Mobile Software Technologies (SW8)

J2ME Platform

Hua Lu
Department of Computer Science
Aalborg University
Spring 2008
J2ME Platform

- Introduction
- CLDC and MIDP
- Developing J2ME Applications
  - IDEs
  - Security
  - MIDP GUI and Event Handling
  - Networking
  - Databases
  - MIDP Timer and Task Scheduling
  - Performance Tuning
Java: “Write Once Run Anywhere”

• But
  - Different devices have different requirements.
  - Mobile devices do not have the same environment as regular computers (standard desktop), the constrains we have:
    ✷ Limited memory and processor.
    ✷ Small screen sizes.
    ✷ Alternative input methods.
  - One platform (solution) cannot address all the market segments (web server, desktops, PDA, mobile phones …). Developers want to choose what they want to use and what they don’t.

• Sun decided to develop a special edition of Java J2ME (Java 2 Micro Edition).
Java 2 Platform Architecture
J2ME Core Concepts

• At the heart of Java 2 Micro Edition (J2ME) are three core concepts
  ▪ Configurations
  ▪ Profiles
  ▪ Optional packages

• You can't write a J2ME application without understanding these concepts, because they determine the features of Java that you can use, which application programming interfaces (APIs) are available, and how your applications are packaged.
Configurations and Profiles

• “A Configuration is a specification that defines a minimum Java Platform functionality for a *family of devices*. It defines the minimum number of Java libraries, VM capabilities, Security specification, and a Generic Connection Framework.”

• “A Profile is a collection of Java APIs that supplement a Configuration to provide capabilities for a specific device group or market type.”
J2ME Core Components

- **Optional Packages**
- **Profile**
  - A collection of Java Classes selected from one or more Java core, extension or vertical APIs. Classes are chosen to provide a complete solution for a specific vertical market
- **Configuration**
  - A subset of the Java core APIs and Java language functionality selected to provide a minimal Java platform for a set of vertical markets
J2ME Platform

• Introduction
• CLDC and MIDP
• Developing J2ME Applications
  - IDEs
  - Security
  - MIDP GUI and Event Handling
  - Networking
  - Databases
  - MIDP Timer and Task Scheduling
  - Performance Tuning
Available J2ME Configurations

- **Connected Device Configuration (CDC)**
  - Used by devices that are more powerful than the low end devices such as cell phones
  - Example set-top boxes, certain PDAs, …

- **Connected, Limited Device Configuration (CLDC)**
  - Used for very resource constrained devices
  - Example, cell phones, two way pagers, …
CLDC

• CLDC Java Virtual Machine
  - Sun’s Kilobyte Virtual Machine (KVM)
  - Designed for small resource-constrained devices
  - Small memory footprint (60K) for KVM
  - Minimum total memory 160K
  - 16/32-bit processors
  - Processor speed 8 to 32 MHZ

• CLDC Hardware
  - Memory size is the only requirement for CLDC
  - At least 128K non-volatile for the KVM and CLDC Libraries
  - At least 32K of volatile for KVM runtime
J2ME Profiles

• Defines requirements for a specific vertical market family of devices
  ■ Extends or is Layered on top of a configuration
  ■ Defines a standard Java platform for a vertical market device family, to ensure interoperability
  ■ Includes more granular domain specific class libraries that a configuration
Available J2ME Profiles

- **Mobile Information Device Profile (MIDP)**
  - Delivers an enhanced user interface, multimedia and game functionality, end-to-end security, and greater networked connectivity to mobile phones and entry level PDAs

- **Foundation Profile**
  - Set of Java APIs that support resource-constrained devices without a standards-based GUI system

- **Personal Profile**
  - Together with CDC and Foundation Profile, Personal Profile provides a complete application environment for the high-end PDA market. Personal Profile contains the full set of AWT APIs, including support for applets and Xlets.

