



University of Osnabrueck

---

**Google maps on mobile devices with J2ME**

---

Li Wang

Supervisor:

Prof. Dr. Oliver Vornberger

Prof. Dr. Helmar Gust

Department of Cognitive Science

University of Osnabrueck

07. 27. 2006

## **Acknowledgement**

I would like to express my sincere gratitude to many people for their guidance and help, especially I want to thank

- Mr. Prof. Dr. Oliver Vornberger for helping me to choose such an interesting topic and his good supervision.
- Mr. Prof. Dr. Helmar Gust for his good supervision and patient guidance.
- Mr. Patrick Fox for his guidance during the whole course. Thanks for having read the draft of this thesis and having made his precious comments and suggestions.
- Mr. Reid for his source code of GMapView. Due to his generous sharing I do not have to develop the whole midlet suite from very beginning.
- Mr. David Song for his help in English.

I also want to thank my family for their support and encouragement.

## Table of Contents

1 Introduction.....	4
2 Background Knowledge.....	5
2.1 Three editions of Java .....	5
2.2 Configuration .....	6
2.3 Profile.....	7
2.4 Optional Packages.....	8
3 Development Environment .....	11
3.1 Download .....	11
3.2 Installation and Integration .....	12
4 Packages in MIDP and CLDC .....	17
4.1 Package javax.microedition.midlet and lifecycle control.....	17
4.2 Package javax.microedition.lcdui and user interface system .....	18
4.3 Package javax.microedition.rms and record management.....	22
4.4 Package javax.microedition.io and generic network system.....	25
5 Google maps on cell phones .....	29
5.1 Introduction.....	29
5.2 Project Aims.....	29
5.3 Report Outline.....	29
5.4 System Analysis .....	29
5.5 System Design .....	30
5.6 System Implementation .....	33
5.7 About the assistant classes .....	41
5.8 Test the program.....	41
5.9 Conclusion .....	42
6 Conclusion .....	45
A References .....	46
B Figure list.....	48
C Content of CD-ROM.....	49

### **1 Introduction**

This thesis is a short presentation about the topics of j2me (java 2 micro edition) which is developed by Sun Microsystems to provide a portable application environment for embedded devices such like cell phones and PDAs. The rest of this thesis will be divided into 5 major parts.

Part 2 will give you an overview of J2ME platform including the history thing and three basic concepts: configuration, profile, optional package.

Part 3 will introduce you how to develop an actual J2ME application – Java midlet. What do we need in our computer and how to deploy it.

Part 4 will give you a short introduction to the basic packages available in CLDC and MIDP1.0.

Part 5 is the largest and the most important part. In this part, I will show you an actual Java midlet that I have developed which can show google maps on our cell phones.

Part 6 is the conclusion of the whole thesis.

This thesis assumes that you have some familiarity with java language.

## 2 Background Knowledge

### 2.1 Three editions of Java

In the beginning of 1990, Sun Microsystems developed a programming language which was called “OAK” for the consumer devices. It becomes today a more general purpose programming language known as Java. But it is not a whole part, actually from Java 2, it was split into 3 distinct parts: J2EE for the enterprise usage, J2SE for the general desktop market and internet applications and J2ME for embedded devices. You can find the following figure everywhere on the web, it shows the relationship between these 3 editions clearly:

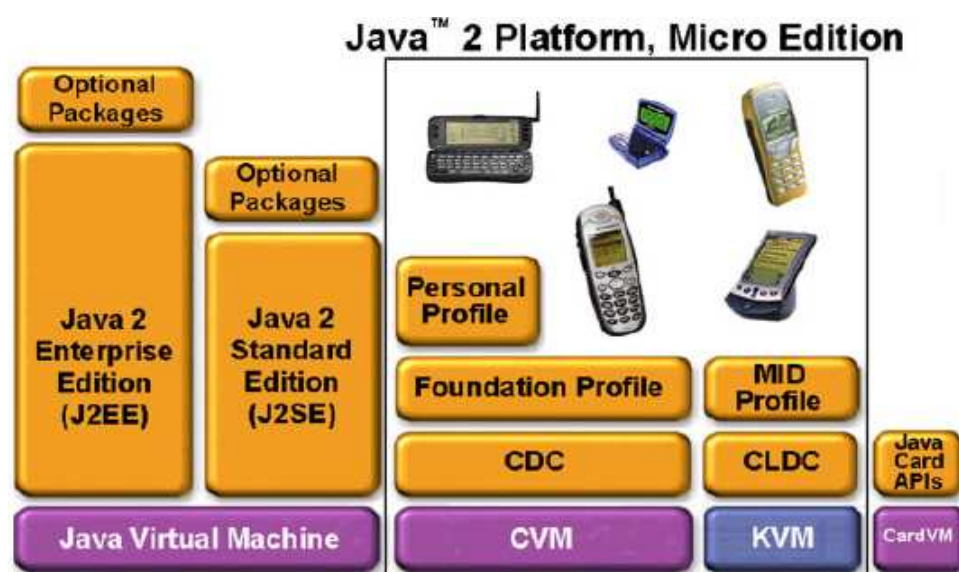


Figure 1: Three Editions of Java

But J2ME is also not a single entity, because those devices that it has to face have entirely different run time environment. For example it could be with or without keyboard, display, network interface. Sun Microsystems introduces then 3 important concepts to deal with this problem: Configuration, Profile and Optional Package. We must understand them well because they determine the features of Java that you can use, which application programming interfaces (APIs) are available, and how your applications are packaged. Before I start to introduce these three concepts there are two abbreviations that I think you should know: JCP and JSR.

