires HTTP

- May base on Gateway
  - Device does not need to implement full IP Stack
  - May use arbitrary protocol, in case Gateway supplies IP Stack, like in WAP or switched circuit
J2ME – MIDP Connection Framework deficiencies

- MIDP does not require other protocols
  - No socket
  - No datagram based communication
  - No required communication via other interfaces (Bluetooth, IRDA, …)

- Portable applications have to rely on HTTP only

- SyncML is not part of MIDP → Has to be implemented individually
Persistent Storage

- Allows storage of application data in so called record stores
- Does not require file system, abstract data store
- Automatic serialization of write requests
  - Application does not have to take care of organizing concurrent write requests

```java
private RecordStore recordStore =
    RecordStore.openRecordStore("nirwana", true);
byte[] b = myChunkOfInfo.toByteArray();
recordStore.addRecord(b, 0, b.length);
```

- Additionally supports enumeration, filtering
- Looks like it was made to access high score tables 😊
Display Capabilities

Probably most important part of the API

- Applications are probably GUI-heavy as platform is not useful for heavy computing
- Most applications will require intensive user interaction

API is split into two parts

- Highly Abstract API (called High level API)
  - Abstracts from all direct screen and input device access

- Low Level API (called Low Level API 😊)
  - Allows access to the screen and input devices
J2ME – MIDP – High Level GUI API

High level API

- Abstracts from Screen and Input device capabilities
- Allows to display choices, forms
  - The way these are displayed on the screen are implementation dependent
  - Device’s own colour scheme is used
  - Information is displayed according to devices GUI scheme and capabilities
- Allows to listen to abstract commands
  - Device decides on how to allow user interaction
  - Device should adhere to it’s own scheme, which the user is accustomed to
- Application programmer does not have to worry about available display capabilities, available input devices
- Application just displays information and listens for returning events
- Widgets are: Lists, Selections, Gauge, Textbox ...
```java
private Form mMainForm;
private Command ok, exit;

public HApi(){
    mMainForm = new Form("Available Memos");
    mMainForm.addCommand(ok = new Command("Next", Command.OK,0));
    mMainForm.addCommand(exit = new Command("Exit", Command.EXIT,0));

    mMainForm.setCommandListener(this);

    Display d = Display.getDisplay(this);
    d.setCurrent(mMainForm);
}

...

public void commandAction(Command c, Displayable s) {
    if (c == exit){
        destroyApp(false);
        notifyDestroyed();
    }
}
```
J2ME – MIDP - High Level API in action

High Level API Example – Nokia 6230 & Sun MediaControlSkin

1 screen – 2 commands

1 screen – 3 commands
J2ME – MIDP - High Level API in action cont.

High Level API Example

All: 1 screen – 3 commands

Sun's imaginary devices (WTK 2.3)
J2ME – MIDP Low Level API

Low Level API

- Allows direct access to display and input devices
  - Pictures may be shown
  - Pixels may be drawn using familiar commands (`drawString()`, …)
  - Application has to adjust to screen dimensions
  - Application has to adjust to input devices
    - Means to find present input devices
    - Application has to adjust its behaviour
J2ME – MIDP Low Level API deficiencies

- Mostly used for graphical game development
  - javax.microedition.lcdui.game in MIDP 2.0

- Combination of Low and High-Level API is not intended
  - But may work and impose security threat → see news on S55 SMS exploit

- Applications using the Low Level API are either
  - not portable at all
  - portable only with significant (non trivial) coding overhead
    - Adjusting to screen size and colour depth
    - Adjusting to present input devices
class TextCanvas extends Canvas
{
    ...

    protected void paint(Graphics g) {
        int height = getHeight ();
        int width = getWidth ();
        if (hasPointerEvents ()){
            // support the input device pointer
        }
        if (hasPointerMotionEvents ()){
            // prepare for motion events
        }
        if (isDoubleBuffered ()){
            // take advantage of double buffered screen
        }
        // draw something that makes sense on the current platform
    }
}
J2ME – MIDP – Application Management

