Java for Small Devices

Java 2 Microedition

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
  - Configurations
  - Profiles

The Mobile World

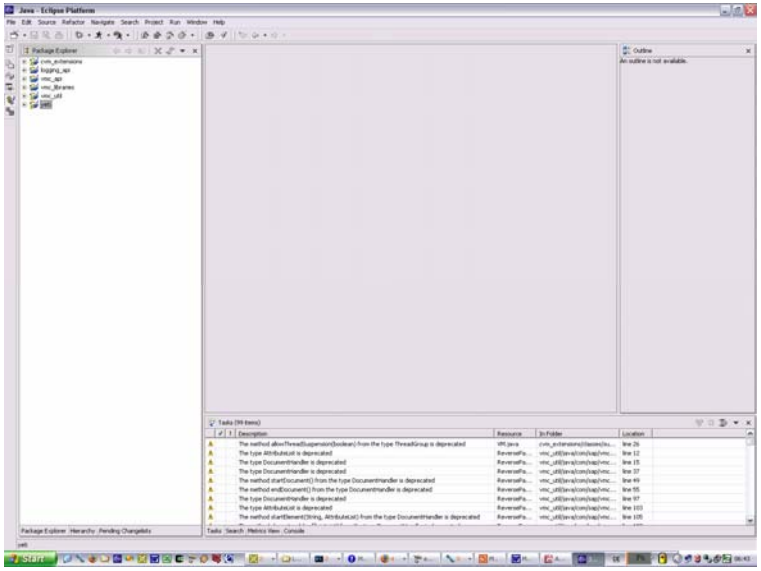
- **Motivation for Applications on Mobiles**
  - Ubiquitous computing
  - Stay connected (with business or friends)
  - Diversify communication methods
  - Retrieve Information
  - Personal Information Management (PIM)

- **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**

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

  - 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

	GSM	GPRS	UMTS	WLAN
Switched	Circuit Switched	Packet Switched	Packet Switched	Packet Switched
Bandwidth	14,4 kBit	56 kb/s	2 MBit	54 Mbit

- **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.)

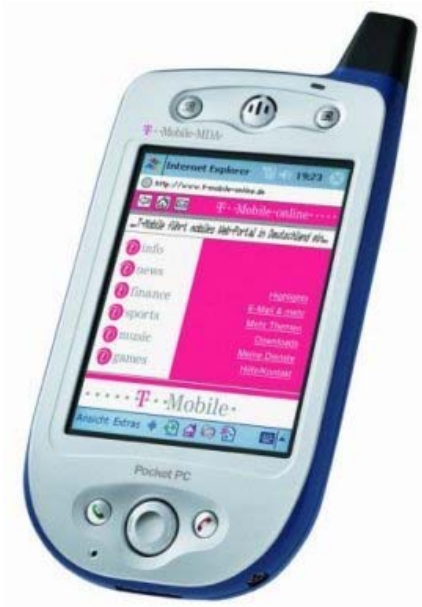
- **Demand for synchronization**
  - Limited connectivity, bandwidth and power
  - Mobile not be accepted as storage platform for information
  - → Information needs to be stored locally which requires synchronization
  - Synchronization demand crosses application domains
    (e.g. Mail & News, Contacts, 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 (Micro Framework, Compact Framework, Smart phone)**
  - VM Approach, Managed code
- **Embedded Linux (e.g. Montavista Linux)**
  - Native Software
- **PalmOS (Access Garnet OS)**
  - Native Software, 68K emulation on ARM architecture

Java™ ME – Java (2) Platform, Micro Edition

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

- **Java ME is a framework**
  - formerly J2ME
  - Designed to support different device classes like