- **Personal Basis Profile**
  - Provides a J2ME application environment for network-connected devices supporting a basic level of graphical presentation
MIDP Hardware

- Memory (added to CLDC memory)
  - 128K non-volatile for MIDP components
  - 8K non-volatile for application persistent data
  - 32K volatile for KVM
- Display
  - Screen 96x54
  - Display depth 1-bit
  - Pixel shape (aspect ratio) 1:1
- Input (one or more)
  - One-handed keypad
  - Two-handed keypad
  - Touch screen
- Networking
  - Two-way
  - Wireless
  - Possibly intermittent
  - Limited bandwidth
MIDP Architecture

- Mobile Information Device (MID)
- CLDC
- MIDP
- OEM-Specific Classes
- OEM-Specific Applications
- Native Applications
- Native System Software/Host Operating System
- Mobile Information Device (MID)
- OEM-Specific Applications

MIDP Applications
Class Libraries

- **CLDC**
  - java.lang
  - java.io
  - java.util
  - javax.microedition.io

- **MIDP**
  - javax.microedition.lcdui
  - javax.microedition.midlet
  - javax.microedition.rms
MIDP Specification

- There are two versions of the MIDP:
  - *MIDP 1.0* - is the original specification, provides core application functionality required by mobile applications, including basic user interface and network security
  - *MIDP 2.0* - is a revised version of the MIDP 1.0. Have new features include an enhanced user interface, multimedia and game functionality, more extensive connectivity, over-the-air provisioning, and end-to-end security.
MIDP Application Model

- MIDlet is the basic application
  - Similar to the J2SE applet
  - GUI based
- MIDlet Suites – security for applications that share resources or data
MIDP Application Lifecycle

- MIDP applications are known as “MIDlets”
- MIDlets move from state to state in the lifecycle
  - Start – acquire resources and start executing
  - Pause – release resources and become quiescent (wait)
  - Destroy – release all resources, destroy threads, and end all activity
Tour of CLDC/MIDP

• Packages
  - javax.microedition.io
  - javax.microedition.lcdui
  - javax.microedition.midlet
  - javax.microedition.rms

• Contains user interface widgets
  - Form, TextField, TextBox, DateField, List, Image, Gauge, Alert, Canvas, Graphics, Display

• Event handling classes – Command and CommandListener

• Two APIs, high and low
  - High level for GUI widgets, scrolling, etc.
  - Low level for graphics and fine-grained UI control
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class HiMIDlet
    extends MIDlet {

    private TextBox textbox;

    public HiMIDlet() {
        textbox = new TextBox ("", "Hello World!", 20, 0);
    }

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

    public void pauseApp() {}

    public void destroyApp(boolean unconditional) {}
J2ME Platform

- Introduction
- CLDC and MIDP
- Developing J2ME Applications
  - IDEs
  - Security
  - MIDP GUI and Event Handling
  - Networking
  - Databases
  - MIDP Timer and Task Scheduling
  - Performance Tuning
Creating J2ME Applications

- Identify the devices
- Identify the profiles supported by devices
- Develop the Application
- Using the MID profile, the application will target cell phones and pagers
- Write an application that displays the String “Hello, Small World!”
- Requires the environment; MIDP reference implementation http://java.sun.com/products/midp/
- Will run in the MIDP emulator
Developing J2ME Wireless Applications

• Use ktoolbar to create a project
• Configure the Midlet
• Build and Run the project
• MIDlet class, the application-level class
• Abstract, need to define methods
  ■ startApp()
  ■ pauseApp()
  ■ destroyApp(boolean unconditional)
J2ME Platform

- Introduction
- CLDC and MIDP
- Developing J2ME Applications
  - IDEs
  - Security
  - MIDP GUI and Event Handling
  - Networking
  - Databases
  - MIDP Timer and Task Scheduling
  - Performance Tuning
Sun’s J2ME Wireless Toolkit (1)

• To run the toolkit itself, select the **KToolbar** shortcut. You should see the following screen.
Sun’s J2ME Wireless Toolkit (2)

• The J2ME Wireless Toolkit works with projects, where the end result of each project is one MIDlet suite. The toolkit works with one project at a time. You can change properties of the current project, build the project, and run the project in a device emulator. Several example projects come installed with the toolkit; we'll look at these later.

• Let's jump right in the water by creating a new project. Click on New Project in the button bar. The toolkit prompts you for a project name and the name of a MIDlet class in the project. Fill in HelloSuite and HelloMIDlet as shown below.
Nokia J2ME SDK
NetBeans IDE (1)
NetBeans IDE (2)

- NetBeans Mobility Pack
  - used to write, test, and debug applications for the Java Micro Edition platform (J2ME) technology-enabled mobile devices. It integrates support for the Mobile Information Device Profile (MIDP) 2.0, the Connected, Limited Device Configuration (CLDC) 1.1.
  