**JCP and JSR:** Actually Sun is the ultimate authority for Java, but much of work in defining and completing this platform is done by the *Java Community Process* (JCP). Through JCP the organizations and individuals can participate in the definition process of different parts of the Java. This is the process: A specification request (known as a JSR) is submitted with a specific proposal to extend the Java platform. If the JSR is accepted for development, an Expert Group is formed to define a formal specification for the JSR. When ready, the specification is published. For more information about the JCP, see the JCP Web site at [JCP].

## Background Knowledge

---

From this site, you can get a list of all JSRs that have been defined or are in the process of being defined including all J2ME related JSRs.

### 2.2 Configuration

All of the devices that J2ME faces can be divided into two categories according to the characteristics like the amount of memory available, the processor type and the network connection type. J2ME defines for each of them the virtual machine and a set of core classes. They are called configurations. There are currently two kinds of configurations: CLDC (Connected Limited Device Configuration) and CDC (Connected Device Configuration). The characteristics of those devices they target separately are listed below so that you can easily find the differences between them:

CLDC targets the devices with

- Very simple user interface
- Low level memory budgets (160 kB to 512 kB)
- 16 or 32 bit processors
- Wireless communication
- Example: cell phones, PDAs

CDC targets those devices with

- Large range of user interface capabilities
- Memory budgets in range of 2 to 16 megabytes
- 16 or 32 bit processors
- Connectivity to some type of network
- Examples: TV set-top boxes, Internet TVs

As a software developer we are of course interested in which class libraries are available in these two configurations. Note that the CDC is a superset of CLDC. The CDC includes all of the classes defined by the CLDC. Java Midlet is J2ME program that runs on cell phones based on CLDC. For this purpose we have to learn to live within the restrictions imposed by CLDC. There are currently 2 versions of CLDC. CLDC 1.0 is defined by JSR-30 in JCP. CDLC 1.1 is defined by JSR- 139.

1) CLDC 1.0. It includes 4 packages:

- `java.io`
- `java.lang`
- `java.util`
- `javax.microedition.io`

The first three packages can be seen as the subset of corresponding J2SE packages. But there are also a few differences between them. The main differences are as follows:

- CLDC1.0 do not implement finalization so there is no `finalize()` method in `java.lang.Object` in CLDC1.0.

## Background Knowledge

---

- CLDC1.0 includes the majority of the exceptions defined by the J2SE `java.lang` package, but most of the error classes have been removed. The CLDC only defines three of these error classes: `java.lang.Error` , `java.lang.OutOfMemoryError` and `java.lang.VirtualMachineError`.
- CLDC1.0 does not support floating point. Methods taking or returning floating point values are removed from all classes.

The package `javax.microedition.io` deals with the generic connection framework. This thesis will discuss the package later in detail.

2) CLDC 1.1 Today more and more cell phones support CLDC1.1. The main difference between CLDC1.0 and CLDC1.1 is the version 1.1 supports floating point.

Besides the class libraries, configuration defines also the virtual machine. Perhaps the biggest difference between the CDC and the CLDC is that the CDC requires a full-featured Java virtual machine. Sun has released a new Java VM, the Compact VM (CVM), for this purpose—it is more portable and efficient than the standard VM. The virtual machine that CLDC defines is called KVM (kilobyte virtual machine). It is not a full java virtual machine. The main difference between them perhaps is the class loading mechanism. Class loading in J2SE is performed by class loaders, including application-defined class loaders. By contrast, the CLDC specification requires implementations to provide their own class loading mechanism that cannot be overridden or extended by application code in the consideration of security. Sun's CLDC reference implementation includes an example implementation of this functionality. We will refer it as a Java Application Manager (JAM). It is sometimes also called Application Management Software(AMS).

### 2.3 Profile

The next important concept in J2ME is profile. As we already learned, the configuration helps already a lot to distinguish different types of devices. But they have to be refined further because a same configuration might cover devices with very different user interface and input output system. A profile complements a configuration by adding additional classes that provide features appropriate to particular type of device. These profiles define mostly the application life cycle model, the user interface, and access to device specific properties. Profiles are implemented on top of a configuration. Both J2ME configurations have one or more associated profiles. Applications are written for a specific profile, as profiles are assembled for a specific configuration. Both profiles and configurations define a set of Java API classes which can be used by the applications. There are many kinds of profiles. Such as MIDP (Mobile Information Device Profile) is based on CLDC. Foundation Profile is based on CDC. Personal Basis Profile and Personal Profile are based on Foundation Profile.

Java Midlets are based on MIDP. They can be called MIDP applications on cell

## Background Knowledge

---

phones. Let us learn more about it. There are currently two versions of MIDP. One is MIDP 1.0. It defines all the core classes of MIDP applications. Besides the four packages which are inherited from CLDC. MIDP defines more three packages:

- `javax.microedition.lcdui`
- `javax.microedition.midlet`
- `javax.microedition.rms`

The following is a short introduction of these three packages. The third part of this thesis will give you more detailed information.

The MIDP has no AWT and Swing component of J2SE. All classes that handle with user interface are in `javax.microedition.lcdui` package. It includes “high-level” APIs and “low-level” API.

As we all know Java applications that run on MIDP devices are known as MIDlets. A MIDlet consists of at least one Java class that must be derived from the MIDP-defined abstract class `javax.microedition.midlet.MIDlet`.

A J2SE application typically stores state in local files that are quickly and easily accessible from the hard drive. Mobile devices, however, do not have local disks. The MIDP specification has to provide a persistent storage facility so that information can be preserved while a MIDlet is not running or when the device is turned off. The package `javax.microedition.rms` handles this stuff.