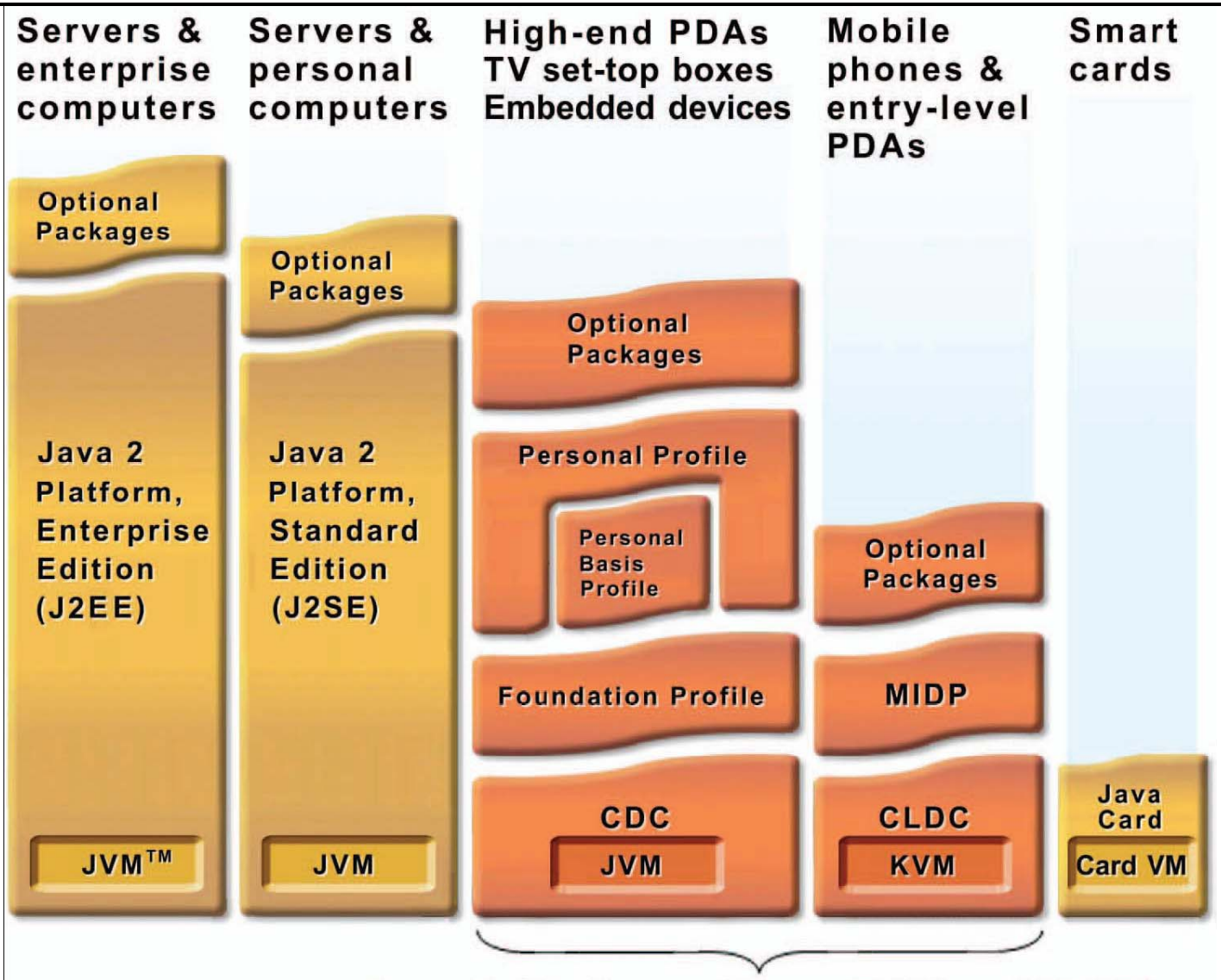
    Parking meter,
    Mobile phone,
    Watch