Application Packaging & Starting

- MIDlets are packet into so called MIDlet Suites
  - One or more MIDlets including all their classes
  - Resources used by the MIDlet(s)
  - JAR-Manifest, containing information about all contained MIDlets and the Suite itself

→ MIDlet Suites are JAR-Files

Selection and Starting of Applications

- Is out of the scope of the standard
- Is individually supplied by underlying operating system
Java Application Descriptor (**midletname.jad**)

- File that additionally comes with the JAR
- Contains Information from the Manifest
- Optionally Contains Information on the location of the JAR (URL)
- Thus allows to retrieve Info before downloading the whole JAR
- Allows checking for updates and **OTA Provisioning**

MIDlet-1: Editor, Editor.png, Editor
MIDlet-Jar-Size: 2070
MIDlet-Jar-URL: Editor.jar
MIDlet-Name: Editor
MIDlet-Vendor: Unknown
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
J2ME – MIDP – Security concepts

Security

Sandbox Concept

- Standard specifies all APIs available to MIDlet
- Additional functionality may not be accessible to MIDlet
- MIDlets may not access system functionality
- MIDlets may only communicate within MIDlet Suites and then only via Record Store
New in MIDP 2.0: Trust Model for sensitive (costly) functionality

- E.g. before a MIDlet may send HTTP Traffic (or SMS, ...) user would like to be asked
- Organized in permissions and protection domains
- Using PKI MIDlets may be trusted without user interaction (optional)
J2ME – Summary

Summarization

In our opinion J2ME is a step into the right direction

- CLDC + MIDP as used in most mobile phones today is not gone far enough
- Not enough functionality to implement the cool things
- Can mainly be used for games
- Portability is only ensured when sticking with the smallest common denominator
CDLC + MIDP + OEM libraries can become powerful tools depending on the platform and the power of OEM libraries

Basic functionality is common, short time to get familiar with a new platform

Now let's look at Blackberry, the Flagship of J2ME devices!!
The Blackberry Platform – The Company

Research in Motion

- founded 1984
- located in Canada (Waterloo, Ontario)
- Developer, manufacturer and marketer of wireless solutions
- RIM Inter@ctive Pager introduced in the mid-90s
  - Mobitex and DataTAC data networks in USA
  - Devices with TrackWheel and small keyboard
- In 1999 change of focus to „Blackberry Wireless Email Solution“
  - Integration into corporate email system
  - Introduction of Java support
  - Support for new networks like GPRS
- Newest models with phone support, color display and 16MB flash memory
The Blackberry Solution

Middleware integrating Handhelds into corporate network

- Provide two-way communication means
- Integrate into email system
- Support only for MS Exchange and Lotus Notes
- Gain access to company data
The Blackberry Solution - Simple Architecture

**Corporate site**
- Mailserver (Exchange or Lotus Domino)
- Blackberry Enterprise Server (BES)

**Carrier**
- provides Wireless Network
- transmits Data sent by BES to Handheld and vice versa
The Blackberry Solution – Detailed Architecture

Four important components

- Handheld(s)
- Carrier (Wireless Networks)
- RIM network operations center (NOC)
- Blackberry Enterprise Server (BES)
The Blackberry Platform - BES

Two-Way connection to NOC
- Both Handheld and email-server have to access each other
- Handheld paired to exactly one BES
- NOC has to know all these linkings
- Encryption between BES and Handheld
  - Key generated by BES when Handheld is associated to BES
- Connected to NOC via permanent TCP Port 3101 connection
  - RIM´s Service Relay Protocol (SRP) is utilized

Mobile Data Service (MDS)
- Enables Handheld to receive data from the company network
  - HTTP POST requests containing data
- Provides internet access for Handheld
The Blackberry Platform - NOC

Two NOCs worldwide
- Waterloo, Ontario (Canada)
- Sterling, Virginia (US)

Single point of connection for all Handhelds
- Handhelds registered with their email-address and id
- Unique association to a user

Directly connected to all carriers supporting the Blackberry system

Directly connected to all BESs installed worldwide
- TCP Port 3101 for incoming traffic by BES
- TCP Port 25 for traffic to company (SMTP Mailserver)
The Blackberry Platform – NOC (cont.)