The MIDP 2.0 includes many enhancement and additions to MIDP1.0. For example it includes a set of media APIs and game APIs. It is very exciting for the mobile phones game developer. The packages that are extended in MIDP 2.0 are listed here:

- `javax.microedition.lcdui.game`
- `javax.microedition.media`
- `javax.microedition.media.control`
- `javax.microedition.pki`

The emphasis in my thesis will be MIDP 1.0. If you are interested in MIDP2.0 refer to this web site: [MIDP2. 0]

### 2.4 Optional Packages

Besides the configuration and profile, the optional packages define also class libraries for specific usage. We can say that it is the third category of J2ME component. They are layered on top of a profile. They extend run time environment but are not so universal like configuration or profile. Some devices support them. Some don't. Therefore they are called “optional”. One device can support multiple optional packages.

Now there are many optional packages to be available. Today there are nearly 80 J2ME-related specifications at various stages in the JCP and many of them define optional packages. For complete list of optional packages, refer to the web site [JSR].

The following are short introduction of the more interesting and important optional



## Background Knowledge

---

packages:

- JSR 120 : Wireless Messaging API. It defines APIs that provide access to wireless communication resources.
- JSR 135 : Mobile Media API (MMAPI). It defines API that provide access and control of multimedia resources.
- JSR 172 : Web Service Specification. It provides access to web services.
- JSR 177 : Security and Trust Services API. These APIs provide security services to J2ME devices.
- JSR 184 : Mobile 3D Graphics API. These APIs provide developing and controlling of 3D applications.

### SVG

When we talk about optional packages, worth noting that now more and more cell phones support the SVG technology. SVG is abbreviation of scalable vector graphics. It is XML syntax for rich 2D animated, dynamic and interactive graphics. Here are some advantages for using SVG.

- First of all, it is scalable, so it adapts to different displays.
- Second it is animated.
- Third, it is interactive to the user.
- Fourth, it allows the user to search for text in the image.

The following are some situations that we use SVG on our cell phones.

- The maps are probably the most important applications.
- We can print your presentations or office documents to SVG and load it into our cell phones.

Just like J2ME can be seen as a subset of J2SE. The SVG Profile that is used on cell phones is called SVG Tiny. It is just subset of SVG full. You can visit the following web site to see how SVG Tiny is defined: [SVGMobile].

In J2ME, there is a java specification request JSR 226 responsible to show and handle SVG. It builds on top of SVG Tiny. On the following page, you can find the complete documentation of JSR 226: [JSR226].

As I know, Sun did not until now provide any reference implementation of it. Nokia did. You can download this implementation on this page: [RIJSR226]

I give you here just a simple introduction of API structure of JSR 226. There are core part and advanced part. The core part consists of basic graphics operation such as loading an SVG file, rendering it and zooming. In advanced part you have a direct manipulation of different elements. For example, in an SVG file you can go to individual shapes and manipulate them, change their color, update. Rotate downward and transform them. It can be shown in the following figure. It comes from the learning program of Sun Microsystems: [JSR226Nokia].