    These impose different requirements

  - Thus a Java ME runtime environment consists of multiple building blocks

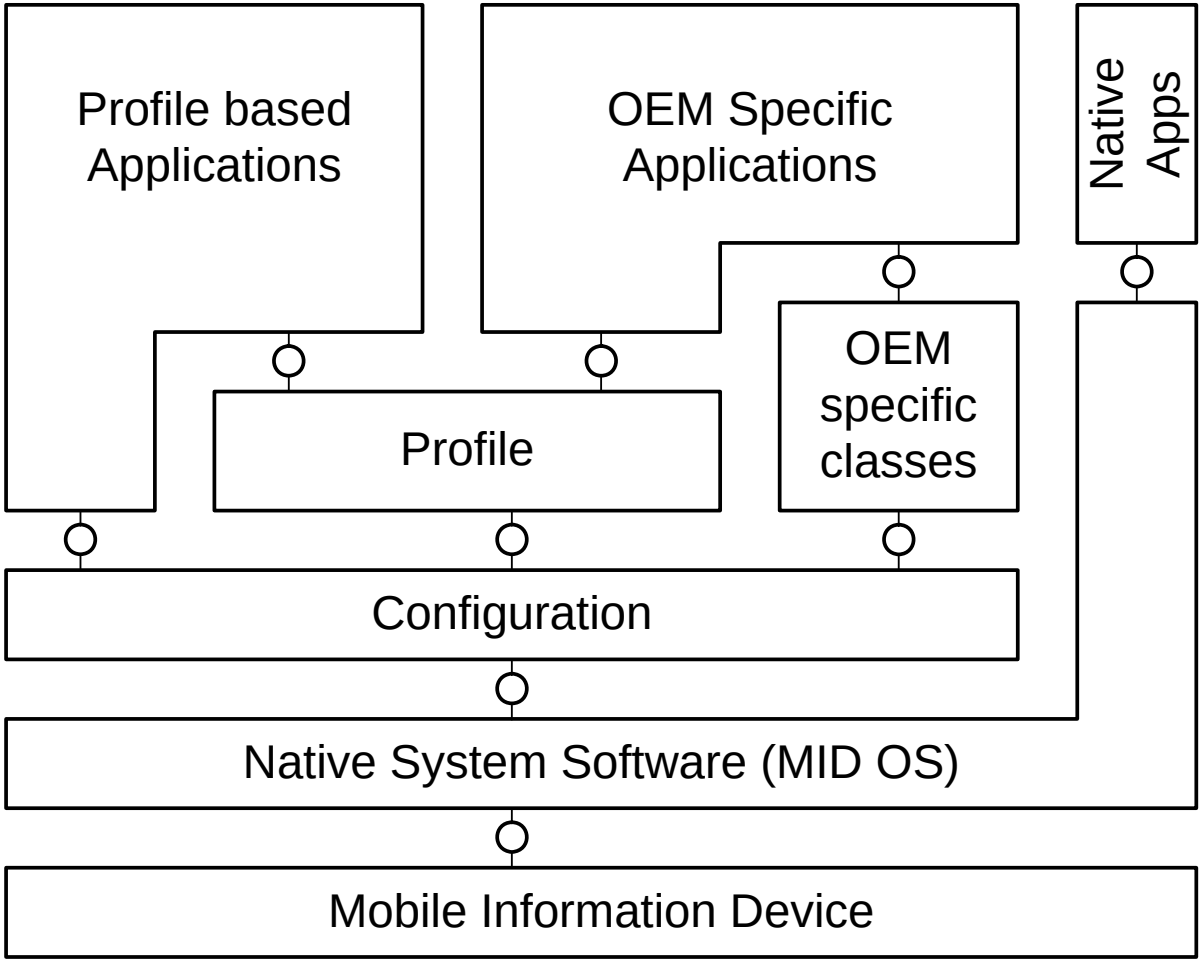
- **http://java.sun.com/javame**

Java ME - Platform



Java Platform, Micro Edition (Java ME)

Java ME – Architecture

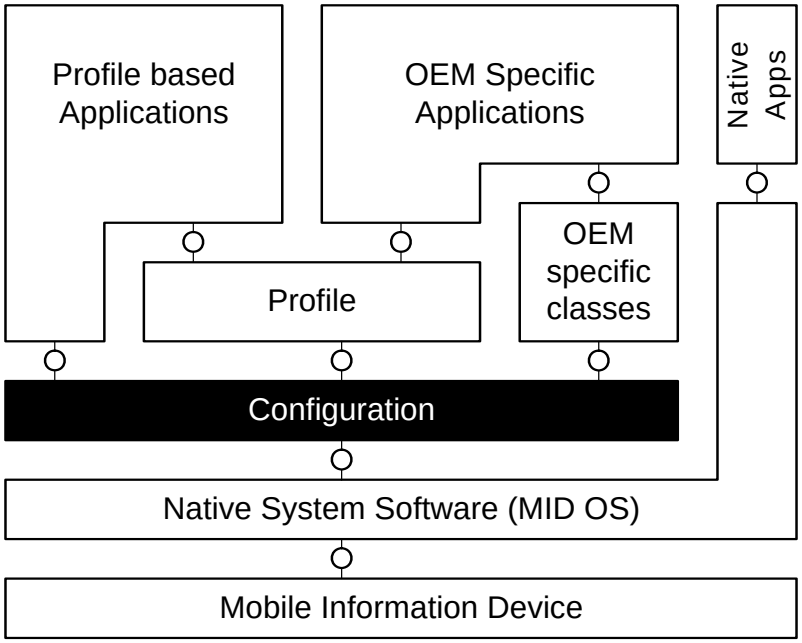


- **Java ME environment:**
  - Profile + Configuration + Virtual Machine = Java Technology Stack

Java ME – Configuration

Configuration

- "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." CLDC 1.1 (JSR 139)



J2ME – Configuration cont.

