Programming Mobile Devices

Overview

University of Innsbruck

WS 2009/2010

STI · INNSBRUCK

thomas.strang@sti2.at

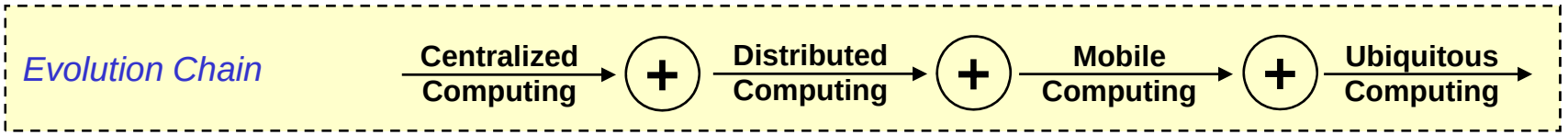
Administrative Issues

- We will start every Friday at 11:15 (i.e. doors are locked on 11:15…)
- Every unit up to 4 academic hours of lecture and practical exercises
- Attendence is not mandatory but strongly recommended
- Please subscribe asap at course mailing list: https://lists.sti2.at/mailman/listinfo/pmd-ws0910
- Lecture webpage at http://www.sti-innsbruck.at/teaching/courses/ws200910/details/?title=programming-mobile-devices-(pmd)

Student Evaluation

- Written exam at the end of the semster (required for VO)
- Bonus points are given for extra fast or extra clever solutions of exercises

History at a glance…



Remote Communications
protocol layering, RPC, end-to-end args ...

Fault Tolerance & Availablity
transactions, replication, load balancing ...

Management
policies, monitoring, managed objects ...

Remote Information Access
distrib. file systems, distrib. databases, caching ...

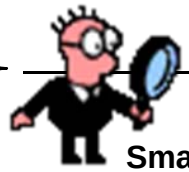
Security
encryption, authentication & authorisation, PKI ...

Mobile Networking
Mobile IP, wireless networks ...

Mobile Information Access
partial autonomy, weak connectivity & consistency ...

Adaptive Applications
proxies, transcoding, agility ...

Position Sensitivity
GPS, triangulation ...



Smart Sensors & Devices
invisibility, ressource limitation awareness ...

Ad-hoc Networks
collaboration, zero config, rerouting ...

Context-Awareness
context refinement, location ...

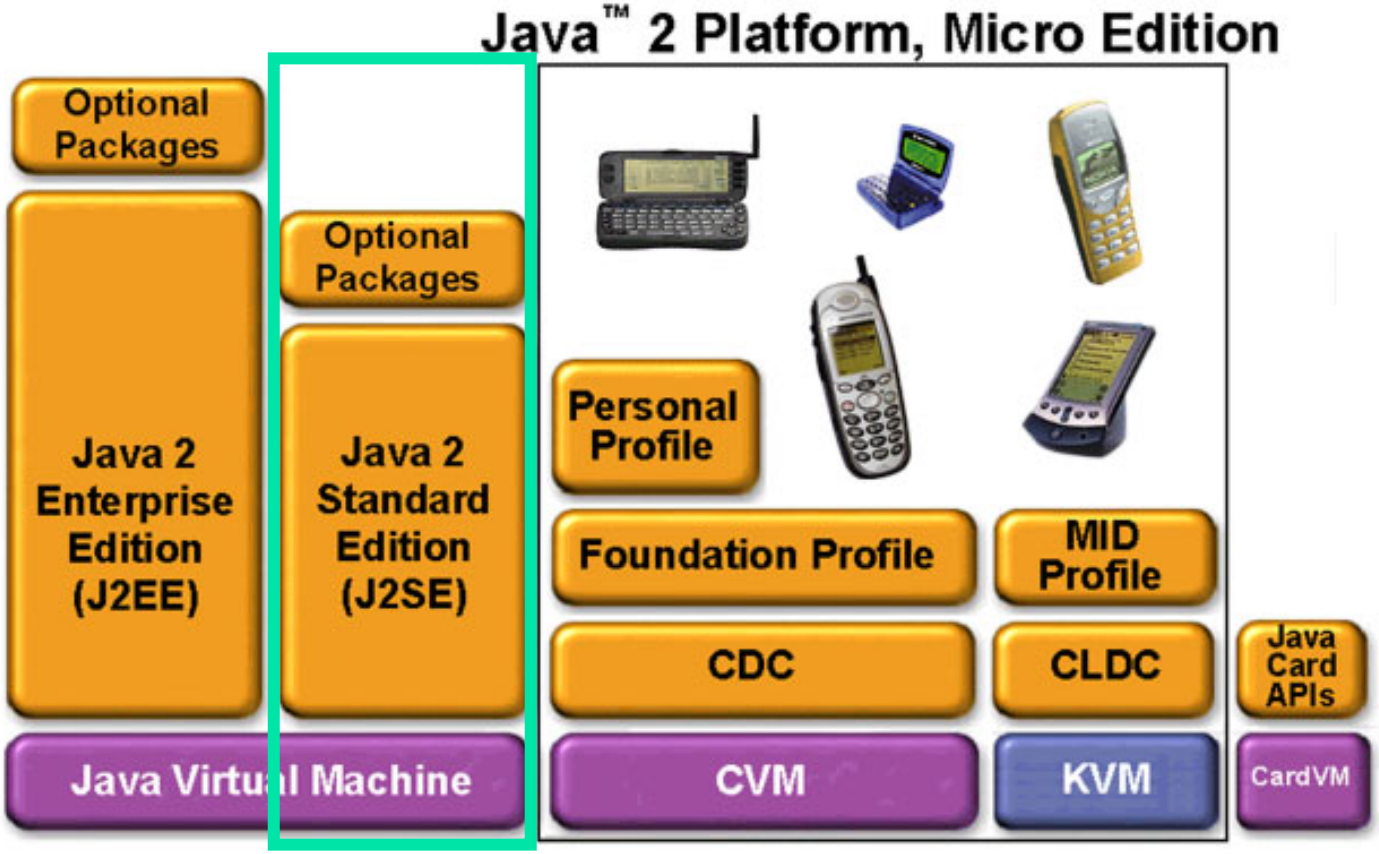
Smart Mobile Devices (SmartPhones etc.)…

