Java Family Overview

Different applications have different expectations, one development platform cannot completely address all of them:

- Web servers
- Workstations
- TV boxes
- Mobile phones
- Washing machines
-...

J2ME is a robust, flexible environment for applications running on consumer devices, such as mobile phones, PDAs, TV set-top boxes, printers and a broad range of other embedded devices.
3 different Java development platforms:

- **J2EE** - Java 2 Enterprise Edition
  (Web server based applications)

- **J2SE** - Java 2 Standard Edition
  (standard desktop applications)

- **J2ME** - Java 2 Micro Edition
  (small & memory constrained devices)

J2ME is promoted by Oracle (Sun Microsystems)
J2ME platform architecture

By selecting a configuration, a profile, and any number of optional packages, a product designer defines the Java runtime environment for a given device based on the Java ME technology.
### JSR (Java Specification Request)

**Categories for Java ME Platform**

<table>
<thead>
<tr>
<th>JSR Category</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Configuration</strong></td>
<td>Provides the most basic set of class libraries and virtual machine capabilities for a broad range of devices.</td>
<td>CDC supports network-connected devices and provides a subset of the APIs in the Java SE, formerly J2SE, platform.</td>
</tr>
<tr>
<td>(required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Profile</strong></td>
<td>Defines additional APIs that support a narrower range of devices. A profile is butli on a specific configuration.</td>
<td>Foundation Profile can be used for network printers without GUls, while Personal Profile can be used for high-end PDAs. Both of these profiles are based on CDC.</td>
</tr>
<tr>
<td>(required)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Optional package(s)</strong></td>
<td>Defines a set of technology-specific APIs.</td>
<td>The Java Database Connectivity (JDBC™) Optional Package provides database access, while the RMI Optional Package enables distributed application development. Both of these optional packages are compatible with each of the CDC profiles.</td>
</tr>
</tbody>
</table>
J2ME Configurations

A configuration provides the base functionality for a particular collection of devices that share similar hardware characteristics:

- CPU speed
- Memory size
- Screen size
- Bandwidth requirements, etc.

Currently, there are two J2ME configurations:

- CDC - Connected Device Configuration (smartphones, TV set-top boxes, telematics systems, RFID readers),

- CLDC - Connected Limited Device Configuration (phones, low-end PDAs/palmtops).
Example: CDC Target Device Categories

- Fixed purpose:
  - Industrial automation
  - Office equipment

- Mobile:
  - Field service automation
  - Personal mobile devices
  - Telematics
  - VOIP

- Stationary:
  - Set-top boxes
J2ME Limitations

**CDC devices:**
- No Applets
- No Beans
- No RMI
- No AWT/Swing
- Special IO protocol

**CLDC devices:**
- as CDC + additionally:
  - No floating point math
  - No zip/jar utilities
  - No reflection
Technical characteristics: **CDC vs CLDC**

- **Connected Device Configuration:**
  - 2Mbytes or more of memory,
  - 32-bit processor with FPU,
  - High bandwidth network connection (≥ 56 kbps)
  - Fully featured VM

- **Connected Limited Device Configuration:**
  - 160 Kbytes to 512 Kbytes of total memory available,
  - 16-bit or 32-bit processor (without FPU),
  - Low power consumption (often operating with battery power),
  - Connectivity: at times, with limited bandwidth (eg.9600 bps).
J2ME profiles

- A profile supports a narrower category of devices within the framework of a chosen configuration.

- Profiles refine how the programmer interacts with the user interface and provides access to device-specific properties.

- Example profiles:
  - TV profile
  - Screen phone profile
  - Car profile
  - Mobile Information Device profile MIDP
MIDP
(Mobile Information Device Profile)

- profile which is most widely deployed on CLDC (phones)
- defines: application behavior, user interface, networking, persistent storage
- allows better control of the applications that are installed on MIDP mobile phones (security, commercial reasons - operators revenue)

Outside MIDP profile scope are system level APIs and OEM specific technologies (e.g. programming camera, Bluetooth or GPS on mobile phones)
MIDP (2)

- The Mobile Information Device Profile (MIDP) provides APIs for application life cycle, user interface, networking, and persistent storage.

- Developers using MIDP can write applications once, then deploy them quickly to a wide variety of mobile information devices.

- MIDP has been widely adopted as the platform of choice for mobile applications.