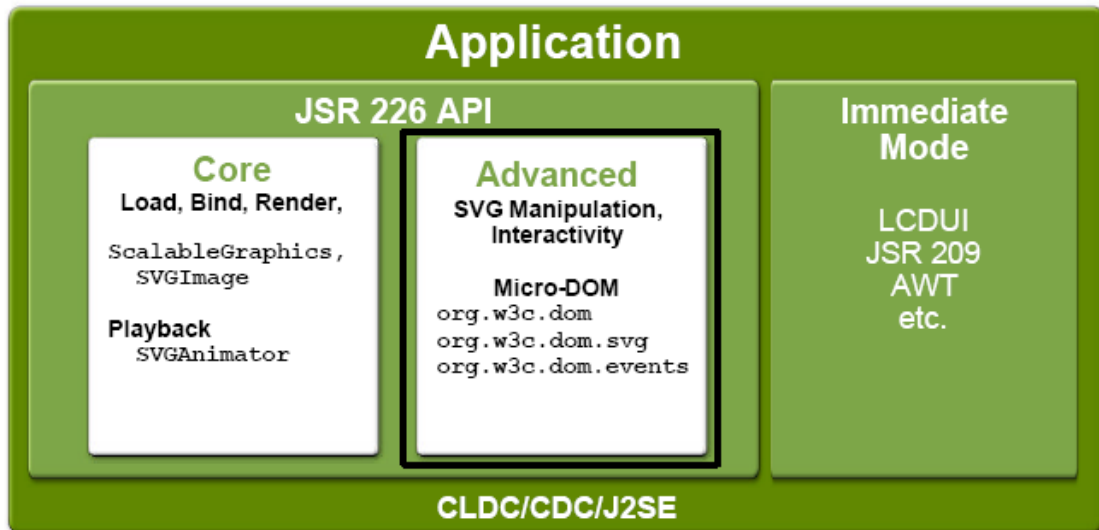


Figure 2: JSR 226 AP

### 3 Development Environment

#### 3.1 Download

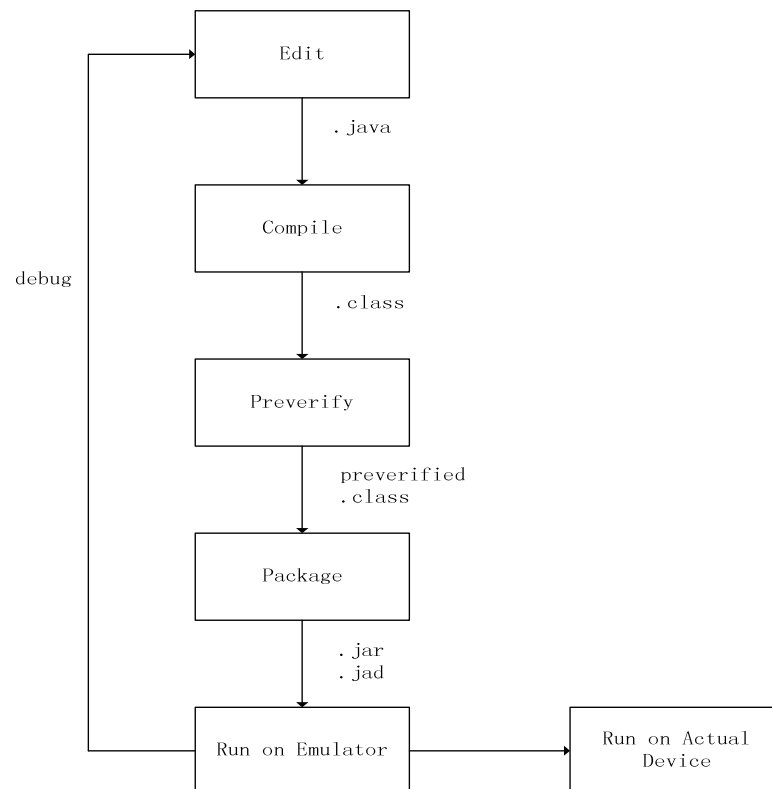


Figure 3: Standard Development Process of Java Midlet

This figure shows us a standard development progress of a java midlet. The whole left part we should do it in our PC. Code writing, compiling we are already very familiar with, the difference of midlet development and normal java applet development is preverifying. For regular Java programs, verification is performed by checking whether or not it remains stable during execution. But Because J2ME is designed for small devices with limited memories, much of the usual Java verification has been removed from the virtual machine. As a result, it is necessary to preverify your J2ME application before deployment. Thus all we must do at run time is to make sure that the class has not changed since preverification. That saves a lot of time. To do all these jobs, what do we need in our computer?

#### 1) J2SE SDK

First of all we need the J2SE SDK (You will sometimes hear developers refer to this as the JDK, or Java Developer's Kit, but the current name is J2SE SDK.). J2ME Wireless Toolkit runs upon the java platform it provides and it includes a Java compiler.

It can be downloaded from this page: [JSDK].

#### 2) WTK

We need J2ME platform, CLDC, MIDP and an Emulator. Sun Microsystems provides

## Development Environment

---

a J2ME Wireless Toolkits (I will refer it as WTK later) for J2ME developer. It consists of build tools, utilities, and a device emulator. Or you can use what the device manufacture provides.

It can be downloaded from this page: [WTK].

### 3) IDE: ECLIPSE AND ECLIPSE ME

Actually WTK already provides a set of J2ME development tools. But most developer used to using integrated development environment. It really provides much convenience. I choose here a free and open source IDE: Eclipse. To use Eclipse to develop Java Midlet, we need a plugin -- EclipseME.

You can download Eclipse from this page: [ECLIPSE].

And EclipseME from this page: [EclipseME].

### 3.2 Installation and Integration

The installation is easy. After the installation we have still two integration jobs to do. First integrate ECLIPSEME to ECLIPSE. Second integrate WTK to ECLIPSE.

#### 1) ECLIPSE and ECLIPSEME

You can visit this web page to see the Steps to integrate Eclipse and EclipseME: [EclipseMEIn].

#### 2) Eclipse and WTK

After restarting of Eclipse, the system will automatically guide you integrating WTK to Eclipse, just press "Next" until last dialog window, then press Finish.

If the plug-in is properly installed, you will find this result when you select Windows→Preference→Device Management

Notice: If the system does not start this progress automatically, just click the "Import" button in the following dialog window and do what the system ask you to do.