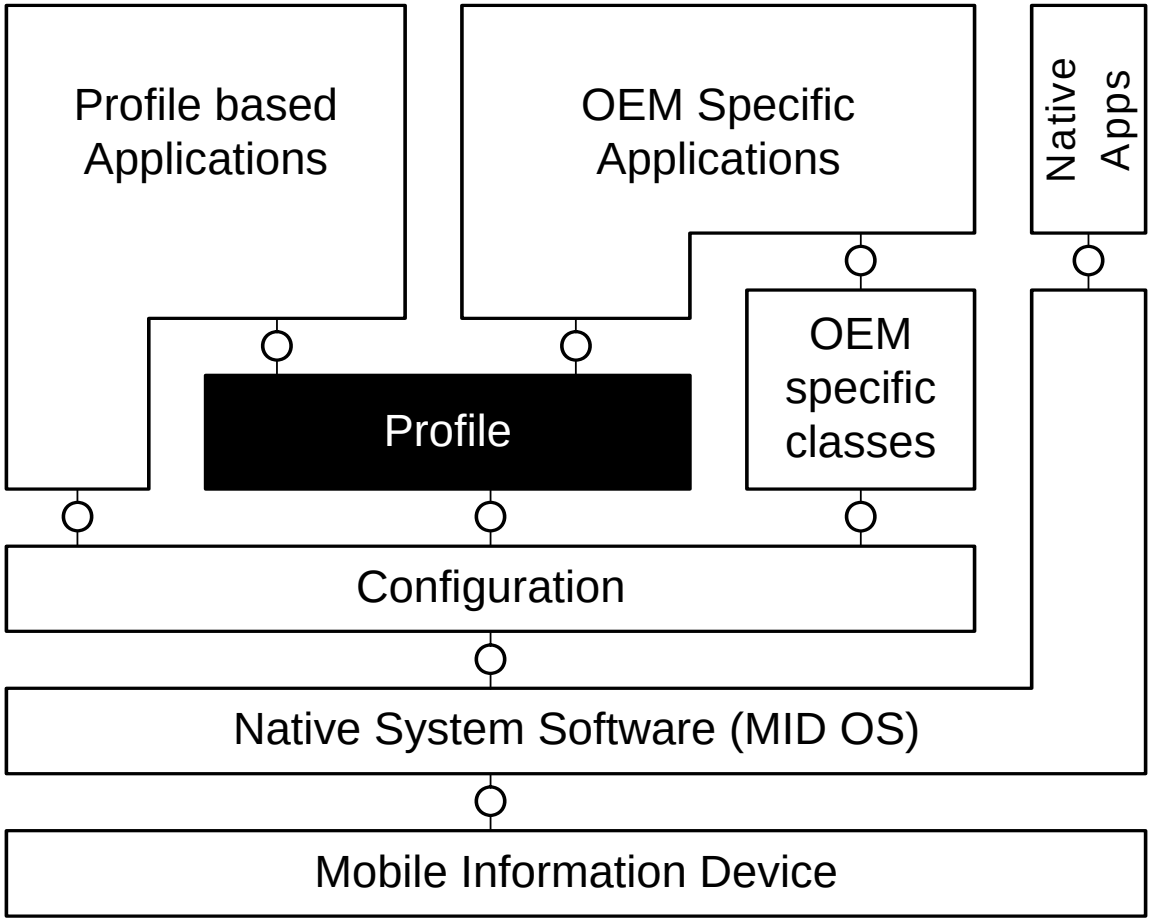
- **Defines:**
  - Application Management (Retrieving, Loading & Executing)
  - Low Level Security Measures (e.g. Byte Code Verification)
  - Available Byte Code Instructions (Usually Subset of J2SE)
  - Class File Format
  - Mandatory Libraries
    - This is just a basic set, additional libraries are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

- **May not define *optional* features!!**

- **Example Configuration:**
  CLDC (Connected Limited Device Configuration)

J2ME – Profile

- **Profile**
  - Based on a Configuration



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)

J2ME –Configurations

- **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)
  - **J**ava **C**ommunity **P**rocess: http://www.jcp.org

