Introduction to Gaming with J2ME

Author: Anurag Gupta
Mindfire Solutions, www.mindfiresolutions.com

Table of Contents

1. Introduction

Mobile devices have a limited processing power, low heap and application size. In order to enable these devices with Java technology, SUN introduced Java 2 Micro Edition (J2ME).

J2ME is a subset of J2SE with some API's added specifically for the wireless devices.
Just like for J2SE we have JVM (Java Virtual machine) to run the java applications; the J2ME applications run on KVM (kilo Bytes Virtual machine), which is basically a subset of JVM having limited resources, for eg, lacking support for floating point calculations.

Every year a number of mobile devices are introduced in the market. These vary in screen size, processing speeds, operating systems, heap and application size limitations, etc. Hence, making a native application that can run on all these handsets is near to impossible. Here's where J2ME comes to fore!!

Properly written J2ME applications can run on a variety of handsets, independent of OS, provided these handsets support J2ME. The most we may need to do is tweak the resources (art and sound) as per the phone specifications, while the same codebase may cover a vast number of devices. The process of making an application suitable for another handset is termed as "porting".

2. Target Audience

This document is intended for Java developers who are looking for a quick introduction to J2ME and a prelude to game programming.

3. J2ME TECHNOLOGY

J2ME is build upon the configurations, profiles and other optional packages.

Configuration is the set of basic APIs on top of which the additional packages are built. Configuration tells you how big the KVM is and helps in the interaction with the device (via APIs).

There are two types of versions available as of now:

- CLDC 1.0
- CLDC 1.1

CLDC stands for Connected Limited Device Configuration.

Profile gives you information about the mobile device and extends capabilities of configuration. It gives the control for the user interface, records storing etc. As of now there are two versions available

- MIDP 1.0
- MIDP 2.0

MIDP stands for Mobile Information Device Profile.
One major difference between the 1.0 and 2.0 is the advanced game APIs and the better key controls support 2.0 provides.

Optional Packages are the packages which Vendors and Manufacturers put on the specific devices to

support additional features for ex Bluetooth, File System access, Camera APIs etc.

Since most of the devices currently available in market support MIDP 1.0, it is always better to base your development on it, unless you have specific requirements to use higher version or if you don't plan to support the older devices. This also helps during the porting process.

4. Tools and SDKs

- JDK 1.4 or higher (available from Sun's website)
- J2ME Wireless toolkit (available from Sun's website)
- Emulators (Device Specific)
- Proguard or Retroguard (for obfuscation)
- Development IDE (generally we use Eclipse being open source and easy to use)

Emulators can be downloaded from the vendor's site. Sometimes it may happen that a specific emulator is not available, in that case you may need to work with any other emulator that provides you with similar specifications, esp the screen dimensions, heap and application size.
Remember that you can only verify the user interface of your game by using emulators. The emulators do not reflect the actual game speed (frame per second, fps). So make sure to test your application on actual devices before you release for testing or in market.

Retroguard or Proguard are for obfuscation purposes. They reduce your application size by removing the unwanted code, shorten the variable names and compress the graphics too if applicable.

5. Making your first J2ME application

- Start the toolkit.
- Select the New Project Tab.
- Insert the Project Name and the Midlet Class Name. Click OK.
- In the settings window, select the Midp 1.0 and Cldc 1.0 check box and select Ok.
- Your New Project will get created.

Project Folder Structure

- Bin
- Src
- Classes
- Res
- Lib

Bin: folder contains your jad, jar and the jar manifest file.

- JAD (Java Application Descriptor) file is your configuration file containing information like Midlet Class Name, Jar Size, Icon Info, Jar Size, Vendor etc.
- JAR (Java Archive) is the file that contains the application files and is deployed on the phone along with JAD file.

What is the need of a JAD file?
It is actually used when you are hosting the application on Internet. When you download application via OTA, then firstly the JAD file is sent to the phone. Using the information in the JAD file, user is prompted with information about size of application, security certificates, company information etc. If the user approves then only the JAR file starts downloading.

Src: Your java (source) files are kept inside this folder.

Classes: When you compile source files using toolkit all your class files will be created inside the classes folder.

Res: All the resources, graphics, sounds etc are kept in this folder.

Lib: Any extra library (jar) files used by your application which may not exist on the phone, should be kept inside this folder.

When you package the application by selecting the "Create Package" option from the toolkit, the files from *classes*, *lib* and *res* folders are packaged into a single jar file.