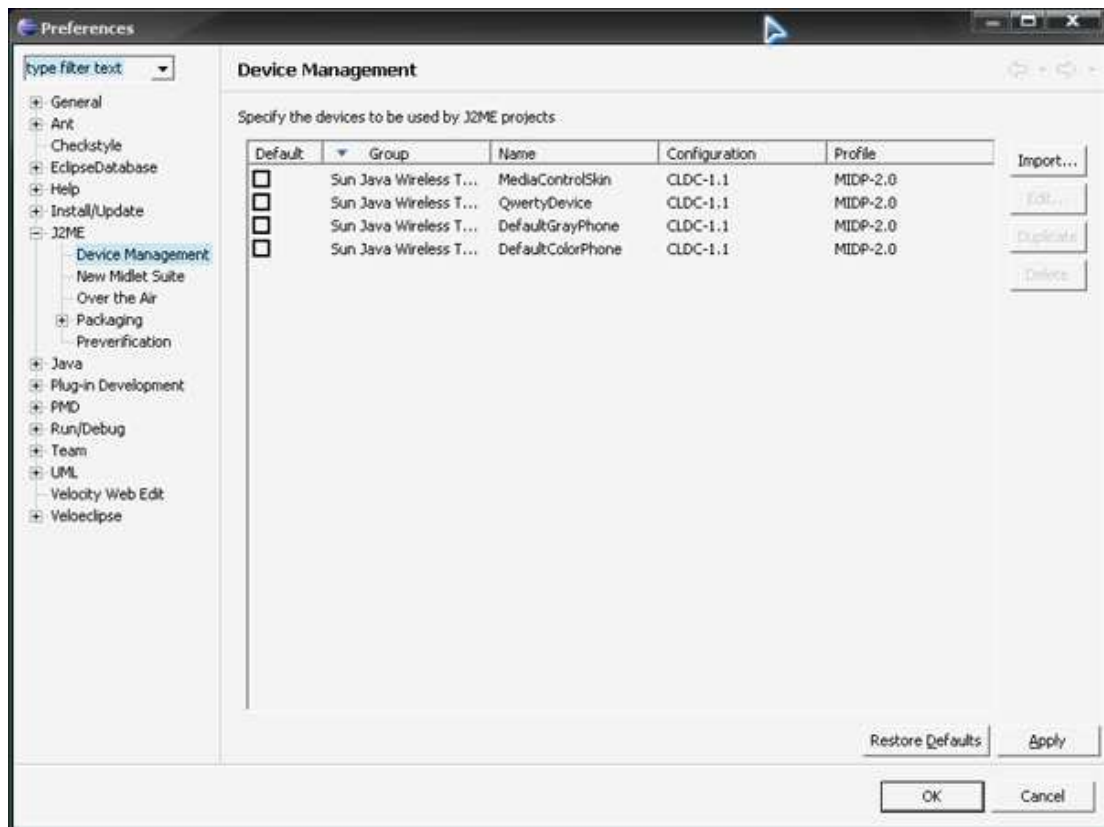


Figure 4: WTK Properly integrated into EclipseME

### 3) Midlet project creation

Now we have all of the development environments installed. We try to create a midlet project now. It is just like creating a java package. The difference is if you select File→New→Project, there is an option J2ME. Choose J2ME→ J2ME Midlet Suite.

Then we create in this package a new class. Select New→Others→J2ME→J2ME Midlet.

You can visit this site to see the whole steps: [MidCreation].

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Hello extends MIDlet{
private Display display;
public Hello(){
display =Display.getDisplay(this);
}
public void startApp(){
TextBox t = new TextBox("Hello", "Hello", 256, 0);
display.setCurrent(t);}
public void pauseApp(){}