- **multiple inhomogenious networks**
  - short range: IrDA, Bluetooth, Wireless LAN
  - wide area: (HS)CSD, GPRS, EDGE, UMTS
  - temporarily also no network
- **resource limited (RLD)**
  - battery power
  - processing power (performance)
  - memory (volatile and non-volatile)
- **programmable**
  - WML script
  - Java2 Micro Edition (J2ME) / i-Mode
  - Android (Google)
  - OS level (e.g. Symbian OS, iPhone)

… become a Computing Platform for Services

- if essential parts of service frameworks (e.g. life-cycle-management, discovery, execution etc.) are available on the device, supported by infrastructure elements
- (at least partial) autonomy

Java2 Standard Edition vs. Micro Edition



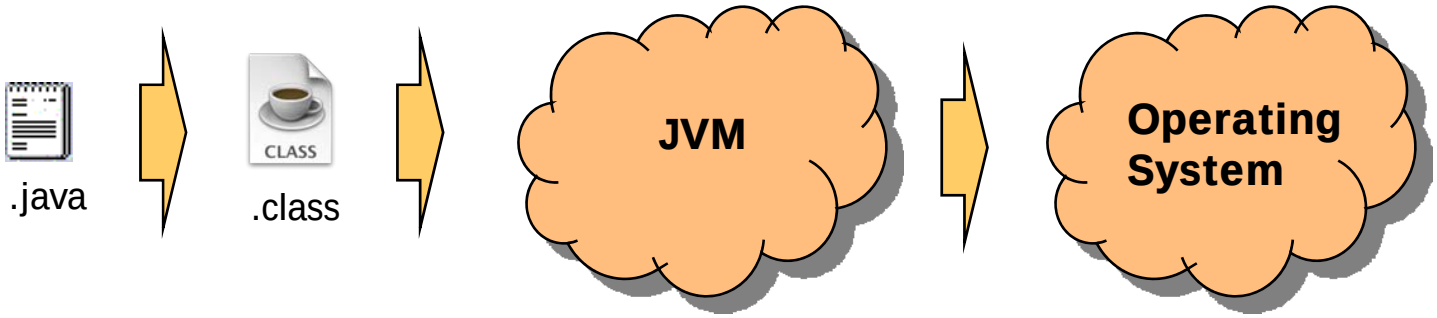
[Source: Sun Microsystems]

Runtime Environment + Application Installable

- **J2SE Platform**
  - Comes from Sun Microsystems
  - It is not a specific hardware or operating system, but an execution engine
- **Java technologies → all included in Java Development Kit (JDK):**
  - **javac,**
  - **jar**
  - **javadoc**
  - **jdb**

Java Virtual Machine (JVM)

- Byte code/binary format



- JVMs interpret the byte code semantically the same way, but the actual implementation are different
- More complicated than just the emulation of bytecode is compatible and efficient implementation of the Java core API which has to be mapped to each host operating system.