- It is deployed globally on millions of phones and PDAs, and is supported by all leading integrated development environments.
MIDP Device Minimal Requirements

➢ **Output**
  – 96 x 54 pixel display

➢ **Input**
  – One-handed keyboard (phone)
  – Two-handed keyboard (qwerty, touch)

➢ **Memory**
  – 128kB ROM
  – 8kB ROM/Flash
  – 32kB RAM (for Java heap)

➢ **Networking**
  – Two-way, wired/wireless
  – Limited bandwidth (9.6kbps)
JSR - J2ME Optional Packages

- Java ME includes Java virtual machines and a set of standard Java APIs — the JSRs (Java Specification Request).

- JSR packages are created to address very specific application requirements, such as database connectivity, multimedia, bluetooth, etc.

- They are defined through the Java Community Process (JCP), by expert groups whose members include leading device manufacturers, software vendors, and service providers.

- J2ME is an extensible platform that continuously expands to address new and upcoming technologies. Each JSR defines a Java API related to the issues it addresses.
<table>
<thead>
<tr>
<th>JSR No</th>
<th>DESCRIPTION OF BASIC „PHONE” PACKAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSR-30</td>
<td><strong>Connected Limited Device Configuration 1.0 [ CLDC1.0 ]</strong>&lt;br&gt;defines the base set of application programming interfaces and a virtual machine for developing applications to run on devices with limited memory, processing power, and graphical capabilities.</td>
</tr>
<tr>
<td>JSR-139</td>
<td><strong>Connected Limited Device Configuration 1.1 [ CLDC1.1 ]</strong>&lt;br&gt;Add the following to CLDC 1.0: floating point maths, weak references support.</td>
</tr>
<tr>
<td>JSR-37</td>
<td><strong>Mobile Information Device Profile 1.0 [ MIDP1.0 ]</strong>&lt;br&gt;is the Java runtime environment for today's most popular compact mobile information devices, such as cell phones and mainstream PDAs.</td>
</tr>
<tr>
<td>JSR-118</td>
<td><strong>Mobile Information Device Profile 2.0 [ MIDP2.0 ]</strong>&lt;br&gt;Adds the following to MIDP1.0:&lt;br&gt;— UDP/Sockets/Secure Sockets/Server Sockets and access to serial ports&lt;br&gt;— Additional image formats (GIF / JPG / WMB)&lt;br&gt;— PCM (Pulse Code Modulation) wave format and/or additional sampled/synthetic sound formats&lt;br&gt;— Copy protection of MIDlet suites&lt;br&gt;— Discovery/Installation of MIDlets via other mechanisms (InfraRed/Bluetooth/etc)</td>
</tr>
<tr>
<td>JSR-248</td>
<td><strong>Mobile Service Architecture [ MSA API ]</strong>&lt;br&gt;Reduces variations in the platform environment and creates a predictable environment for application developers. Defines a clear set of component JSRs that must be supported (mandatory JSRs !) on next-generation mobile phones:&lt;br&gt;— 3D graphics,&lt;br&gt;— personal information management,&lt;br&gt;— Bluetooth,&lt;br&gt;— Scalable Vector Graphics (SVG),&lt;br&gt;— web services,&lt;br&gt;— location services&lt;br&gt;— payment services</td>
</tr>
<tr>
<td>JSR</td>
<td>DESCRIPTION OF OPTIONAL PACKAGES</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>JSR-75</td>
<td><strong>PIM &amp; File Data API [PDAP]</strong> → Personal information management: phone contacts, calendar events, alarms. Enables access to device file system, Secure Digital card, Memory Stick card</td>
</tr>
<tr>
<td>JSR-82</td>
<td><strong>Bluetooth API [BTAPI]</strong> → Search for Bluetooth devices. Query device capabilities. Set BT connection to device</td>
</tr>
<tr>
<td>JSR-120</td>
<td><strong>Wireless Messaging API 1.0 [WMA1.0]</strong> → support for Short Message Service (SMS), (CBS)</td>
</tr>
<tr>
<td>JSR-135</td>
<td><strong>Mobile Media API [MMAPI]</strong> → support for time-based media such as audio and video</td>
</tr>
<tr>
<td>JSR-172</td>
<td><strong>J2ME Web Services Specification [WSAPI]</strong> → provides access from J2ME to web services.</td>
</tr>
</tbody>
</table>
| JSR-177 | **Security and Trust API [SATSA]**  
|        | — Provides support for the management of user certificates, user authentication using certificates  
|        | — Allows for J2ME applications to communicate with SmartCard-based applications  
|        | — General purpose encryption functionalities                                                                                                                                                                                        |
| JSR-179 | **Location API [LAPI]** → enables developers to write mobile location-based applications.                                                                                                                                              |
| JSR-184 | **Mobile 3D Graphics [M3G]**  
|        | — Provides an efficient 3D Graphics API suitable for the J2ME platform  
|        | — Supports scene graph API (high-level) and immediate API (low-level, subset of OpenGL)  
|        | — Provides importer functions for textures, meshes, animations, and scene hierarchies                                                                                       |
| JSR-205 | **Wireless Messaging API 2.0 [WMA2.0]** → support for Multimedia Message Service (MMS)                                                                                                                                                 |
| JSR-211 | **Content Handler API [CHAPI]** → enabling to handle multi-media and web content.                                                                                                                                                     |
| JSR-226 | **Scalable 2D Vector Graphics API [SVGAPI]** → defines an optional package API for rendering scalable 2D vector graphics, including image files in W3C Scalable Vector Graphics (SVG) format.                                                        |
| JSR-256 | **Mobile Sensor API [MSAPI]** → extends the usability and choice of sensors for J2ME applications                                                                                                                                    |
| JSR-257 | **Contactless Communications API [CCAPI]** → one-way and bi-directional data capture and data integration from RFID, barcode and other existing and standard approaches.                                                                 |
# Evolution of SonnyEricsson Java Platforms