- The mobility pack allows for the unique “On-Phone” debugging mode.
MIDlet Suite

• One or more MIDlets are packaged together into a *MIDlet suite*, composed of:
  
  ■ JAR (Java archive) file - The JAR file contains Java classes for each MIDlet in the suite and Java classes that are shared between MIDlets. The JAR file also contains resource files used by the MIDlets and a manifest file.
  
  ■ JAD (Java Application Descriptor) file - This 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 upload to the machine in order to run the application.
J2ME Platform

• Introduction
• CLDC and MIDP
• Developing J2ME Applications
  ■ IDEs
  ■ Security
  ■ MIDP GUI and Event Handling
  ■ Networking
  ■ Databases
  ■ MIDP Timer and Task Scheduling
  ■ Performance Tuning
Application Loading Process

Advertise App on Web Page

User Selects App

JAM Downloads App

Web Page

Choose a game:
Asteroids
Backgammon
Battleship
Black Jack
Breakout
Missile

Confirm:
Buy (1)
Breakout
game for:
$2.95

Installing...

Descriptor File

Network Transfer

Java Application Manager

Descriptor File

Jar File

(Name, Version, Size, ...)

Network Transfer
Security

• J2ME Cannot support full J2SE security model
  ▪ J2SE protection domain model is larger than the entire CLDC implementation

• So how’s security implemented in J2ME?

• Split the security model into two parts
  ▪ Low-level-virtual machine security
    ◆ Guaranteed by the CLDC Two-Phase class file verifier
  ▪ Application level security
    ◆ Sandbox model
Low-Level VM Security

• Two-phase class verification

• Off-device verification
  ■ Pre-verification tool add “stack map” attribute to each method in Java class file
  ■ Pre-verification is performed on server or desktop system before class file is downloaded to the device
  ■ This “stack map” helps to facilitate in-device verification
    ◆ Faster verification process
    ◆ Less VM code and memory consumption

• In-device verification
  ■ Environment verification (e.g. memory req. to run app.)
  ■ Byte code check
  ■ Type check
  ■ Proper termination (no unconditional Jump instruction)
Stack map attribute increases the size of a classfile by approx. 5%
Application-Level Security

• Sandbox Model

• Application must run in closed environment in which app can only access those APIs defined by Configuration, Profiles and licensee open classes supported by the device

• More specifically sandbox model means:
  - Class files have been properly verified and guaranteed to be valid
  - Ensures that only the mandated Java APIs are available to applications as defined by CLDC, Profile and licensee’s extension classes
  - Application download occurs at the native code level – JAM
  - Cannot override class loading mechanism
  - Set of native functions accessible to the VM is closed
J2ME Platform

• Introduction
• CLDC and MIDP
• Developing J2ME Applications
  ■ IDEs
  ■ Security
  ■ MIDP GUI and Event Handling
  ■ Networking
  ■ Databases
  ■ MIDP Timer and Task Scheduling
  ■ Performance Tuning
MIDP UI Programming Model

• The central abstraction is a screen
• Only one screen may be visible at a time

• Three types of screens:
  - Predefined screens with complex UI components (List, TextBox)
  - Generic screens (Form where you can add text, images, etc)
  - Screens used with low-level API (Canvas)
MIDP UI and Display

• The *Display* class is the display manager
• It is instantiated for each active MIDlet
• Provides methods to retrieve information about the device’s display capabilities
• A screen is made visible by calling
  - `Display's setCurrent(screen);`
MIDP UI Classes

- **javax.microedition.lcdui classes**
  - Alert, AlertType, Canvas, ChoiceGroup, Command, DateField, Display, Displayable, Font, Form, Gauge, Graphics, Image, ImageItem, Item, List, Screen, StringItem, TextBox, TextField, Ticker

- **javax.microedition.lcdui interfaces**
  - Choice, CommandListener, ItemStateListener
MIDP UI Class Diagram

- Major classes and interfaces
High-Level API Examples (1)

• List

Display display = Display.getDisplay(this);
List menu = new List(“Method of payment”, Choice.EXCLUSIVE);
menu.append(“Visa”);
menu.append(“MasterCard”);
menu.append(“Amex”);
display.setCurrent(menu);
High-Level API Examples (2)

• Form (Date/Time info)
  DateField date = new DateField("Today’s date", DateField.TIME);
  Form form = new Form("Date Info");
  form.append(date);
  display.setCurrent(form);
High-Level Examples (3)