public void destroyApp(boolean unconditional){}
}
```

## Development Environment

---

I write here a piece of code, the result is to show “Hello” on the screen of the cell phones. To run this midlet in the emulator, choose “Run” from the “Run” menu.

Here is the result



Figure 5: Hello.jar on Cell Phone Emulator

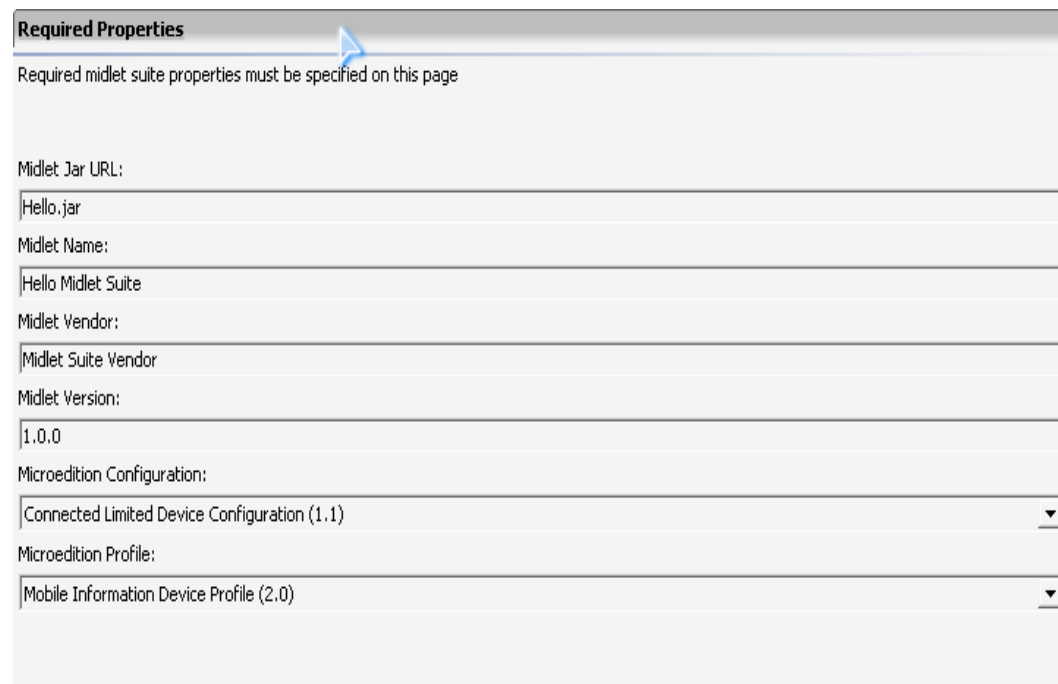
4) Now it's time to package. That can be done very easily in Eclipse. Right click on the project name in the Package Explorer, and select "J2ME | Create Package". A new folder which is called “deployed” will be then created. In this folder are there two files – a JAR file and a JAD file.

The pre-verified class files in the "verified" directory and the resource files in the "res" directory are included in the JAR file. You maybe are not familiar with JAD file. All the packaging information that tells the device what is in the JAR must be supplied in the JAR's manifest file. In order to make use of this information, however, the device must download the whole JAR file and extract the manifest. After that the manifest the MIDlet-Name, MIDlet-Version, and MIDlet-Vendor attributes and the optional MIDlet- Description can be showed to us. These attributes allow the user to decide whether the MIDlets should be installed. However, the JAR for a MIDlet suite might be quite large. To save time and money some of the attributes from the manifest file, together with extra information, is duplicated in the JAD file. Instead of downloading the whole JAR, a MIDP device first fetches its JAD file, which is much smaller than the JAR and can be transferred quickly. The device then displays the

## Development Environment

---

JAD file's contents to the user so that she can decide whether to fetch the JAR file. The JAD contains some attributes that come from the manifest file and others that do not appear in the manifest. The common attributes are as follows:



**Required Properties**

Required midlet suite properties must be specified on this page

Midlet Jar URL:  
Hello.jar

Midlet Name:  
Hello Midlet Suite

Midlet Vendor:  
Midlet Suite Vendor

Midlet Version:  
1.0.0

Microedition Configuration:  
Connected Limited Device Configuration (1.1)

Microedition Profile:  
Mobile Information Device Profile (2.0)

Figure 6: Example of JAD File

### 5) Deploy the midlet

Until now all we must do in our computer has been done. The next step is to deploy our midlet jar file to the device. There are several ways to do that.

- Through cable
- Through BlueTooth
- OTA

The first two ways are different from device to device. You can look up the concrete steps in the manual that the device producer provides you. The third way OTA is the abbreviation of “Over-the-Air”. First put the JAD file and JAR file on a web server which supports Java Midlet, then download it from our cell phones:

We can also do it on the Sun J2ME Wireless Toolkits.

- a) Choose “Start → Programm → J2ME Wireless Toolkits 2.2 → OTA provisioning”
- b) Press “Apps” and select “Lauch”
- c) Enter the URL to the page containing the link to JAD file and press “Go”.
- d) When the page is displayed, you can choose which JAD file to open. Then press “Install”.
- e) JAD file is displayed. The name of the application, its size, version, vendor and the full location of it JAR file are displayed. You can choose to cancel the installation or to install it.

The real device works in the same way. You point its browser directly to JAD file and

## Development Environment

---

the device lets you choose if you wish to install the package. If you wish to, it will download the JAR file and install it.



### 4 Packages in MIDP and CLDC

In this part, I will give a short introduction to the three basic class packages in MIDP1.0:

- `javax.microedition.midlet`
- `javax.microedition.lcdui`
- `javax.microedition.rms`

and one important package in CLDC:

- `javax.microedition.io`

#### 4.1 Package `javax.microedition.midlet` and lifecycle control

I have already talked about the OTA provisioning of a java midlet. This process is controlled by the AMS (Application Management Software) of the cell phones. AMS not only control the download and installation of java midlets. It also controls the lifecycle of a midlet. In a Midlet suite there must be a class that derives from the abstract basic class `javax.microedition.midlet.MIDlet`. There are three abstract methods (`startApp()`, `pauseApp()`, `destroyAPP()`) in this class, all midlets that extend this class must implement these three methods. AMS just control the execution of a Midlet by calling these 3 methods.

A MIDlet is always in one of three states: paused state, active state, or destroyed state. When a MIDlet is loaded, it is first in its paused state. Its public, no-argument constructor is invoked. If the MIDlet throws an exception during the execution of its constructor, the MIDlet will be destroyed. If the MIDlet does not throw any exception, AMS calls `startApp()` method and its state is changed from paused to active. The active state is the functional state. Every MIDlet wants to be in this state! This is when the MIDlet can do its functions, possess the device resources and , do whatever it wants to do. But if necessary , AMS can at any time put a MIDlet into the paused state by calling the method `pauseApp()`. On a cell phone, for example, this might happen when a call comes and the software needs to set the phone's display free so the user can answer the call. What a midlet should do in his paused state depends on what you write in the `pauseApp()` method. But generally, it should release any resources it is holding and save the current state so it can restore itself when it is reactivated later. When you finished this call, AMS will call the `startApp()` again so that the Midlet enters the active state again. Be careful here, the active state can be entered again and again, anything that should only be initialized once, you should do it in the constructor. When AMS decide to terminate a MIDlet, it calls the MIDlet's `destroyApp()` method. It can be done whether the midlet is in paused state or active state. Interesting here is this method has a Boolean argument. If it is true, the MIDlet should release all the resources that it has allocated, terminate any background threads, and stop any active timers. If it is false, the MIDlet can indicate that it wants to continue by throwing a `MIDletStateChangeException`.

### 4.2 Package `javax.microedition.lcdui` and user interface system

When we talk about user interface, first we thought we could use the subset of Abstract Windows Toolkits or Swing component from J2SE. But it turns out impossible because of the limited memory and processor resources. On the other hand, cell phone users do not expect to be able to work with more than one window at the same time since the screen size is limited. So J2ME expert group introduced several classes that are much simpler and screen-based. All those classes are in the package `javax.microedition.lcdui`. They are grouped into two categories: High-level APIs which are subclasses of abstract class `Screen` and low-level APIs which are subclasses of abstract class `Canvas`. Before I begin to introduce the user interface, I will take a simple program to explain the high level Event handling mechanism in J2ME. Look at this piece of code.

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

public class HLEventMidlet extends MIDlet implements
CommandListener {
    private Command exitCommand;

    private Command info1Command;

    private Command info2Command;

    private Display display;

    public HLEventMidlet() {
        display = Display.getDisplay(this); // get a DISPLAY object
        exitCommand = new Command("Exit", Command.SCREEN, 1);
        info1Command = new Command("Info1", Command.SCREEN, 2);
        info2Command = new Command("Info2", Command.SCREEN, 2);
    }

    public void startApp() {
        TextBox t = new TextBox("Hello MIDlet", "Test string", 256,
                                0);

        t.setCommandListener(this);
        t.addCommand(exitCommand);
        t.addCommand(info2Command);
        t.addCommand(info1Command);
        display.setCurrent(t);
    }
}
```