Hello World Example:

MyMidlet.java

```
public class MyMidlet extends MIDlet{

        private Display display;
        private MyCanvas canvas;

        //starting the application
        public void startApp(){

        display = Display.getDisplay(this);
                canvas = new MyCanvas(this);
                display.setCurrent(canvas);
        }

        //pausing method for the midlet
        public void pauseApp(){
                notifyDestroyed();
        }

        //destroying method for the midlet
        public void destroyApp(boolean isDestroy){
                notifyDestroyed();
        }
}
```

MyCanvas.java

```
public final class MyCanvas extends Canvas
{
        private MyMidlet midlet;

        MyCanvas(MyMidlet midlet)
        {
                this.midlet= midlet;
        }

        public void showNotify()
        {
                repaint();
        }



public void paint(Graphics g)
        {
                g.drawString("Hello World", (getWidth() >> 1),
                (getHeight() >> 1),      Graphics.TOP|Graphics.HCENTER);
        }

        public void keyPressed(int keycode)
        {

        }
}
```

Every J2ME Gaming Application needs to have one Midlet class and one Canvas class.
MIDlet contains the entry point which is called when you invoke the application.
Canvas class is used to prepare the UI that is shown to the user when application starts running.

When this program is compiled and run, it will display "Hello World" at the center of the device screen.

As you might have seen in the example, we have created one midlet and one canvas class. We need to get the display object for the current Midlet and show the current screen which in this case is the canvas. Then the paint method is called and it paints the "Hello World" string at the center.

Isn't that a simple job?

6. Typical Game Application Class Structure

Following are the basic classes you require when you are developing a game:

- Midlet
- Canvas
- Data
- Engine
- Sprite

Midlet
As previously discussed, every J2ME application needs to have Midlet as an entry point for the application.

Canvas
Whatever you need to show to the user, you need a canvas class for that. Override the paint method to customize the application view.

Data
Many times you require constant data while developing an application. Make sure that all such data is kept in the Data java file as *"final static"* and you use these variables everywhere else inside the application. This will come handy when you need to change a value or when you are doing porting of the application.

Engine
This is the place where you keep your game logic.
You may need to port your application to various devices and sometimes you may need to use device manufacturer / device specific APIs too, eg. FullCanvas class from Nokia. Hence, it is better to separate out the game logic from the view and other helper classes. This way you won't need to change the game logic when you port to other devices.

Sprite
Since most of the games require animation of characters, so you need a sprite class for that. A sprite class has all the information required to show the various stages of a character at different points in time / screen.

MyMidlet.java

Same as in the previous example

MyCanvas.java

```
public final class MyCanvas extends Canvas
{
        private MyMidlet midlet;
        //define all your pages here like
        // splash pages , menu page, help page , game page etc

        private MyEngine engine;

        MyCanvas(MyMidlet midlet)
        {
                this.midlet= midlet;
        }

        public void showNotify()
        {
                repaint();
        }

        public void paint(Graphics g)
        {
                switch(currentPage)
                {
                        case SPLASH_PAGE :
                                //paint splash page
                        break;

                        case HELP_PAGE:
                                //paint help page
                                g.drawString("Test", Data.MENU_TOP_X,
                        Data.MENU_TOP_Y,
                                Graphics.TOP | Graphjics.HCENTER);
                        break;
                }
        }

        public void keyPressed(int keycode)
        {

        }
}
```

Data.java

```
public final class Data
{
        public final static int MENU_TOP_X = 60;
        public final static int MENU_TOP_Y= 70;
}
```

MyEngine.java

```
public final class MyEngine implements Runnable
{
        private MyCanvas canvas;
        //Engine States
        private final static byte STARTED = 1;
        private final static byte PAUSED = 2;
        private boolean gameRunning = false;

        MyEngine(MyCanvas canvas)
        {
                this.canvas = canvas;
                loadGameImages();
        }

        private void loadGameImages()
        {

        }

        private void start()
        {
                Thread th = new Thread(this);
                th.start();
        }

        public void run()
        {
                gameRunning = true;

                while(gameRunning)
                {
                        switch(currentState)
                        {
                                case STARTED:

                                        doWork();
```

```
                                break;

                                case PAUSED:

                                break;
                        }
                }

        }

        private void doWork()
        {
                //creating sprites
                //updating the sprites
                //calculating the collision
                //destroying
                //canvas.repaint();
        }

        public void paint(Graphics g)
        {
                //paint the game according to the current state
        }
}
```

MySprite.java

```
public final class MySprite
{
        private int currentFrame;

        public void update()
        {

        }

        public void paint(Graphics g)
        {

        }
}
```

Above example illustrates the basic game structure. You can add your own custom methods in the existing classes or use these classes as a template according to your game requirements.
Hope this document helps you in getting started with programming with J2ME, specifically games, which is the most widely used implementation of J2ME.