- **Handles all email traffic of the Handhelds**
  - Handhelds send email via carrier to NOC
  - NOC forwards traffic to company’s mail server
  - BES sends mails for Handheld to NOC which forwards it to the carrier

- **Handles all data push traffic to Handhelds**
  - Forwards traffic received by MDS to Handheld
  - Forwards traffic received by Handheld to MDS
The Blackberry Platform - Carrier

Provides wireless network access to the Handheld
- Connected to RIM’s NOC
- Provides additional internet access via WAP (in contrast to MDS)
- Provides phone services if supported by handheld

More than 50 Carriers connected to NOC
The Blackberry Platform – Handheld Overview

email via BES Push-Service
- Handheld associated to exactly one BES and one user
- All emails delivered to and by this BES

email via other account (POP3)
- No BES needed
- No push-service possible

email via special account by carrier / RIM
- No BES needed
- Suitable for personal users
- Push-service possible
- Device-User-Association required
The Handheld - Details

Old devices with C++ Interface
- Data services only
- Still available on market
- Application development more complex (i.e. Memory Management)

Modern Blackberry Handhelds with Java
- Data and phone services
- Easy application development
- Garbage Collection leads to less memory consumption problems

Java VM
- Developed by RIM
- Supports J2ME CLDC 1.0 and MIDP 1.0 specification
- Device-specific API for access to all Blackberry features
- Runs non-standard bytecode => RIM Development tools have to be used
Communication Paradigms

Communication Means
- HTTP(s)
- Sockets
- Push Data Service
  - Server-initiated connections to Handheld
  - Requires BES and MDS

Two available gateways
- Blackberry Enterprise Server
- WAP gateway

Standard Communication routed through BES
- Seamless, no special connection establishment to carrier needed

Connections via WAP gateway
- Specially formatted URLs necessary, containing WAP gateway information
Application Models

**Browser-based**
- Connection to Application Server via WAP or MDS
- Limited functionality defined by browser capabilities
- High network utilization because completely located on server
- Easy application deployment: no installation necessary

**Custom Java Applications**
- Rich feature set
- Custom User Interface
- Data processing on Handheld possible
  - Minimizes Network utilization
- Installation necessary
Blackberry Java Stack

- MIDP Java Application
- Custom BlackBerry Java Application
- MIDP
- CLDC
- BlackBerry APIs
- Java Virtual Machine
Application Portability

- **J2ME CLDC / MIDP Applications**
  - Portable to all JavaME capable devices
  - Reach more possible customers
  - Suffer from small functionality

- **Blackberry API Applications**
  - Portable only to other Blackberry Devices
  - Some features only on certain devices (Color vs Black/White)
  - Provide more functionality to less customers
Securing Applications

Connection security between Handheld and BES

- Triple DES encryption between Handheld and BES
  - Symmetric encryption mechanism
  - Key generated when Handheld is associated to BES
  - Key stored in user’s Exchange Mailbox and on Handheld
  - BES retrieves Key from Exchange server to encrypt data

- HTTPS between BES and ApplicationServer
- SSL/TLS possible between Handheld and ApplicationServer
Securing Applications

Communication

- Support for HTTP and HTTPs via MIDP API’s `HttpConnection` class
- MDS can act as proxy for the Handheld
  - Blackberry encryption used between corporate network and Handheld
- S/MIME support
  - private-public key system
  - S/MIME support package required to transfer keys to Handheld

Controlled APIs for system access

- Access to controlled APIs requires digital signment by RIM
  - Encryption API
  - Persistent storage API
  - System APIs
- Without those APIs no communication, device access or storage possible
Blackberry Memo Example