## Packages in MIDP and CLDC

---

```
public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
}

public void commandAction(Command c, Displayable s) {
    if (c == exitCommand) {
        notifyDestroyed();
    } else if (c == infoCommand) {
        notifyDestroyed();
    }
}
}
```

As we know, a MIDP device displays only a single “window” at a time. We say that even though there is more than one MIDlet running in a device at the same time, only one of them can have access to the screen. Here we use the `Display` class. Usually a Midlet invokes the static method `getDisplay(Midlet m)` in his constructor or `startApp()` method to get a `Display` object

```
display = Display.getDisplay(this); // get a DISPLAY object
```

This method has an argument `Midlet`, usually is `this`, the midlet itself. That means at this time this Midlet has the right to access the screen. What does it show? Usually the midlet in his `startApp()` method produce a `Displayable` Object. In this example is `TextBox`, which is subclass of the class `Screen`.

```
TextBox t = new TextBox("Hello MIDlet", "Test string", 256, 0);
// at this time, a TextBox will be showed on the screen
```

And `Screen` is the subclass of `Displayable`. And we use the instance method `setCurrent()`.

```
display.setCurrent(t); // allocate for this screen
// a displayable object
```

This method has an argument which is a `Displayable` object. In this example is this `TextBox`. That means at this moment this `t` (`TextBox`) should be on the screen. We will see it is like a string. On the other hand, as a `Displayable` object. `T` can also invoke `addCommand()` method. You can just imagine a `Command` is a menu. The user can select one from them. `T` also can invoke the method `setCommandListener()`. The argument is of the class `CommandListener`. That means if one of the commands is selected, the `actionCommand()` method of

this `CommandListener` will be executed. In this example, the `CommandListener` is the `Midlet` self. So let us see the result

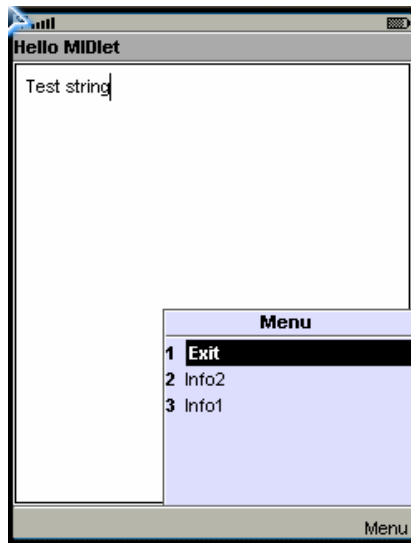


Figure 7: LCDUI Example (TextBox)

First, the textbox show on the screen. If we click “Menu” there will be three choices that we can choose. We select exit, the `actionCommand()` method of this method will be executed and the whole midlet will be terminated. This is a very simple but very typical high level event handling in Midlet. All high level interfaces interact with user in this way. There are four this kind of classes in the package `javax.microedition.lcdui` -- `List`, `Alert`, `Textbox` and `Form`. They are easy to handle but can not control the exact look of the Midlet. On opposite, using low level interface you can draw everything on the screen. But it is relative complicate. The class `Canvas` is the basic building block of the low-level API. It is an abstract class. To use `Canvas`, you have to subclass it and implement the `paint()` method. This method gets an argument, which is an instance of another low-level API class called `Graphics`. `Graphics` class provides methods that allow you draw whatever you want. Check this example code out

```
public class SimpleCanvas extends MIDlet {
    private Display display;

    public SimpleCanvas() {
        display = Display.getDisplay(this);
    }

    public void startApp() {
        MyCanvas mc = new MyCanvas();
        display.setCurrent(mc);
    }
}
```

```
    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}

class MyCanvas extends Canvas {
    public void paint(Graphics g) {
        g.setColor(255, 255, 255);
        g.fillRect(0, 0, getWidth(), getHeight());
        g.setColor(255, 0, 0);
        g.drawString("Hello", 10, 10, 0);
    }
}
```

We say that if we give the `setCurrent()` method an instance of `Canvas` as argument.

```
display.setCurrent(mc) ; // invoke
                          //mc.paint(Graphicsg)
```

The midlet will invoke the `paint()` method of this object automatically and give it an `Graphics` as argument. In this example, we first clear the screen using the method `setCollor()`.

```
g.setColor(255,255,255) ; // clear the whole screen
```

Then we make the whole screen white.

```
g.fillRect(0,0,getWidth(),getHeight()) ; // make the screen white
```

The third argument here we use the method `getHeight()` and `getWidth()` of `Canvas` class to get the whole displayable area.

```
g.drawString("Hello",10,10,0) ;
```

This line means we draw a string on the area.

```
g.setColor(255,0,0) ;
```

And this means we set at first the color of the string.  
Here is the result

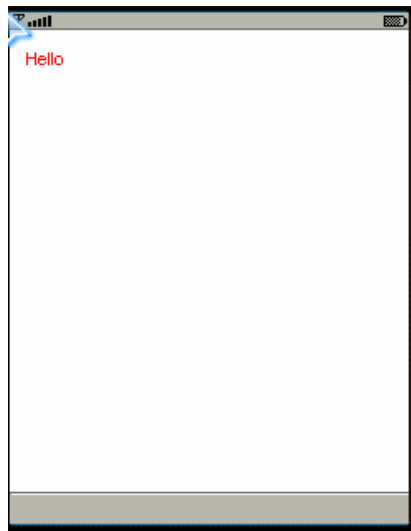


Figure 8 : Canvas Example

This is typical usage of Canvas class. Using Graphics Object, we can draw a line, draw a rectangle, draw an arc, and even an animation. In addition to the class Canvas there is also an important class in low level user interface API: Image. Simply saying, you can use the static method `createImage()` (there are four of them with different arguments) to create a immutable or mutable Image und use the method from Graphics `drawImage()` to draw it.