Java Compiler

- The Java source files (.java files) get compiled into byte code (.class files)
- The following are major Java compilers:
  - Javac
  - Jikes,
  - GCJ
- Interpretation of the byte code
- Just-in-time compilation of the byte code to machine code followed by dynamic compilation

Classpath

- Tells where to find third-party and user-defined classes and resources
- How to set:
  - -classpath
    - C:> *sdkTool* **-classpath** *classpath1*;*classpath2*...
    - C:> **set CLASSPATH=***classpath1*;*classpath2*...
  - CLASSPATH environment variable
- Classes are stored either in directories (folders) or in archive files
- Java classes are organized into packages
    - C:> **java -classpath C:\java\MyClasses utility.myapp.Cool**
    - C:> **java -classpath C:\java\MyClasses\myclasses.jar utility.myapp.Cool**

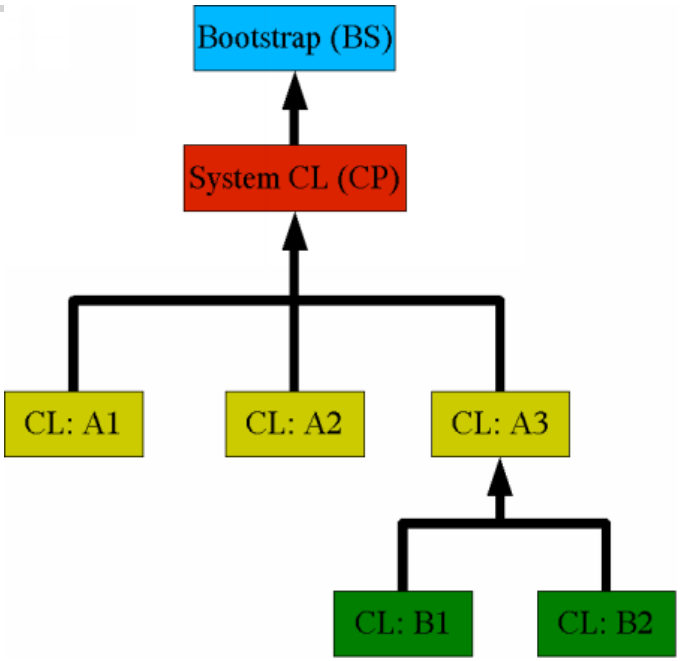
J2SE Entrypoint (main-routine)

- The method signature for the main method contains three modifiers:
  - public
  - static
  - void

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

Class Loaders

- Class loaders are hierarchically organized
- Class loaders should (practically: must) delegate the loading of a class to the parent
- A class is defined by its class type and the effective class loader
- A class is only loaded once and then cached in the class loader to ensure that the byte code cannot change



