5. More on app development

- **Some other mobile development systems**
  - MIDP Java
  - Maemo
  - Android
  - iPhone
  - Windows Mobile
  - OpenMoko
- Commonly used implementation techniques
- Summary

Mobile Java (MIDP in particular)

- Most importantly an add-on to proprietary phone OS based systems
  - Huge device space that is already deployed
  - No need to modify the already existing proprietary platform but simple wrapping is enough!
- Disclaimer: Full Java software stacks exist:
  - RIM & Blackberry
  - JavaFX Mobile (earlier SavaJe)
  - (Google Android --- at least kind of!)

### Building Mobile Java Infrastructure

<table>
<thead>
<tr>
<th>Hosting operating system</th>
<th>Access to hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>KVM</td>
<td>Mobile HotSpot VM</td>
</tr>
<tr>
<td>Standard-featured VM</td>
<td></td>
</tr>
</tbody>
</table>

Connected Limited Device Configuration
- No floating point support (v.1.0)
- Simplified security scheme
- No user-defined class loaders
- No finalize, thread groups, reflection, or JNI
- Also other simplifications

Connected Device Configuration
- Standard facilities

### Application development facilities

<table>
<thead>
<tr>
<th>MIDP</th>
<th>IMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLDC</td>
<td>CDC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KVM</th>
<th>Mobile HotSpot VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard-featured VM</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hosting operating system</th>
<th>Access to hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>KVM</td>
<td>Mobile HotSpot VM</td>
</tr>
<tr>
<td>Standard-featured VM</td>
<td></td>
</tr>
</tbody>
</table>

**Midlet (MIDP)**

![Midlet Diagram]

**Development Process**

![Development Flowchart]

**Sample Midlet**

![Sample Midlet Chart]

**Midlet Implementation**

- Extend class `Midlet`
- Implement methods
  - Constructor
  - `startApp`
  - `destroyApp`
  - `pauseApp`
- Usually some way of interaction for the user required as well
  - E.g. `CommandListener` interface

**Behavior**

![Behavior Chart]

**Code...**

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

public class PersonalityTest extends MIDlet {
    private Command positive, negative, exitCommand;
    private String Questions[] = {""The weapon of the brave is in his heart."",""A man is a lion for his own cause.",""Courage leads to the stars, fear toward death.",""Faced with crisis, the man of character falls back upon himself.",""It is not the oath that makes us believe the man, but the man the oath.",""Whoever is careless with the truth in small matters cannot "
    };
    private int nStd, count = 0; /* Counter for statement and answer pair. */
    private String Questions[] = {""Questions 1",""Questions 2""};
    private TextBox tb;
    private int nStd = 0;

    public void startApp() {
        tb = new TextBox(""You seem to be an overly positive person."");
        tb.show();
        nStd = 0;
    }

    public void commandAction(Command c, int id) {
        switch (id) {
            case positive.getId():
                tb = new TextBox(""You appear as a grumpy/gang negative person."");
                tb.show();
                nStd = 0;
                break;
            case negative.getId():
                tb = new TextBox(""You appear as a grumpy/gang negative person."");
                tb.show();
                nStd = 0;
                break;
        }
    }