- Form (Sign in screen)
  
  ```java
  Display display = Display.getDisplay(this);
  TextField userName = new TextField("LoginID:", "", 10, TextField.ANY);
  TextField password = new TextField("Password:", "", 10, TextField.PASSWORD);
  Form form = new Form("Sign in");
  form.append(userName);
  form.append(password);
  display.setCurrent(form);
  ```
Low-level Example (1)

• Canvas
  
  public class MyCanvas extends Canvas
  {
    public void paint(Graphics g)
    {
      g.setColor(255, 0, 0);
      g.fillRect(0, 0, getWidth(), getHeight());
      g.setColor(255, 255, 255);
      g.drawString("Hello World!", 0, 0, g.TOP | g.LEFT);
    }
  
  }
Low-level Example (2)

• Instantiate and display MyCanvas

```java
public class MyMidlet extends MIDlet
{
    public MyMidlet() {
        // constructor
    }
    public void startApp()
    {
        Canvas canvas = new MyCanvas();
        Display display = Display.getDisplay(this);
        display.setCurrent(canvas);
    }
    // pauseApp() and destroyApp()
}
```
Input Handling

• High-Level API input is handled using abstract commands
  - No direct access to soft buttons
  - Commands are mapped to appropriate soft buttons or menu items
Input Handling: Example

• **TextBox screen with commands**
  
  ```java
  Display display = Display.getDisplay(this);
  TextBox tb = new TextBox("MIDP", "Welcome to MIDP Programming", 40, TextField.ANY);
  Command exit = new Command("Exit", Command.SCREEN, 1);
  Command info = new Command("Info", Command.SCREEN, 2);
  Command buy = new Command("Buy", Command.SCREEN, 2);
  tb.addCommand(exit);
  tb.addComment(info);
  tb.addCommand(buy);
  display.setCurrent(tb);
  ```

  ![Example of a TextBox screen with commands](image)
Event Handling: High-level

• High-level Events:
  ■ Based on a listener model
  ■ Screen objects can have listeners for commands
  ■ For an object to be a listener, it must implement the CommandListener interface
  ■ This interface has one method: commandAction
Event Handling Example (1)

• MIDlet implements CommandListener

```java
public class MyMIDlet extends MIDlet implements CommandListener {
    Command exitCommand = new Command(…);
    // other stmts
    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```
Event Handling Example (2)

- Handling List events

```java
public void commandAction(Command c, Displayable d) {
    if (c == exitCommand) { .. }
    else {
        List down = (List)display.getCurrent();
        switch(down.getSelectedIndex())
        { 
            case 0: testTextBox();break;
            case 1: testList();break;
            case 2: testAlert();break;
            case 3: testDate();break;
            case 4: testForm();break;
        }
    }
}
Event Handling: Low-level

- Low-level Events:
  - Low-level API gives developers access to key press events
  - Key events are reported with respect to key codes
  - MIDP defines key codes: KEY_NUM0 .. KEY_NUM9, KEY_STAR, KEY_POUND
Handling Events: Example

- **Low-level events**

```java
protected void keyPressed(int keyCode) {
    if (keyCode > 0) {
        System.out.println("keyPressed " + ((char)keyCode));
    } else {
        System.out.println("keyPressed action " + getGameAction(keyCode));
    }
}
```
MIDP UI Design Principles

- Make the UI simple and easy to use
- Use the high-level API (portability)
- If you need to use low-level API, keep to the platform-independent part
- MIDlets should not depend on any specific screen size
- Entering data is tedious, so provide a list of choices to select from
J2ME Platform

- Introduction
- CLDC and MIDP
- Developing J2ME Applications
  - IDEs
  - Security
  - MIDP GUI and Event Handling
  - Networking
  - Databases
  - MIDP Timer and Task Scheduling
  - Performance Tuning
Networking

• J2SE and J2EE networking APIs are not suitable for handheld devices
  ■ Require several megabytes of memory to run
  ■ Device manufacturers who work with circuit-switched networks require TCP support
  ■ Device manufacturers who work with packet-switched networks require UDP support
  ■ Other devices have specific mechanisms for communications
CLDC Generic Connections

- A set of related abstractions at the programming level
- No abstractions for different forms of communications
- All connections are created using the `Connector.open()`
- If successful, it returns an object that implements one of the generic connection interfaces
Connection Interfaces
Example Connections