<table>
<thead>
<tr>
<th>JP</th>
<th>FEATURES</th>
<th>OPTION</th>
<th>HANDSET MODELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP-1</td>
<td>CLDC-1.0, MIDP1.0, JSR-135</td>
<td>JSR-120</td>
<td>T610, T616, T618, Z600, Z608, T628,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>T630, T637</td>
</tr>
<tr>
<td>JP-2</td>
<td>CLDC-1.1, MIDP2.0, JSR-120, JSR-135, JSR-185, NokiaUI 1.1</td>
<td></td>
<td>Z1010</td>
</tr>
<tr>
<td>JP-3</td>
<td>CLDC-1.1, MIDP2.0, JSR-120, JSR-135, JSR-184, JSR-185, Mascot Capsule v3, NokiaUI 1.1</td>
<td>JSR-120</td>
<td>K700, K500, K506, K508, F500, S700, S710a, K300, Z500</td>
</tr>
<tr>
<td>JP-4</td>
<td>CLDC-1.1, MIDP2.0, JSR-120, JSR-135, JSR-184, JSR-185, Mascot Capsule v3, NokiaUI 1.1</td>
<td>VSCL-2.0</td>
<td>V800, Z800</td>
</tr>
<tr>
<td>JP-5</td>
<td>CLDC-1.1, MIDP2.0, JSR-75, JSR-120, JSR-135, JSR-184, JSR-185, Mascot Capsule v3, NokiaUI 1.1</td>
<td>JSR-82</td>
<td>K750, W800, K600, D750, V600, K608, Z520</td>
</tr>
<tr>
<td>JP-6</td>
<td>CLDC-1.1, MIDP2.0, JSR-75, JSR-120, JSR-135, JSR-172, JSR-184, JSR-185, JSR-205, Mascot Capsule v3, NokiaUI 1.1</td>
<td>JSR-82</td>
<td>W500, W600, W900</td>
</tr>
</tbody>
</table>
MIDlet

- MIDP application is typically called a MIDlet.
  - Similar to java applet
  - Extends MIDP MIDlet class

- MIDlets are packaged/downloaded in a JAR file including:
  - class files of the MIDlets
  - resource files (e.g., images, icons, and audio)

- MIDlet content is described in an Application Descriptor (JAD) file which accompanies each JAR.
  - Describe crucial properties for the MIDlet
    (developer/vendor, size, version, storage requirements, etc.)
Sample JAD file

MIDlet-1: BuzzGame, Buzz.png, BuzzGame
MIDlet-Jar-Size: 1032
MIDlet-Jar-URL: http://www.me.com/Buzz.jar
MIDlet-Name: BuzzGame
MIDlet-Vendor: MobileEntertainment
MIDlet-Version: 2.0
MIDlet Suites

- One or more MIDlets are packaged in a single jar file and distributed as MIDlet suites.
- They have greater ability to share external resources than separate MIDlets.
- All of the MIDlets in a suite are installed onto a device as a single entity.
- At runtime, if the device supports concurrent running of more than one MIDlet, all active MIDlets from a MIDlet suite run in the same Java VM.
MIDlet Deployment Methods

1. Development or „private” deployment
   - download via Bluetooth, cable, or infrared
   - download from a web site via GPRS/3G connection
     (download the jad file first, which makes phone download the jar file)