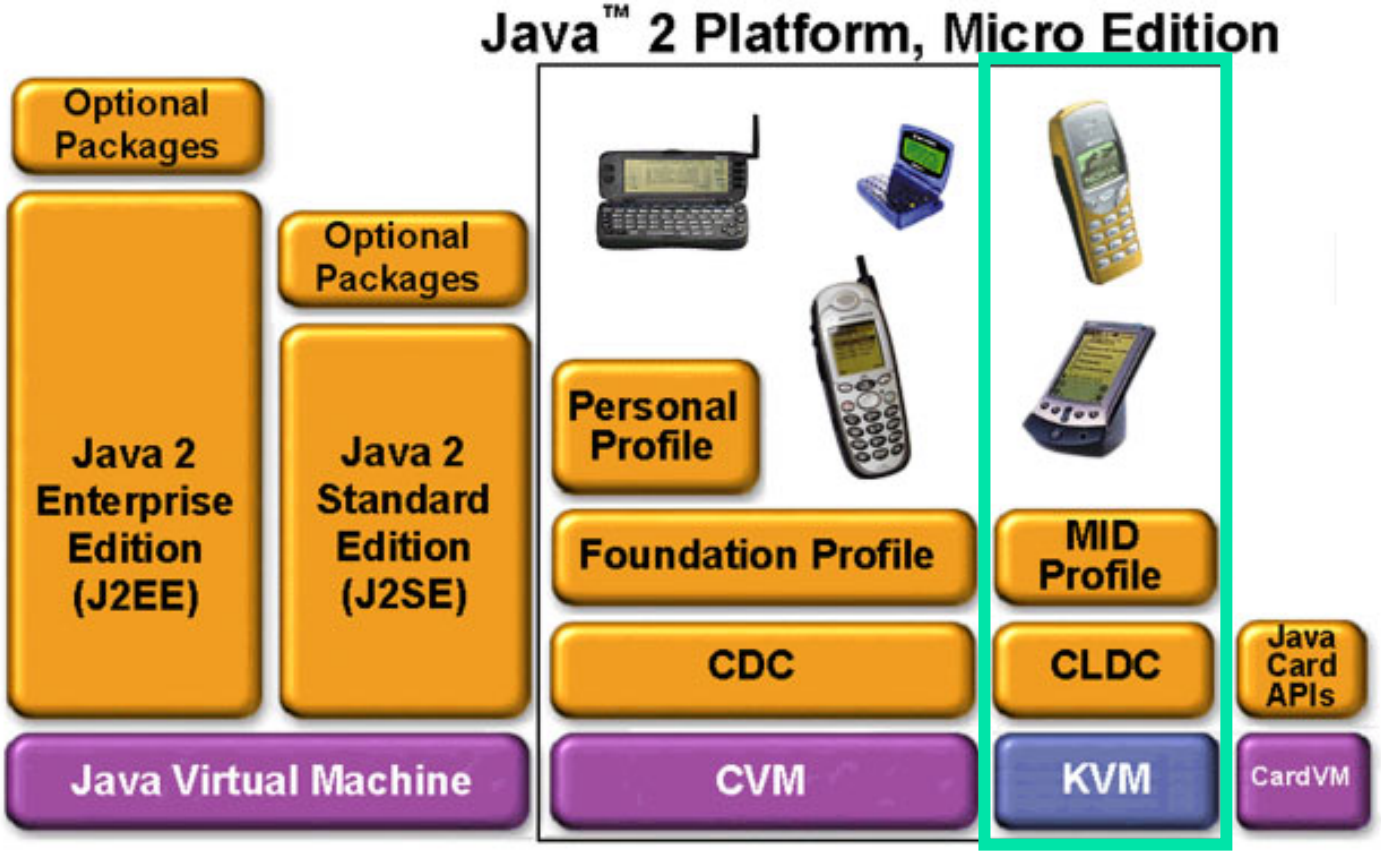
Basic libraries

- **Base Libraries java.lang, java.util Packages**
- Monitoring and Management
- Package Version Identification
- Reference Objects
- Reflection
- Collections Framework
- Concurrency Utilities
- Java Archive (JAR) Files
- Zip Files
- Logging
- Regular Expressions
- Preferences

Development Environments

- An **integrated development environment** (**IDE**), also known as **integrated design environment** and **integrated debugging environment**
- **Free Java-based IDEs**
  - Eclipse
  - Oracle JDeveloper
  - NetBeans IDE
  - JGrasp
  - jEdit
  - TruStudio Foundation
- **Proprietary (not free ☺) Java-based IDEs**
  - JBuilder
  - IntelliJ IDEA
  - Sun ONE Studio based on the Open source NetBeans
  - Rational Application Developer based on Eclipse
  - JCreator
  - TruStudio Professional

Java2 Standard Edition vs. Micro Edition



[Source: Sun Microsystems]

At the low end: MIDP on top of CLDC

- as defined in the original CLDC 1.0 (JSR-30) as well as MIDP 1.0 (JSR-37)

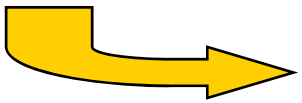
160 kB to 512 kB memory for **KVM**, CLDC- and MIDP-libraries and applications, whereas ▪ 256 kB persistent memory (eg. Flash) for system ▪ 8 kB persistent memory for applications ▪ ≥ 64 kB RAM for applications
low power consumption, often battery driven
network connected, often wireless & unreliable connections with limited bandwidth (often 9600 bps or less)
display with 1:1 pixel shape, and a minimum of ▪ 96 x 54 pixel ▪ 1 bit color depth
ITU-T-onehand-keyboard or touch-screen

Selective Remarkables

- KVM
  - classfile preverification required
- CLDC 1.0
  - no float, no double, no serialization, no custom classloaders
  - Generic Connection Framework (GCF), but no mandatory network protocol (no sockets / IP, no RMI)
- MIDP 1.0
  - no Swing or AWT, just own GUI:
    Liquid Crystal Display User Interface (LCDUI)
  - mandates HTTP as network protocol
  - Application Management