    public void displayMessage(String s) {
        for (int i = nStd; i < nStd + 3; i++) {
            tb = new TextBox(""You appear as a grumpy/gang negative person."");
            tb.show();
            nStd = 0;
        }
    }
}
```
public void destroyApp(boolean u) {}
protected void pauseApp() {}

private void pickStatement() {
    if (nth == Qlen) {
        giveInfo();
    } else {
        displayStatement(Questions[nth]);
        nth = nth + 1;
    }
}

taxbox = new TextBox("Statement Selection", statement, 256, 0);
taxbox.addCommand(positive);
taxbox.addCommand(negative);
taxbox.setCommandListener(this);
Display.getDisplay(this).setCurrent(taxbox);

private void giveinfo() {
    if (count > 1) {
        tb = new TextBox("PersonalityTest", "You seem to be an overly positive person.", 55, 0);
    } else if (count < -1) {
        tb = new TextBox("PersonalityTest", "You appear as a graviy negative person.", 55, 0);
    } else {
        tb = new TextBox("PersonalityTest", "You seem to have difficulties in being in line with yourself.", 80, 0);
    }
    tb.addCommand(exitCommand);
    tb.setCommandListener(this);
    Display.getDisplay(this).setCurrent(tb);
}

private void pickStatement() {
    if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    } else if (c == positive) {
        count++;
    } else {
        count--;
    }
    pickStatement();
}

protected void startApp() {
pickStatement();
}

protected void commandAction(Command c, Displayable d) {
    if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    } else if (c == positive) {
        count++;
    } else {
        count--;
    }
pickStatement();
}

private void PersonalityTest() {
    exitCommand = new Command("EXIT", Command.EXIT, 1);
negative = new Command("NO", Command.CANCEL, 2);
positive = new Command("YES", Command.OK, 2);
}

protected void exitCommand() {
    destroyApp(false);
    notifyDestroyed();
}

taxbox = new TextBox("Statement Selection", statement, 256, 0);
taxbox.addCommand(positive);
taxbox.addCommand(negative);
taxbox.setCommandListener(this);
Display.getDisplay(this).setCurrent(tb);

private void giveinfo() {
    if (count > 1) {
        tb = new TextBox("PersonalityTest", "You seem to be an overly positive person.", 55, 0);
    } else if (count < -1) {
        tb = new TextBox("PersonalityTest", "You appear as a graviy negative person.", 55, 0);
    } else {
        tb = new TextBox("PersonalityTest", "You seem to have difficulties in being in line with yourself.", 80, 0);
    }
    tb.addCommand(exitCommand);
    tb.setCommandListener(this);
    Display.getDisplay(this).setCurrent(tb);
}

private void pickStatement() {
    if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    } else if (c == positive) {
        count++;
    } else if (c == negative) {
        count--;
    }
pickStatement();
}

protected void startApp() {
pickStatement();
}

protected void commandAction(Command c, Displayable d) {
    if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    } else if (c == positive) {
        count++;
    } else if (c == negative) {
        count--;
    }
pickStatement();
}
MIDP 1.0 libraries and interfaces

- **UI**
  - Different device specific wrapping for widgets
    - Canvas: Low-level interface; control for programmer
    - Screen: Device specific widgets; control for the widgets and library;
      navigation, scrolling etc. built in
      - Different screen capabilities (e.g. screen, color, ...)
    - Different input mechanisms (keyboard, touch screen, ...)
  - Network interface
    - Originally http focused, no link to final implementation-level
      protocol; later improved considerably
- **Persistence**
  - RMS, record management system
    - Reflects a simple, record-oriented database
    - Records interfaced with byte arrays

Other screen-related issues

- **Game API (MIDP 2.0)**
  - Sprites, layering, etc.
  - Enables native implementation
- **GUI enhancements**
  - Builds on MIDP 1.0 LCDUI
  - Custom items, layout control, graphical enhancements...
  - Backward compatibility
- **Further enhancements provided in later JSRs for e.g.**
  - Multimedia
  - 3D

Networking Interface

- Originally http focused (MIDP 1.0) with multiple carriers (e.g. TCP/IP, WAP, i-Mode)
- Numerous networking extensions and improvements
  - HTTPS, SSL
  - Serial port
  - Sockets
  - Server Sockets
  - Data grams
  - Network push feature
    - Network initiated MIDlet launch
- OTA (over the air) applications
  - Installation
    - Verify integrity, certificates, requested permissions
  - Invocation
    - Permissions
    - Originally extension, later required

Persistence

- Based on Record Management System
  - Records contained by record stores
- RMS sharing obeys MIDlet suite principle
  - Same name can be used for different RMS record stores in different MIDlet suites
  - Later relaxed
- Programmer (or device manufacturer) handles mutual exclusion
  - listRecordStores, openRecordStore, getName, getSize, addRecord, deleteRecord, ...
  - DataInputStream, DataOutputStream, ByteArrayInputStream, ByteArrayOutputStream

Other services

- **Timers**
  ```java
class myTask extends TimerTask {
    public void run() { ... }
}
myTimer = new Timer();
myTask = new myTask();
myTimer.schedule(myTask, 100, 1000);
```
- **Access to resource files**
  - getResourceAsStream
- **System properties**
  - microedition.platform (name of the device)
  - microedition.encoding (character encoding)
  - microedition.configuration (used configuration and its version)
  - microedition.locale (language and country)
Further interfaces

- JSR-120: Wireless Messaging API
  - Short Message Service (SMS)
  - Unstructured Supplementary Service Data (USSD)
  - Cell Broadcast Service (CBS)
- JSR-135: Mobile Media API
  - Straightforward access and control of basic audio and multimedia resources and files