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

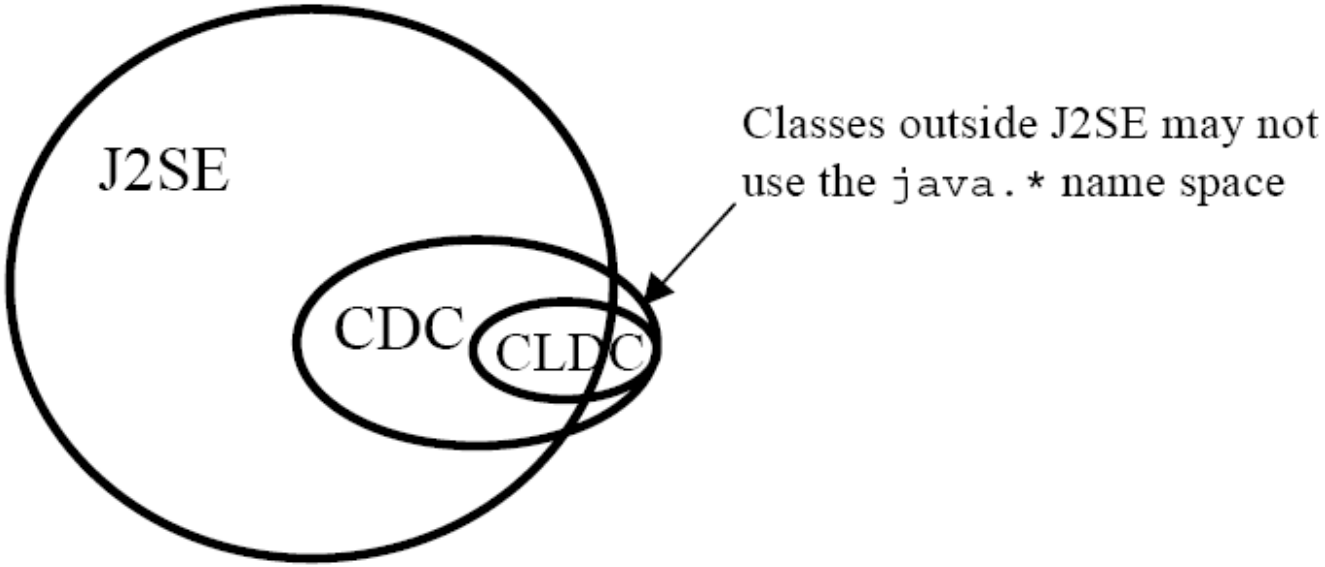
Application Management

- 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

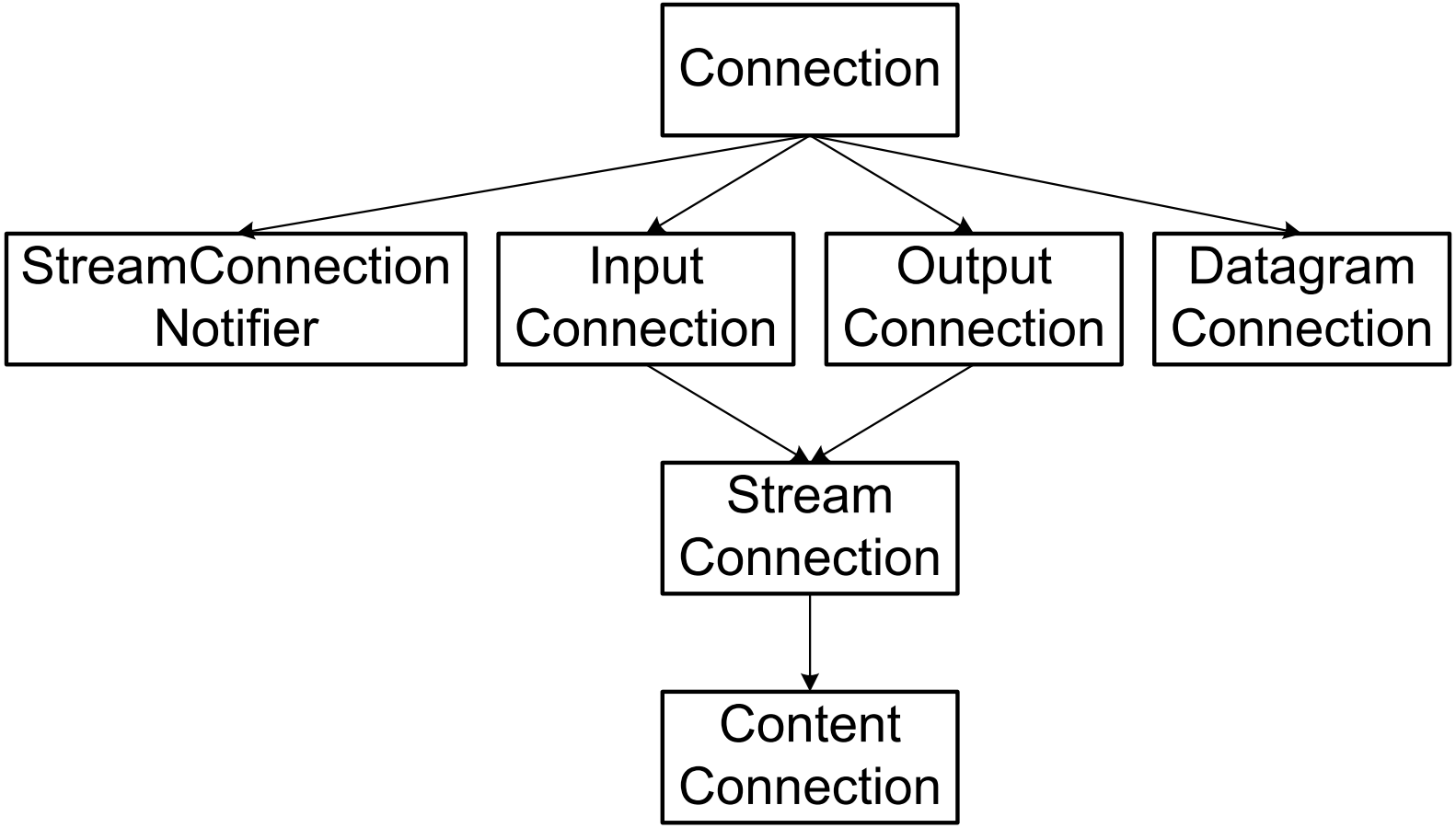
- **Subset of J2SE**
  - No floating point support
    - Byte code instructions removed
    - was reintroduced in CDLC 1.1
  - No Java Native Interface (JNI, calling of native c-functions)
  - No user defined class loaders (security)
  - No reflection
  - No Thread groups
    - Multithreading is supported,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