- HTTP
  - Connector.open("http://www.host.com");

- Socket
  - Connector.open("socket://host.com:80");

- Datagram
  - Connector.open("datagram://address:port");

- File
  - Connector.open("file:/myfile.txt");
MIDP Connectivity

• It provides support for HTTP (HttpConnection)
• Why?
  ■ Because HTTP can be implemented using IP protocols or non-IP protocols
HttpConnection

- Part of the javax.microedition.io
- Defines the necessary methods and constants for an HTTP connection

```java
HttpConnection c = (HttpConnection) Connector.open("http://quotes.yahoo.com");
C.setRequestMethod(HttpConnection.POST);
C.setRequestProperty("Content-Language", "en-CA");
```
Invoking Remote Applications

• A MIDlet may invoke remote applications
  ■ Fetching a page
  ■ Invoking a CGI script (GET or POST method)
  ■ Invoking a Servlet

• Example: Invoke a CGI script using GET method

```java
String url = "http://host/cgi-bin/getgrade?idnum=182061";
c = (HttpConnection) Connector.open(url);
c.setRequestMethod(HttpConnection.GET);
// set some request properties: c.setRequestPropert(" ", " ");
is = c.openDataInputStream();
while((ch = is.read()) != -1) {
    b.append((char)ch);
}
```
Another Example

• If you want to send data to a remote application

```java
String s = “stuffToSend”; byte postmsg[] = s.getBytes(); for(int i=0;i<postmsg.length;i++) {
    os.writeBytes(postmsg[i]);
} // OR
os.write(s.getBytes());
```
J2ME Connection Types

- **HTTP/HTTPS**
  - Very simple to use

- **XML over HTTP/HTTPS**
  - Beginning to be a viable choice due to varying parsers available
    - **Model**
      - Creates an object representation of a document in memory (e.g., DOM)
    - **Push**
      - Parses through an entire document, spitting out events to registered listeners (e.g., SAX)
    - **Pull**
      - Parses a little at a time, returning a single element or tag

- **MIDP 2.0**
  - The only addition to MIDP 2.0 for connectivity was the requirement to support HTTPS.
Wireless Web Services APIs (1)

- J2ME Web Services (JSR 172)
  - Public review
    - No API available yet
  - J2ME Web Services specification goals
    - Access remote SOAP/XML based web services
      - Does not support web service hosting capabilities
      - JAX-RPC subset
        - Must support stub based invocation (stub created using WSDL)
        - No support yet for Dynamic proxies and Dynamic Invocation Interface (DII)
        - No asynchronous messaging with attachment support
        - Not required to support Java beans
    - Provides parsing XML data
      - JAXP subset
        - Must support SAX 2.0 (with exceptions)
        - Must not support DOM
Wireless Web Services APIs (2)

- SOAP and J2ME
  - J2ME has only limited string functionalities, a problem for every wireless Java XML parser
  - J2ME/CLDC lacks support for Float datatypes
  - SOAP parsing requires to read the whole document

- kSOAP & kXML
  - kSOAP
    - API with Small footprint
    - kSOAP is open source and based on kXML
    - Adds support for data types not supported by J2ME
  - kXML
    - Non validating XML pull parser
J2ME Platform

• Introduction
• CLDC and MIDP
• Developing J2ME Applications
  ■ IDEs
  ■ Security
  ■ MIDP GUI and Event Handling
  ■ Networking
  ■ Databases
  ■ MIDP Timer and Task Scheduling
  ■ Performance Tuning
Databases

• A persistent storage
  - A place to store the state of objects
• Facilities provided in J2SE and J2EE are not suitable for handheld devices
• MIDP provides a record-oriented database mechanism to persistently store data and retrieve it later
  - Record Management System, RMS
MIDP’s RMS

- Lightweight record-oriented database
  - Device independent API
  - Unique recordID for each record within the store
  - A record is an array of bytes
  - Shared within MIDlet suite
  - Support for enumeration, sorting, and filtering

- Package
  - javax.microedition.rms
MIDP RMS Methods

- **Record Store**
  - openRecordStore, closeRecordStore, listRecordStore, deleteRecordStore, getRecordSize, getNumRecords

- **Record Data**
  - addRecord, deleteRecord, getRecord, setRecord, getRecordSize