1 Screen, 3 Commands
Securing Handheld

Handheld supports Password Authentication

- Only SHA-1 Hash stored on Handheld => even memory access cannot reveal password
- Too many attempts result in Handheld memory erasure
- Admin can change Handheld password remotely
- Admin can lock or delete information from lost Handhelds

Policies for third-party applications

- Restrictions on type of network connection (WAP, BES, serial, USB)
- Restrictions on connection target (internal / external)
Handheld Programming – Hardware Details

Low-Level access to System Hardware
- net.rim.device.api.system package

Serial Port / USB Port
- Read / Write operation transmitting byte(s) are provided

Alerts
- Buzzer, Sound, Vibrate alarm

Example:

```java
public void alertMe() {
    Alert.startVibrate(1000);
    short[] p = {5000, 500}; // 5 kHz for 500 ms
    Alert.startAudio(p, 50); // play tune p at 50% volume
}
```
Handheld Programming

Persistent Data Storage

- Write data into flash memory via Blackberry’s PersistentStore class
  - PersistentObject acts as storage root
- Use MDIP RecordStore for better portability

Example:

```java
public void storeUser() {
  final long appID = 0x6af0b5eb44dc5164L; // i.e. App Hash
  PersistentObject store = PersistentStore.getPersistentObject( appID );
  String username = "Hans Meier";
  String password = "hmpwd";
  String[] userinfo = {username, password};
  synchronized (store) {
    store.setContents(userinfo);
    store.commit();
  }
}
```
static final long appID = 0x60ac754bc0867248L;

public void shareObject() {
    String msg = "Some shared text";
    RuntimeStore store = RuntimeStore.getRuntimeStore();
    store.put( appID, msg ); // share object
    String newmsg = "Some other shared text";
    Object obj = store.replace( appID, newmsg); // replace shared object
    Object storedObj = store.get(appID); // receive shared object
}

public void waitForSharedObject() {
    RuntimeStore store = RuntimeStore.getRuntimeStore();
    try {
        Object obj = store.waitFor(appID); // wait for object to be shared
    } catch (ControlledAccessException e) { /* insufficient permissions */} catch (RuntimeException e) { /* time out */ }
}
Handheld Programming – Low Memory Management

- In case of low memory old data is automatically removed
  - Remote Address Lookup Requests
  - Browser cache
  - Old attachments
  - Old calendar entries
  - Old messages

- Calendar and emails are not sent back to the server but simply removed

- Applications can
  - Register listeners for low-memory events
  - Register objects for removal by VM in case of low memory
Handheld Programming – Device Details

**Device class**
- Turn Handheld off
- Set Handheld into storage mode

**DeviceInfo class for retrieving information about**
- Battery Power state
- Battery temperature (too hot / cold)
- Missing battery ;-)  

**RadioInfo class**
- Get radio state / signal strength
- Get available network type and services

```java
public void getRadioInfo() {
    int type = RadioInfo.getNetworkType(); // like CDMA, GPRS
    int service = RadioInfo.getNetworkService(); // like home zone, city
    int signal = RadioInfo.getSignalLevel(); // signal in dBm
}
```
Handheld Programming – Mail

Full access to email storage

- email encapsulated in Message class
  - Full message manipulation possible
- email Storage encapsulated in Store class
  - Predefined folders for inbox, outbox, sent, other
- Transmission means encapsulated in Transport class
  - Initially, only 2k received, more data can be requested
  - email can be sent, forwarded etc
Handheld Programming – Mail

Example

```java
public void sendEmail() {
    Store store = Session.getDefaultInstance().getStore();
    Folder[] folders = store.list(Folder.SENT);
    Folder sentfolder = folders[0];
    Message msg = new Message(sentfolder);
    try {
        Address toList[] = new Address[1];
        toList[0] = new Address("hans.m@email.de", "Hans Meier");
        Address from = new Address("helga.m@email.de", "Helga Meier");
        msg.addRecipients(Message.RecipientType.TO, toList);
        msg.setFrom(from);
        msg.setSubject("Test Message");
        msg.setSubject("This is a test message.");
        Transport.send(msg);
    }
    catch (AddressException e) { /* invalid email-address detected */ }
    catch (MessagingException e) { /* sending or setting content failed */ }
}
```
Handheld Programming – Pim