Verification

- Runtime byte code verification guarantees type safety.
- Classes that pass the verifier cannot, for example, violate Java virtual machine's type system and corrupt the memory.
- Static and dynamic memory footprint of the on-device verifier is too excessive for small resource limited devices (J2SE: ~50 kB binary code space, ≥ 30-100 kB RAM)



phase 1: off-device pre-verification
phase 2: on-device verification

Preverification

- Preverification performs certain checks on the Java bytecodes ahead of runtime:
  - Inline all subroutines and remove jumps
  - Add special `StackMap` attributes, describing the types of local variables and operand stack items
- A missing, incorrect or corrupted verification attribute causes the class to be rejected by the verifier.
- Pre-verified classfiles increased by about 5% and are still valid J2SE classfiles (because the `StackMap` attribute is a sub-attribute of the standard `Code` attribute)

Selective Remarkables

- KVM
  - classfile preverification required
- CLDC 1.0
  - no float, no double, no serialization, no custom classloaders
  - Generic Connection Framework (GCF), but no mandatory network protocol (no sockets / IP, no RMI)
- MIDP 1.0
  - no Swing or AWT, just own GUI:
    Liquid Crystal Display User Interface (LCDUI)
  - mandates HTTP as network protocol
  - Application Management

No Custom Classloaders

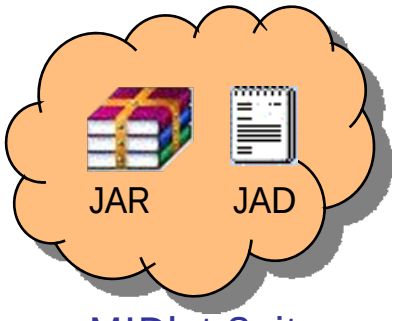
- To maintain upward compatibility for tools and environments, CLDC implementations are mandated be able to read pre-verified standard Java classfiles, including compressed Java Archive (JAR) support
- Unlike J2SE, CLDC / MIDP are not installed on the device by the customer
  - no classpath support (predefined loading order)
  - no overriding of system classes (classes belonging to CLDC or MIDP libraries), including classloader, as part of security concept

Selective Remarkables

- KVM
  - classfile preverification required
- CLDC 1.0
  - no float, no double, no serialization, no custom classloaders
  - Generic Connection Framework (GCF), but no mandatory network protocol (no sockets / IP, no RMI)
- MIDP 1.0
  - no Swing or AWT, just own GUI:
    Liquid Crystal Display User Interface (LCDUI)
  - mandates HTTP as network protocol
  - Application Management

Application Management

- System classes pre-installed on the device
- MIDP applications are called MIDlets
- Packaging unit is a MIDlet Suite
  - contains one or more MIDlets
    - one active at a time
  - persistent data access boundary
  - unit of installation / de-installation
    - distributed as one JAR archive (classfiles, resources, manifest) + associated description JAD
    - if installation of e.g. one class of a suite fails, the whole suite will be rejected from installation
    - determines binding options → "closed late binding"