- **Record Selection**
  - RecordEnumeration, RecordFilter, RecordComparator

- **To open a record store**
  - RecordStore db = RecordStore.openRecordStore("myDB", true);

- **To close a record store**
  - db.closeRecordStore();
Create/Add a New Record

- To create a new record

```java
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutputStream dos = new DataOutputStream(baos);
dos.writeUTF(record);
Byte b[] = baos.toByteArray();
db.addRecord(b, 0, b.length);
```
J2ME Platform

• Introduction
• CLDC and MIDP
• Developing J2ME Applications
  ■ IDEs
  ■ Security
  ■ MIDP GUI and Event Handling
  ■ Networking
  ■ Databases
  ■ MIDP Timer and Task Scheduling
  ■ Performance Tuning
MIDP Timers

- Handle queuing and delivery

- Timer task
  - Multiple tasks per timer
  - Periodic
  - Fixed interval
  - One-time execution

- To define a task, create a subclass of TimerTask
  ```java
  import java.util.*;
  public class MyTask extends TimerTask {
      public void run() {
          System.out.println("Run Task");
      }
  }
  ```
Schedule Tasks

• Schedule a task for execution by creating a Timer object and invoking schedule()

    Timer timer = new Timer();
    TimerTask task = new MyTask();
    // wait five seconds before executing
    timer.schedule(task, 5000);
    // wait two seconds before executing then
    // execute every five seconds
    timer.schedule(task, 2000, 5000);
J2ME Platform

• Introduction
• CLDC and MIDP
• Developing J2ME Applications
  ■ IDEs
  ■ Security
  ■ MIDP GUI and Event Handling
  ■ Networking
  ■ Databases
  ■ MIDP Timer and Task Scheduling
  ■ Performance Tuning
Performance Tuning (1)

- **String vs. StringBuffer**

```java
String s = new String();
for (int i=0; i<10000; i++)
    s += "a";
```

```java
StringBuffer s = new StringBuffer();
for (int i=0; i<10000; i++)
    s.append("a");
```

1162 ms

10 ms

Origin from Java Two Conference Session "Java Performance Tuning"
• **String Object Overpopulation**

```java
ucA = a.toUpperCase();
ucB = b.toUpperCase();

boolean bool = ucA.equals(ucB);
```

- **331 ms**

```java
boolean bool = a.equalsIgnoreCase(b);
```

- **50 ms**

Origin form Java Two Conference Session “Java Performance Tuning”
Performance Tuning (3)

- **String Equality**

  - `String p = "Hello";`  
  - `String q = "Hello";`  
  - `p.equals(q);`

  ![140 ms]

  - `String p = "Hello";`  
  - `String q = "HEllo";`  
  - `p.equals(q);`

  ![1700 ms]

  - `String p = "Hello";`  
  - `String q = "Hello!";`  
  - `p.equals(q);`

  ![650 ms]

  Origin from Java Two Conference Session "Java Performance Tuning"
public static void copy (String from, String to) throws IOException
{
    InputStream in = null;
    OutputStream out = null;
    try {
        in = new FileInputStream(from);
        out = new FileOutputStream(to);

        while (true) {
            int data = in.read();
            if (data == -1)
                break;
            out.write(data);
        }
        in.close();
        out.close();
    } finally {
        if (in != null)
            in.close();
        if (out != null)
            out.close();
    }
}
Performance Tuning (5)

- I/O Performance

Origin from Java Two Conference Session "Java Performance Tuning"
Performance Tuning (6)

- I/O Performance

```java
public static void copy (String from, String to) throws IOException {
    try {
        InputStream in = null;
        OutputStream out = null;
        InputStream inFile = new FileInputStream(from);
        in = new BufferedInputStream(inFile);
        OutputStream outFile = new FileOutputStream(to);
        out = new BufferedOutputStream(outFile);

        while (true) {
            int data = in.read();
            if (data == -1)
                break;
            out.write(data);
        }
        in.close();
        out.close();
    } finally {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
    }
}
```
Performance Tuning (7)

• I/O Performance

Origin form Java Two Conference Session “Java Performance Tuning”
Performance Tuning (8)

- **Vector and HashMap**
  - When using vector or hashmap, please estimate the size at first
    - Vector(int initialCapacity, int capacityIncrement)
  - When using finished, please clear the vector or hashmap
    - Collection.clear()
    - Vector.setSize(0)