- JSR-172: J2MET Web Services Specification
  - Infrastructure for basic XML processing capabilities
  - Reuse of web service concepts when designing J2ME clients for enterprise services
  - Provides APIs and conventions for programming J2ME clients of enterprise services
  - Programming model for J2ME client communication with web services, consistent with that for other Java clients such as J2SE.

Further interfaces

- JSR-184: Mobile 3D Graphics for J2MET
  - Lightweight, interactive 3D graphics API, which sits alongside J2ME and MIDP as an optional package.
  - API targeted at devices that typically have very little processing power and memory, and no hardware support for 3D graphics or floating point math
- JSR-190: Event Tracking API for J2MET
  - Optional package that standardizes the tracking of application events in a mobile device
  - Submission of these event records to an event-tracking server via a standard protocol
- JSR-195: Information Module Profile
  - Like MIDP but no screen or keyboard

Further interfaces

- JSR-177: Security and Trust Services for J2MET
  - Necessary step for a device to become trusted, i.e., to provide security mechanisms to support a wide variety of application-based services, such as access to corporate network, mobile commerce, and digital rights management
  - Model and a set of APIs that enable applications running on a J2ME device to communicate with a smartcard
- JSR-179: Location API for J2MET
  - Optional package that enables developers to write mobile, location-based applications for J2ME devices
- JSR-180: Session Initiation Protocol (SIP) for J2MET
  - Session Initiation Protocol (SIP) is used to establish and manage multimedia IP sessions
  - General SIP API for J2ME devices based on the SIP protocol defined by IETF and 3GPP, and targeting resource constrained platforms.

Some fundamental specifications

- JSR-68: J2MET Platform Specification
  - Defines the “ground rules” for the J2ME platform architecture and J2ME standardization activities.
  - Formalizes the fundamental concepts behind J2ME, such as the notions of a configuration and profile
  - Defines how new J2ME APIs can be formed, e.g., by subsetting existing APIs from Java 2 Platform, Standard Edition
- JSR-185: JavaT Technology for Wireless Industry
  - Defines how various technologies associated with MIDP work together to form a complete handset solution for the wireless services industry
  - Which optional packages fit with which profiles? How an end-to-end solution for interoperable Java applications will work? How the migration of applications can occur and to which profiles as the devices become more capable?

5. More on app development

- Some other mobile development systems
  - MIDP Java
  - Maemo
  - Android
  - iPhone
  - Windows Mobile
  - OpenMoko
- Commonly used implementation techniques
- Summary

Maemo

- Nokia Internet Tablet
- Based on Debian Linux distribution and package management
- Development environment based on Scratchbox virtualization system
- Pretty much everything can be programmatically accessed
- C, C++, Python, JavaScript
5. More on app development

- Some other mobile development systems
  - MIDP Java
  - Maemo
  - Android
  - iPhone
  - Windows Mobile
  - OpenMoko
- Commonly used implementation techniques
- Summary

Android

- Originally Google, later Open Handset Alliance
- Open source – No licence or distribution payments
- Beta SDK 11/2007, 1.0 SDK 09/2008
  - SDK includes emulator, debugging tools, on-device debugging
- T-Mobile G1
  - 10/2008 (USA ja UK)
  - Europe early 2009
- Several manufacturers aim at launching Android phones during 2009
  - Motorola, Lenovo, Sony Ericsson, Samsung

Android – Technology

- Built on top of Linux
  - Drivers, processes and memory management, security, networking, power management
  - Libraries such as SQLite, WebKit, OpenGL
- Dalvik VM
  - Abstract view to the hardware
  - Development done using Java, not a Java VM
  - Runs Dalvik executables which are obtained based on Java bytecodes
- Possible to bypass VM and use C/C++
  - Requires information on target device
  - Not officially supported

Android – Application development

- All applications are equal
  - Can be replaced by 3rd party apps
  - Same APIs are available for all programs (calls, SMS, WLAN, GPS, camera, sensors, …)
  - Inter-application communication
    - Application can request making a call and the system seeks the corresponding service
- Any app can offer services and use services offered by others
- Background execution allowed
  - (Unlike in Standard Java MIDP 2.0)
  - Programs cannot control their own life cycles
  - VM can interfere with the execution of the VM by suspending and resuming them; apps must be designed to be compatible with this
5. More on app development

- Some other mobile development systems
  - MIDP Java
  - Maemo
  - Android
  - iPhone
  - Windows Mobile
  - OpenMoko
- Commonly used implementation techniques
- Summary

Technology layers

- Core OS and Core Services layers: fundamental interfaces for iPhone OS
  - File access, low-level data types, Bonjour services, network sockets, SQLite, access to POSIX threads and UNIX sockets, etc.
- Media layer:
  - 2D and 3D drawing, audio, video, OpenGL ES, Quartz, and Core Audio, Core Animation
- Cocoa Touch layer:
  - Fundamental infrastructure used by applications, object-oriented support for collections, file management, network operations, access to the user’s contact and photo information and to the accelerometers and other hardware features of the device.
  - UIKit framework provides the visual infrastructure for your application, including classes for windows, views, controls, and the controllers that manage those objects
- The starting point for any new project is the Cocoa Touch layer, and the UIKit framework in particular.

Objective C Code