MIDlet Suite

Late Binding

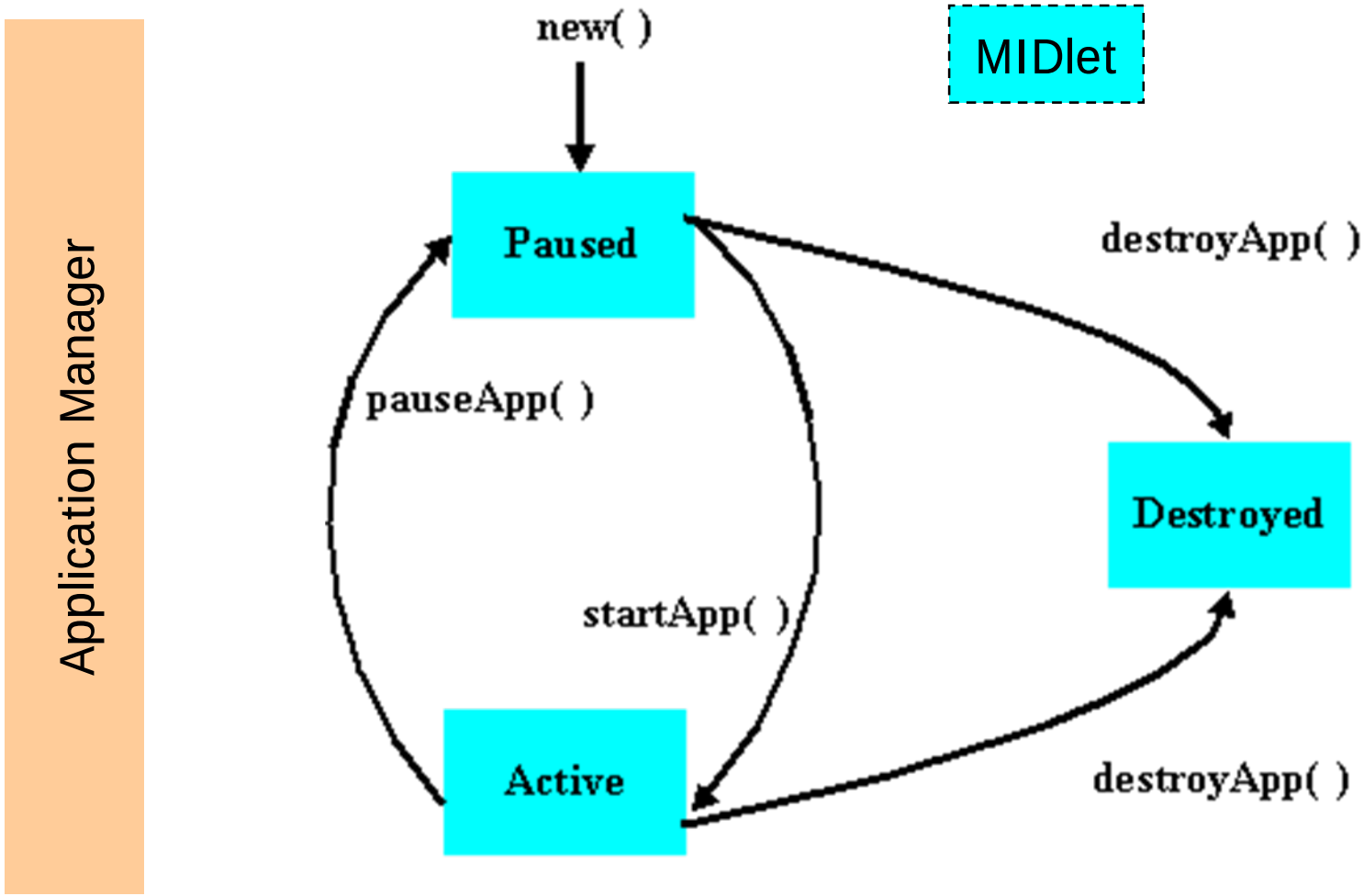
- *late binding* (also called *dynamic binding* or *run-time binding*) means that the binding of a method call to a method body occurs at run-time (usually based on the type of object)
- As in any language supporting late binding, the Java environment provides a mechanism to locate and load the method body at run-time, in this case using a Classloader
- In standard Java several kinds of classloaders exist, e.g. to load classfiles from network resources

Closed Late Binding

- In a CLDC / MIDP runtime environment, the system classloader may load classes only from the system libraries or the closed set of classes contained in the JAR used to install the current suite.
  - no custom classloaders allowed
  - no overriding of system classloader allowed
  - no classloading from other resources than JAR
    - withing one suite not even from other suites
- Late binding is "closed" to what is there at install-time

"Closed Late Binding"

MIDlet lifecycle



Hello World MIDlet

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

public class HelloWorld extends MIDlet implements CommandListener {
  private Command exitCommand;
  private TextBox tbox;

  public HelloWorld() {
    exitCommand = new Command("Exit", Command.EXIT, 1);
    tbox = new TextBox("MIDletTextBox", "Hello MIDP World!", 25, 0);
    tbox.addCommand(exitCommand);
    tbox.setCommandListener(this);
  }