J2ME – J2SE, CDC, CLDC



J2ME – CLDC – Connection Framework

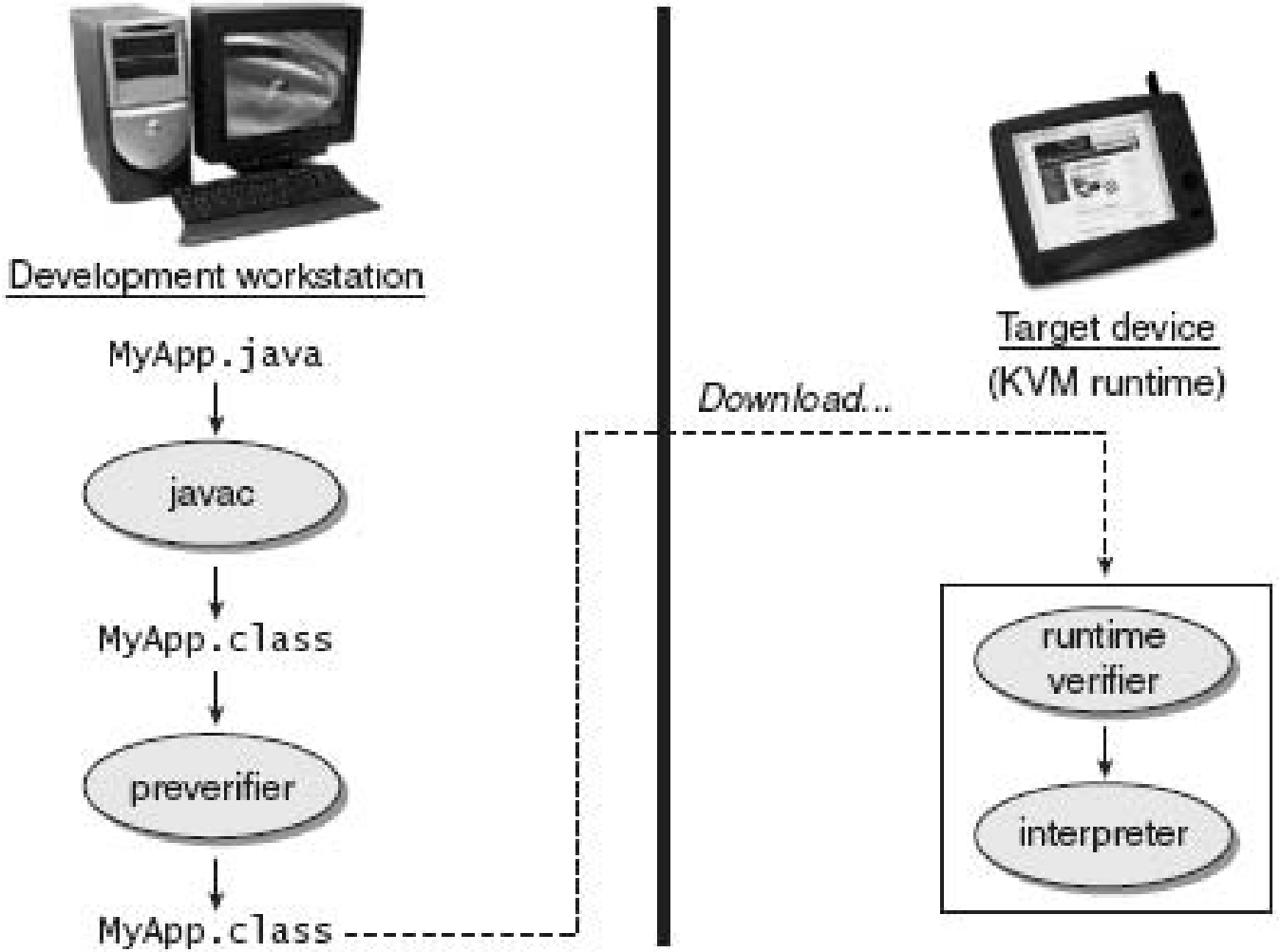
- **Connection Framework of CLDC**



J2ME – CDLC – Connection Framework

- **Key ingredient of CLDC**
- **`Connector.open("<protocol>:<address>;<parameters>");`**
  - May potentially be used for
    - HTTP
    - Bluetooth,
    - RS232 `comm:com0;baudrate=19200`
    - TCP, `socket://host:port`
    - UDP `datagram://host:port`
    - Any other unidirectional `sms://+4917….. , mms://+4917...` 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



J2ME – CDLC – Security cont.

- **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 (5-15 percent →CLDC 1.1)
    - 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)
  - MIDP 3.0 (JSR 271, In Progress)
- 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