2. Commercial deployment
   - some MIDlets are packaged with phone,
   - download from the MNO (mobile network operator)
   - download from a software vendor/aggregator’s web site,
   - request by SMS.
Standard Process of MIDlet Distribution (OTA Provisioning)

1. A user chooses a link (JAD file) on a website
2. Server returns JAD file
3. User checks JAD properties describing selected MIDP application
4. If description is satisfactory, MIDP virtual machine is executed and JAR is requested.
5. Website returns JAR file
6. JAR file downloaded, MIDlet in JAR file is executed
MIDP - Development Toolkits

- Sun Java **Wireless Toolkit (WTK)** - provides complete development support for developing MIDP applications, and works in combination with Java technology IDE

- **Netbeans Mobility** (5.0 with mobility pack, or 6.x and above which includes the mobility pack)

- **Eclipse IDE** with Java Mobile plugin (Eclipse ME)

- **Sun ONE Studio, Mobile Edition** - Java technology IDE for developing applications that can be deployed to Java technology-enabled mobile devices.
In the lab - NetBeans IDE
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class HelloMIDlet
    extends MIDlet
    implements CommandListener {
    private Form mMainForm;

    public HelloMIDlet() {
        mMainForm = new Form("HelloMIDlet");
        mMainForm.append(new StringItem(null, "Hello, MIDP!"));
        mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
        mMainForm.setCommandListener(this);
    }

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

    public void pauseApp() {}

    public void destroyApp(boolean unconditional) {}

    public void commandAction(Command c, Displayable s) {
        notifyDestroyed();
    }
}
The Application Management Software (AMS) is a part of the device's operating environment and manages MIDlets states (lifecycle) during the execution process.

When a user selects a MIDlet for execution, the AMS instantiates it by calling the MIDlet's `startApp()` method, which starts the life of the MIDlet.

Only one application can be active (shown on the display) at a time.

The AMS can send a MIDlet to the background (paused state) or bring it to the foreground (active state), as a response to user initiated actions or other events (like a phone call).
AMS (2)

- In active state, a MIDlet can call `Display.setCurrent(null)` to send the MIDlet to the background (next application getting the foreground is determined by the platform).
- To activate a background MIDlet and set it to the foreground, call `Display.setCurrent(Displayable)`.
- The system can close a MIDlet at any time, (for example due to shortage of resources e.g memory)
  - the AMS calls the MIDlet's `destroyApp()` method, forcing the application to a graceful exit (and wait 5 seconds)
  - if this fails, the AMS terminates the MIDlet forcefully.
MIDlet Life Cycle

MIDlets move through a number of different phases during execution:

- **Start** - allocate resources (such as window display etc) and start executing.
- **Pause** - release resources and wait for some external event.
- **Destroy** - deallocate all resources, destroy threads and end all activities.
MIDlet has to implement three abstract methods:

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

public class HelloMidlet extends MIDlet {
    public void startApp() { // called when application is started
    }

    public void pauseApp() { // called when application is paused
    }

    public void destroyApp(boolean unconditional) { // when application is exited
    }
}
```

These three methods are called by the J2ME runtime system (AMS) on your phone.
Creating MIDlet User Interface

- The central abstraction of the MIDP UI is a **screen**, which is an object that encapsulates device-specific graphics rendering user input.

- **Only one screen** can be visible at a time, and the user can traverse only through the items on that screen.

- The **Display** class is the display manager that provides methods to retrieve information about the device's display capabilities.

- A screen is made visible by calling the **Display.setCurrent(...)** method
MIDlet graphic APIs
(organised as HIGH level & LOW level API)

➢ High level APIs:
  – High level APIs are common to all MIDP devices
  – The high-level API is implemented by classes that inherit from the **Screen** class (**Alert**, **TextBox**, **List**, **Form**).
  – Typical applications utilising high level API include news updates, ticket purchases, etc.
  – No direct access to device features (address book, calendar, etc.)

➢ Low level APIs:
  – **Canvas** - provide access to display device and keypad input.
  – Typical applications include games.
High Level API

Similar to J2SE GUI, but reduced. Can not be positioned or nested freely.

Contains:

- **TextBox** → allows the user to enter multi-line text
- **List** → Enables the selection of elements in group
- **Alert** → shows a dialog for a limited period of time
- **Form** (can contain multiple form items for required interface/screen organization):
  - **ImageItem** → used to control the layout of an Image
  - **StringItem** → used for read-only text elements
  - **TextField** → holds a single-line input field
  - **DateField** → used for editing date and time information
  - **Gauge** → displays a bar graph for integer values
  - **ChoiceGroup** → enables the selection of elements in a group
Displaying Objects

To show a Displayable object (e.g. Alert, Form) you must use the `setCurrent()` method on the Display object:

```java
... Form myForm = new Form("My First MIDlet Form"); Display display = Display.getDisplay(this); display.setCurrent(myForm);
... or short

Display.getDisplay(this).setCurrent(myForm);
```
The Displayable Hierarchy
User Interaction - Commands

- Displayable objects can declare commands and declare a command listener:
  – addCommand(Command cmd)
  – addCommandListener()
- The device determines how the commands are shown on the screen or invoked by the user.
User Input - Buttons

- In J2ME, commands are commonly represented with softbuttons on the device. The following diagram shows two Command objects, one with the label "Exit" and one with label "View."

- If there are too many commands to be shown on the display, a device will create a menu to hold multiple commands. The following diagram shows how this might look.

Soft Buttons

Navigation (Arrow) Buttons

Select (OK) Button
Commands Types

• To create a Command, you need to supply a label, a type, and a priority.
  Command c = new Command("OK", Command.OK, 0);

• The type is used to signify a commonly used command. It helps the device to arrange the commands.

• Command Meaning:
  BACK → returns to the previous screen.
  CANCEL → standard negative answer to a dialog
  EXIT → for exiting from the application.
  HELP → a request for on-line help.
  ITEM → specific to the items of the Screen or the elements of a Choice.
  OK → standard positive answer to a dialog
  SCREEN → an application-defined command
  STOP → a command that will stop some currently running process, operation, etc.
Responding to Command Events

- When a Command is invoked by the user, a method is called to service the command.

- The exact method is:
  
  ```java
  public void commandAction(Command c, Displayable d)
  ```

  where ‘c’ is the Command invoked and ‘d’ is the Displayable the Command was added to.
RMS (Record Management System)

MIDP Persistent Storage

- Record store defined in `javax.microedition.rms`
- Record store identified by name:
  
  ```java
  recordStore = RecordStore.openRecordStore( "scores", true );
  ```
- Write byte array to datastore
  ```java
  recordId = addRecord( byte[] data, int offset, int numBytes );
  ```
- Read the record
  ```java
  getRecord( int recordId );
  ```
- A RecordStore is not a typical database → lightweight record oriented database
- MIDP „Database”
  - its consists of a set of records (arrays of bytes)
  - unique record ID for each database record.
  - with MIDP 2.0 a RecordStore can be shared across MIDlet suites.
  - support for enumeration, sorting and filtering
  - provides device independent API
Accessing network and data services

Example code using HTTP Connection:

```java
HttpConnection hc =
    HttpConnection)Connector.open(URL);
DataInputStream in =
    new DataInputStream(hc.openInputStream());
int ch;
String alltext = new String("");
while ((ch = in.read()) != -1)
    alltext += ((char) ch);
hc.close();
in.close();
```
MIDP Features- Multimedia & Game

- MIDP is ideal for building **portable games** and multimedia applications.
- A low-level UI API complements the high-level UI API, giving developers greater **control of graphics and input**.
- A **game API** adds game-specific functionality, such as sprites and tiled layers, which take advantage of devices' native graphics capabilities.
- **Built-in audio** provides support for tones, tone sequences, and WAV files.
- In addition, developers can use the Mobile Media API (MMAPI) **optional** package to add **video** and other rich **multimedia** content to MIDP applications.
J2ME Strengths & Weaknesses

**Advantages:**
- Wide device support
- Lack of learning curve
- Easy GUI development
- Multiple IDE’s
- Multiple emulators
- Community support

**Disadvantages:**
- Sandbox model
- API fragmentation
- Model differences
- Non native GUI
- No access to all native resources
- Not as fast as native
Summary

- MIDP applications **provide the foundation** for graphical and intuitive applications.
- MIDlets are **portable** across compatible mobile devices that support the same optional APIs.
- The **GUI is optimized** for the small display size, varied input methods, and other features of modern mobile devices.
- MIDP provides **intuitive navigation** and data entry by taking full advantage of phone keypads, extra buttons/arrow keys, touch screens, and small QWERTY keyboards.
- MIDP applications are installed and run locally, can **operate** in both **networked** (HTTP, HTTPS, datagrams, sockets) and **unconnected** modes.
- MIDP can **store** and manage persistent local **data securely**.
- MIDP also **supports** the Short Message Service (SMS) and Cell Broadcast Service capabilities of GSM and CDMA networks.