  protected void startApp() {
    Display.getDisplay(this).setCurrent(tbox); }
  protected void pauseApp() {}
  protected void destroyApp(boolean bool) {}
  public void commandAction(Command cmd, Displayable disp) {
    if (cmd == exitCommand) { destroyApp(false); notifyDestroyed();
} } }
```

Environment Variables

- `String MIDlet.getAppProperty(String key)`
  - to read MIDlet specific environment variables such as
    - `userid` if a MIDlet is personalized to a user

- `static String System.getProperty(String key)`
  - to read system properties such as
    - `microedition.profiles` The set of profiles supported by this device, for example, "`MIDP-1.0.`"
    - `microedition.configuration` The J2ME configuration supported by this device, for example, "`CLDC-1.0.`"

Provisioning

- Neither MIDP 1.0 nor CLDC 1.0 specify how a suite may be installed on the device
- A "Recommended Practice Addendum" describe how MIDlet suites can be deployed *Over The Air (OTA)*
- In practice, the following options of provisioning are available:
  - Cradle (cheap, fast, disliked by mobile operators)
  - OTA
    - Bluetooth + OBEX (cheap, fast, disliked by mobile operators, weakly supported by device manufacturers)
    - GSM/GPRS/UMTS + WAP/HTTP (expensive, slow, pushed by mobile operators)

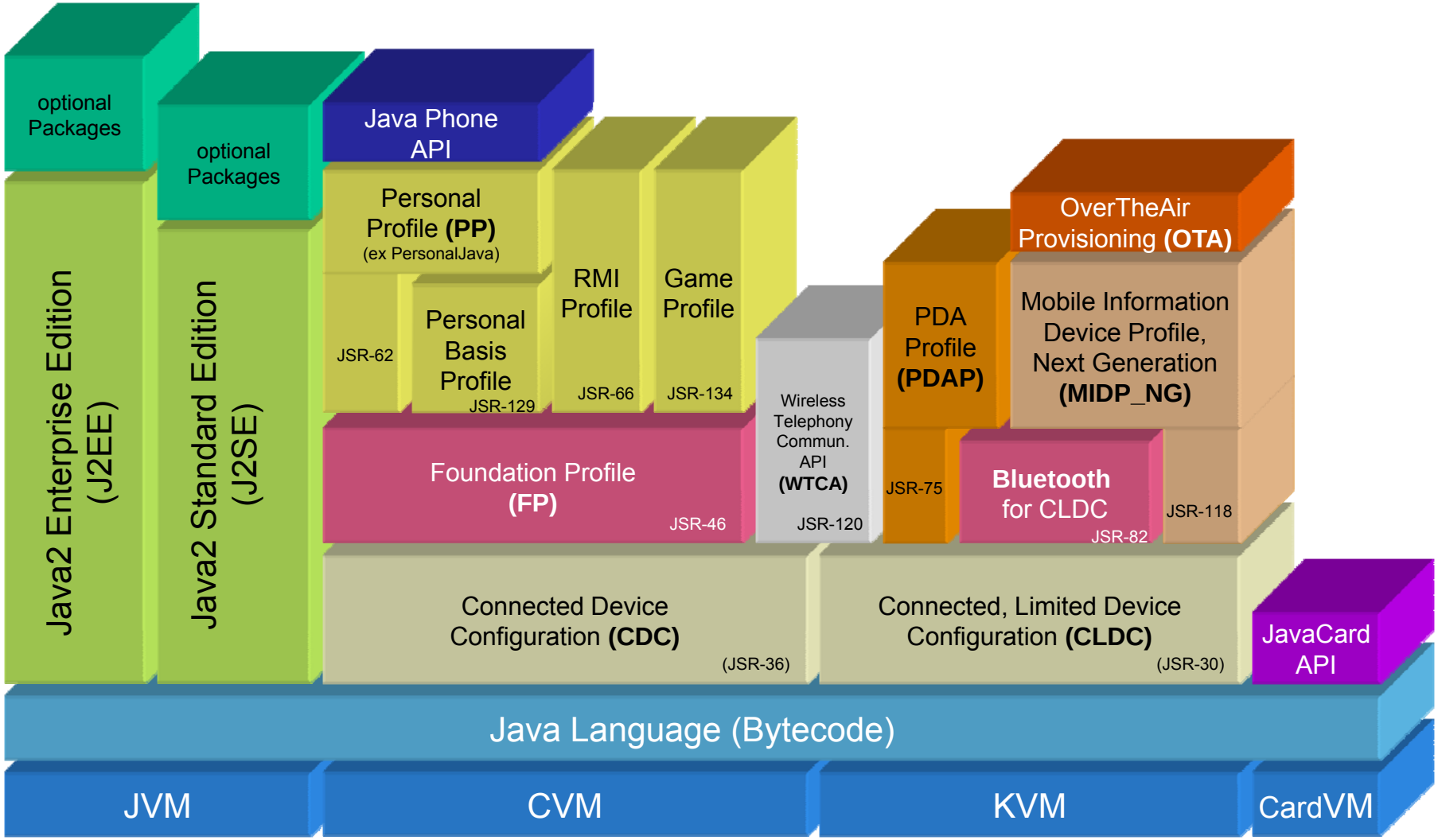
Changes by MIDP 2.0 on top of CLDC 1.1

- CLDC 1.1 (CLDC_NG, JSR-139)
  - released December 2002
  - Backward compatible with CLDC 1.0
  - `Float` and `Double` added
  - Small refinements of selected classes (e.g. `Date`)
  - Minimum memory budget has been raised from 160 kB to 192 kB (mainly because of floating point support)