MIDlet Life Cycle

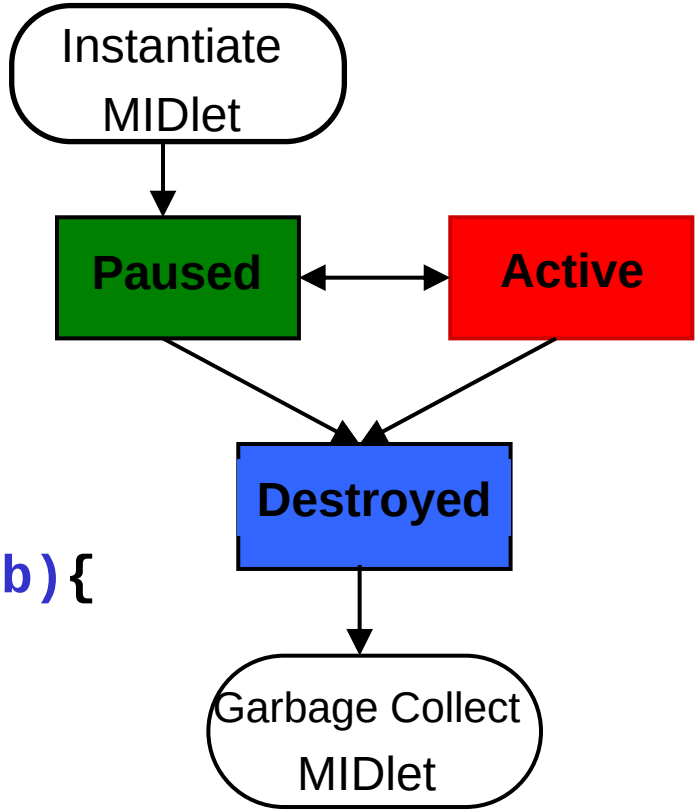
```
import javax.microedition.lcdui.MIDlet;

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**

J2ME – MIDP – Persistent Storage

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

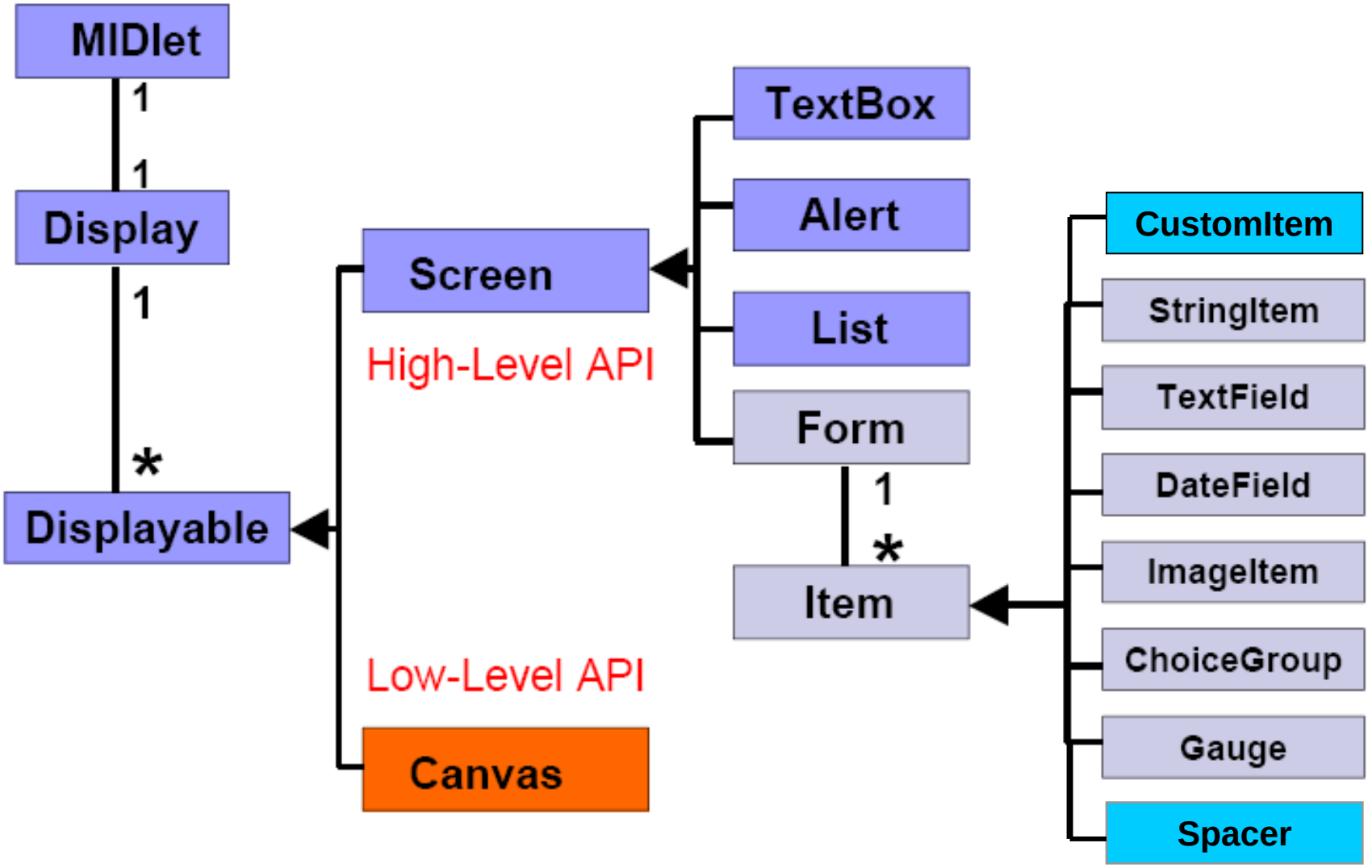
```
private RecordStore recordStore =
  RecordStore.openRecordStore("nirwana", true);
byte[] b = myChunkOfInfo.toByteArray();
recordStore.addRecord(b, 0, b.length);
```

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

J2ME – MIDP – Display Capabilities

- **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 – GUI API


MIDlet
1
1
Display
1
*
Displayable
Screen
High-Level API
Low-Level API
Canvas
TextBox
Alert
List
Form
1
*
Item
CustomItem
StringItem
TextField
DateField
ImageItem
ChoiceGroup
Gauge
Spacer

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 …

J2ME – MIDP - Code Example High Level API