PIM accessible through PIM class
- Contact List, ToDo List, Event List

```java
public void addContact() {
    ContactList contactList = null;
    try {
        contactList = (ContactList)PIM.getInstance().openPIMList(
            PIM.CONTACT_LIST, PIM.READ_WRITE);
        Contact contact = contactList.createContact();
        String[] name = new String[7];
        name[Contact.NAME_FAMILY] = "Hans";
        name[Contact.NAME_GIVEN] = "Meier";
        name[Contact.FORMATTED_NAME] = "Hans Meier";
        contact.addStringArray(Contact.NAME, Contact.ATTR_NONE, name);
        contact.addString(Contact.TEL, Contact.ATTR_HOME, "030-1234567");
        contact.addString(Contact.EMAIL, Contact.ATTR_WORK, "hans.m@email.de");
        if (contact.isModified()) {
            contact.commit();
        }
    } catch (PIMException e) { return; }
}  
DEMO: PIM Demo
```
Location Based Services

Location Based services require

- Access to a table associating Cell IDs to locations
- Table stored on the handheld
  - Updates on all handhelds in case of changes
  - Memory consumption not acceptable
- Table stored on some server
  - Connection to internet required
  - Easy updates
  - Higher traffic and thus costs

Blackberry API supports retrieval of

- Base Station Identity, Cell ID, Location Area, Routing Area

```java
public voidgetCellInfo() {
    GPRSCellInfo info = GPRSInfo.getCellInfo();
    int baseStationId = info.getBSIC();
    int cellId = info.getCellId();
}
```
Location Based Services II

Different Approach: use Carrier Services
- T-Mobile offers HTTPS interface to retrieve location
  - Latitude, Longitude
  - Point
  - Circle (+ Radius)
  - City Area Code

Common Problems
- Cell size ranges from 500m in cities to 30km on flat land
- Position within cell cannot easily be determined
Application Deployment

- Deployment possible via BES or Desktop Software

- Classes must be converted into Blackberry-specific type

- Application Installation only with user approval
  - User knows which applications are installed
  - Prevents unauthorized application installations
Development Tools

RIM Java Development Environment
- Special Build Process (Blackberry requires custom class format)
- Supports Application Signing
- Supports Debugging

Emulation environment
- Email server emulator
- MobileDataService emulator
- Handheld emulator
Development Tools
Implementation – EclipseME - Screenshots
Implementation – IDE showcase – J2WT

Sun J2ME Wireless Toolkit 2.3

Basic Toolkit used for plain MIDlet programming

- No OEM APIs supported

- Includes device simulators for imaginary devices
  - Different screen sizes, colour depth and input devices
  - Supports detailed configuration of simulated environment
    (e.g. connectivity, cpu speed, memory available …)
Implementation – J2WT - Screenshots

Project "Editor" loaded
Project settings saved
Building "Editor"
Build complete
Project settings saved
Building "Editor"
Wrote C:\Java\WTK21\apps\Editor\bin\Editor.jar
Wrote C:\Java\WTK21\apps\Editor\bin\Editor.jad
Build complete
Running with storage root DefaultColorPhone
constructor begin
hello world I am alive
constructor done
startApp begin
StartApp done
run begin
setting display
run done
Implementation – IDE showcase – Blackberry IDE

- RIM Java Development Environment
  - Special Build Process (Blackberry requires custom class format)
  - Supports Application Signing
  - Supports Debugging
Testing – Nokia Series 40 SDK

**Nokia 6230 Emulator**

- Look & Feel of real device
- Bluetooth connection with 2nd emulator instance
- Network Sniffer
- Memory consumption
- VM log messages