```objective-c
int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

5. More on app development

Windows Mobile

- .NET Compact Framework
  - Based on .NET environment
  - Common Language Runtime (CLR)
  - MicroSoft Intermediate Language (MSIL)
- Just-In-Time compilation from MSIL to binary when used
- Tools somewhat similar than in general with .NET development
  - Small differences in programming perspective (UI, performance, resources, etc…)

iPhone

- Mac OS X based system; origins trace to Mach kernel old Unix implementations
- Objective C for programming
  - Low level interfaces commonly based on C; more advanced interfaces (e.g. Multimedia), mixture of C and Objective C
- Explicit redesign to support phone UI
  - Touch events
  - Not application centric
5. More on app development

- Some other mobile development systems
  - MIDP Java
  - Maemo
  - Android
  - iPhone
  - Windows Mobile
  - OpenMoko
- Commonly used implementation techniques
- Summary

OpenMoko
(www.openmoko.org)
- An open source project dedicated to delivering mobile phones software stack
  - Linux, etc. What could be expected based on Maemo and Android
- Currently selling the Neo FreeRunner phone to advanced users and will start selling it to the general public as soon as the software is more developed
- In general, seems comparable to Maemo, except that geared towards implementation of a phone, not Internet tablet

5. More on app development

- Some other mobile development systems
  - MIDP Java
  - Maemo
  - Android
  - iPhone
  - Windows Mobile
  - OpenMoko
- Commonly used implementation techniques
- Summary

Improving portability with VM

- Possibility to implement the same system on top of different infrastructure
  - Same environment can be used in both development and execution environment
- Some tedious and error-prone tasks can be allocated to the virtual machine instead of the programmer
  - Memory management in particular
  - Support for recovering from errors
- Simplifies the implementation of security features
  - There already is a central element that is responsible for executing applications

Unifying stimuli with event-based programming

<table>
<thead>
<tr>
<th>service caller</th>
<th>service provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>event source</td>
<td>event observers</td>
</tr>
</tbody>
</table>

Benefits

- Commonly used technique in UI programming
- Serialization of operations triggered to the execution; events can be handled one by one
- Simplified model of execution
  - No need to define the order of execution in user programs; it is enough to generate an event
- Simplified introduction of new event generators
- Simplified introduction of new event observers
- Long-lasting operations form a problem
  - Commonly initiate a new event that continues processing after a while
    - Introduces additional complexity in the form of a state machine
Separating device specific parts using MVC pattern

The model should gather all common aspects. In practice, many systems force commonalities to views and controllers.

Model-View-Controller, MVC

MVC Execution

Interfacing

- Access to device specific resources vs. access to a sandbox
  - Security considerations unavoidable
- Access to device specific widgets vs. access to a generic canvas on top of which can be rendered
  - All apps similar in a device vs. App similar in all devices
- Managed code for applications vs. using native applications

5. More on app development

- Some other mobile development systems
  - MIDP Java
  - Maemo
  - Android
  - iPhone
  - Windows Mobile
  - OpenMoko
- Commonly used implementation techniques
- Summary

Conclusions

- Numerous approaches to programming mobile devices exist
- Commonalities
  - Virtual machines to unify the development approach in several different configurations
  - Event-based programming for user input
  - Separating UI specifics using MVC
  - Auxiliaries packaged in installation packages