```
private Form mMainForm;
private Command ok,exit;

public CPandM(){
  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);
}

public void startApp() {
  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

J2ME – MIDP - Code Example Low Level API

```
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

J2ME – MIDP – Application Management cont.

- **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-1.1
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

J2ME – MIDP – Security concepts cont.

- 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)

Extensions

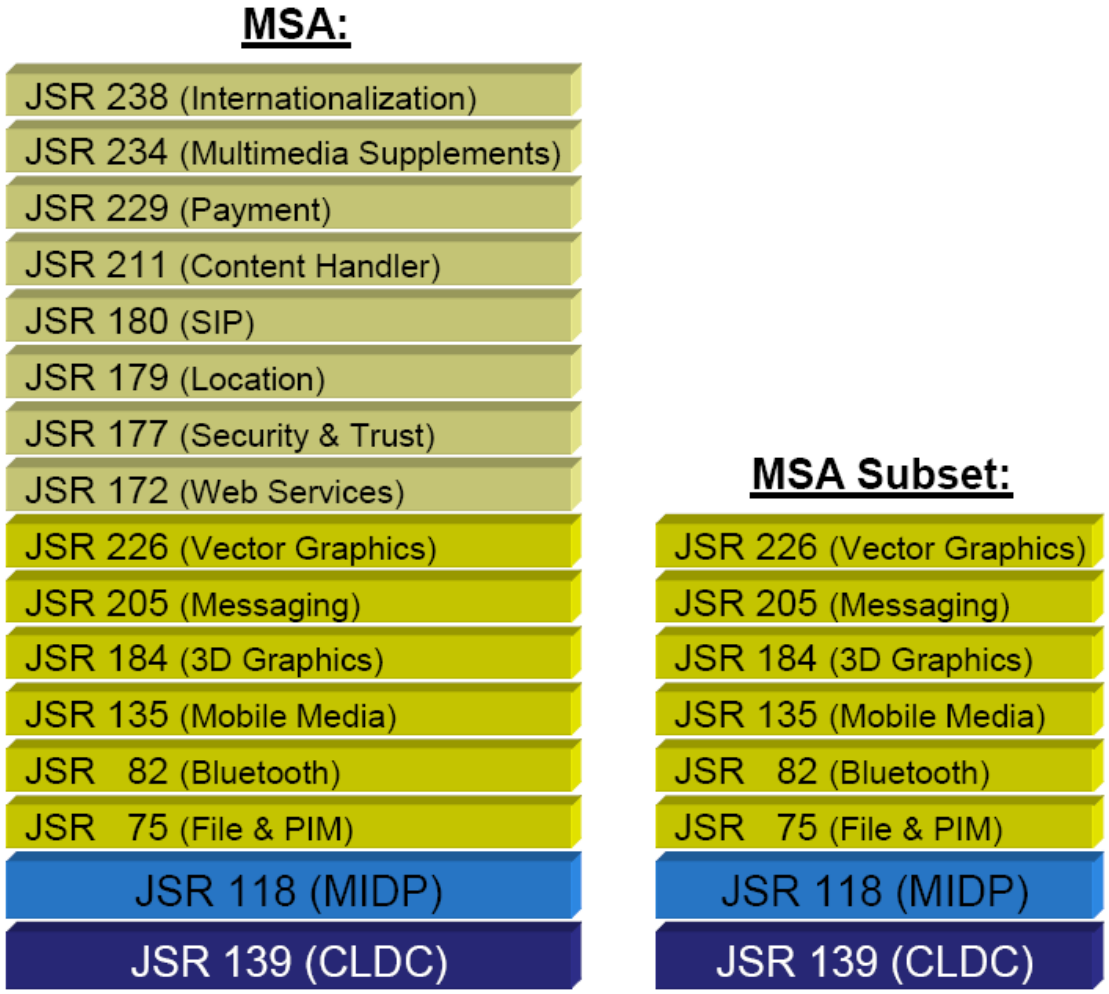
- **JSR-066**
  - RMI Optional Package Specification Version 1.0
- **JSR-082**
  - Bluetooth
  - Obex
- **JSR-120, 205**
  - Wireless Messaging API
- **JSR-135**
  - Mobile Media API
- **JSR-172**
  - J2ME Web Services (Sax,XML RPC, RMI)

Java Technology for the Wireless Industry (JTWI)

- **JSR-185**
  - To improve the compatibility, interoperability and completeness of J2ME technology implementations in mobile phones
- **Mandatory**
  - MIDP 2.0 (JSR-118)
  - WMA 1.1 (JSR-120)
- **Conditionally**
  - MMAPI 1.1 (JSR-135)
- **Minimum**
  - CLDC 1.0 (JSR-30)
- Must meet Technology Compatibility Kit (TCK) tests
- Recommendations for device and memory resources

Mobile Service Architecture API

- **JSR-248**
- addresses an even broader set of devices with more enhanced and diverse capabilities



Java ME – Summary

Summarization

- 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
- Java ME extensions not mandatory in CLDC or MIDP
- Java ME isn’t “Write Once Run Anywhere”