  - Much more detailed pre-verifier specification

Changes by MIDP 2.0 on top of CLDC 1.1

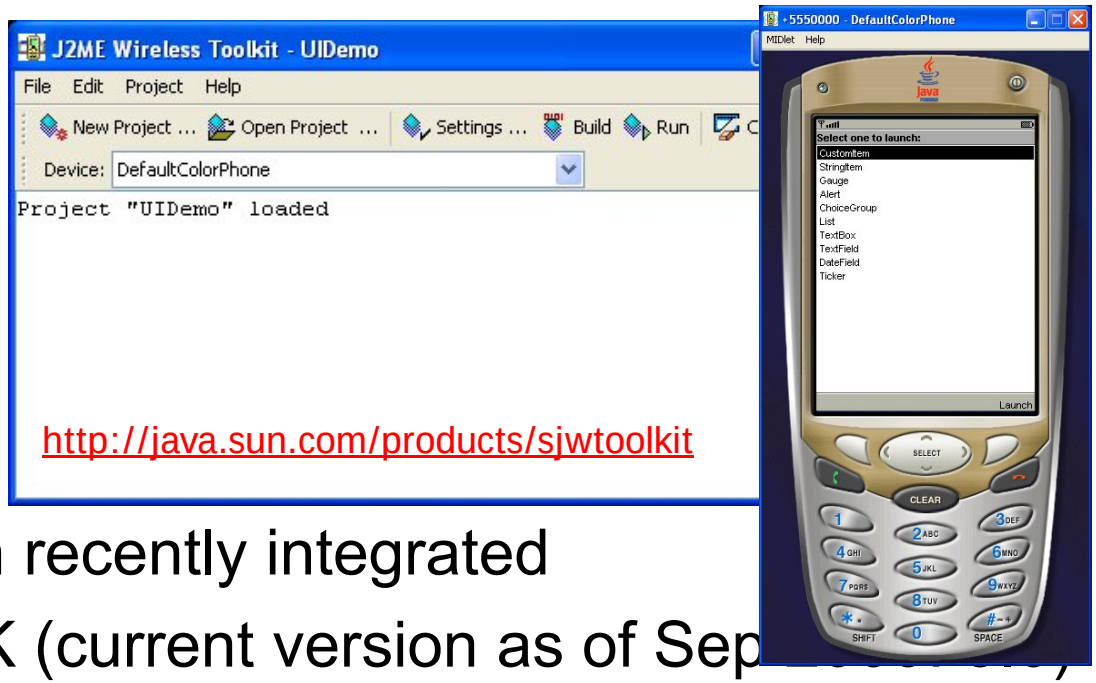
- MIDP 2.0 (MIDP_NG, JSR-118)
  - released November 2002
  - Backward compatible with MIDP 1.0
  - Integrates *Over The Air User Initiated Provisioning Specification* as part of MIDP 2.0 spec
  - Added PKI support for security/trust of MIDP applications, similar to MExE trust model
  - Added HTTPS (Secure HTTP, Transport Level Security)
  - Push-Connections (e.g. triggered by SMS)
  - Inter-suite API & RecordStore access
  - Extended LCDUI (e.g. media player support)

Java2 Family Tree (snapshot of 2001, still expanding!)



Development Environment

- SUN provides reference emulator: Wireless Toolkit



- WTK has been recently integrated into J2ME SDK (current version as of Sep
- In RR15: Either enter „ktoolbar“ in console or fedora -> siteapps -> WTK starten
  - In case of problems, ask ZID (not me ☺)

Tips to install WTK on your own Linux computer

- in order to get the j2me wireless toolkit working on linux, the following instructions may be useful:

  note: the downloaded and unpacked directory is ${WTKDIR}, the workspace for the toolkit is ~/j2mewtk/2.5.2/

  1. copy the file ${WTKDIR}/wtklib/Linux/ktools.properties to ~/j2mewtk/2.5.2/wtklib/
  2. open the file ~/j2mewrk/2.5.2/wtklib/ktools.properties in your favourite editor
  3. find the line beginning with „kjava.class.path: “ and add ":lib/j2me-xmlrpc.jar“ (without quotes) to the classpath. The final classpath is: kjava.class.path: lib/midpapi20.jar:lib/cldcapi10.jar:lib/j2me-xmlrpc.jar
  4. restart the j2me wireless toolkit

  [provided by Simon Bailey, IFI systems administrator]