### 4.3 Package `javax.microedition.rms` and record management

Rms is the abbreviation of Record Management System. It provides methods that Midlet can use to save information. It is just like a small Database Management System. We know that in a general DMS, there exist a lot of tables. In RMS is there something that plays the same role which is called Record Store. As its name implies, a Record Store is a collection of records. In one Midlet suite, a record store has a unique, case sensitive name. Different Midlets in a same Midlet suite can share the same Record store. Midlets that from different Suite can not share record stores. Each record in a record store is an array of bytes and has a unique integer identifier. As I said, all classes or interfaces that handle information storage are in this package. There are

- 2 Classes : `RecordStore`, `RecordEnumeration`
- 3 Interfaces : `RecordComparator`, `RecordFilter`, `RecordListener`
- 5 Exceptions : `InvalidRecordException`, `RecordStoreException`,  
`RecordStoreFullException`,  
`RecordStoreNotFoundException`,  
`RecordStoreNotOpenException`

The most important class in this package is `RecordStore`. It provides several methods to manage such as insert, update, and delete records in a record store. The

## Packages in MIDP and CLDC

---

interface `RecordComparator` defines a `compare` method to compare two candidate records. The `RecordEnumeration` interface is responsible for enumerating records in a record store. The `MIDlet` implements the `RecordFilter` interface, defining a filter that examines a record to see if it meets application-defined criteria. Here is a simple example

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

public class RecordStoreTest extends MIDlet implements
ItemStateListener {
    private Display display;

    public RecordStoreTest() {
        display = Display.getDisplay(this);
    }

    public void startApp() {
        String dbname = "testdb";
        Form f = new Form("RS Test");
        RecordStore rs = openRSAnyway(dbname); // open the RS testdb
        if (rs == null) {
            f.append("Table open fail");
        } else {
            try {
                f.append("Last modified : " + rs.getLastModified());
                f.append("\nRS Name : " + rs.getName());
                f.append("\nNext ID : " + rs.getNextRecordID());
                f.append("\nRecord num : " + rs.getNumRecords());
                f.append("\nSize : " + rs.getSize());
                f.append("\nAvaliable      Size      : "      +
rs.getSizeAvailable());
                f.append("\nVersion : " + rs.getVersion());
                rs.closeRecordStore();
                deleteRS(dbname);
            } catch (Exception e) {
            }
        }
        display.setCurrent(f);
    }

    public void pauseApp() {
    }
}
```

```
public void destroyApp(boolean unconditional) {
}

public void itemStateChanged(Item item) {
}

public RecordStore openRSAnyway(String rsname) {
    RecordStore rs = null;
    if (rsname.length() > 32)
        return null;
    try {
        rs = RecordStore.openRecordStore(rsname, true);
        return rs;
    } catch (Exception e) {
        return null;
    }
}

public boolean deleteRS(String rsname) {
    if (rsname.length() > 32)
        return false;
    try {
        RecordStore.deleteRecordStore(rsname);
    } catch (Exception e) {
        return false;
    }
    return true;
}
}
```

First we try to open a recordstore

```
RecordStore rs = openRSAnyway(dbname) ;
```

We define the method `opRSAnyway( )` as following:

```
rs = RecordStore.openRecordStore(rsname,true) ;
```

The second parameter means that if the recordstore that we try to open does not exist, create one. Then we use some methods in the class `RecordStore` to get some attributes of this recordstore and show it on the screen:

```
rs.getLastModified()
rs.getLastModified()
```



```
rs.getNextRecordID()  
rs.getNumRecords()  
rs.getSize()  
getSizeAvailable()  
rs.getVersion()
```

In the end, we use

```
rs.closeRecordStore();
```

to tell the AMS that the thread that we are using does not use this RecordStore any longer. If we do not use this RecordStore itself any more, we use

```
deleteRS(dbname) ;
```

to delete it. Here is the result that appeared on the screen:



Figure 9: Example using RecordStore

### 4.4 Package `javax.microedition.io` and generic network system

Again because of the limited resource we must give up the `java.net` API that supports connection in normal J2SE application. J2me provides a collection of API to support for connection types which are in the package `javax.microedition.io`. Here is an example for using the `HttpConnection`:

```
import javax.microedition.midlet.*;  
import javax.microedition.lcdui.*;  
import javax.microedition.io.*;  
import java.io.*;  
  
public class HttpTest extends MIDlet {
```

```
private Display display;

public HttpTest() {
    display = Display.getDisplay(this);
}

public void startApp() {
    try {
        String url = "http://www.uos.de";
        HttpConnection hc =
            (HttpConnection) Connector.open(url);
        DataInputStream dis =
            new DataInputStream(hc.openInputStream());
        String content = "";
        int ic;
        while ((ic = dis.read()) != -1) // read the source code
        {
            content = content + (char) ic;
        }
        Form f = new Form("HTTP Test");
        f.append(content);
        display.setCurrent(f);
    } catch (Exception e) {
        System.out.println(e.getMessage());
        notifyDestroyed();
    }
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
}
}
```

First we use

```
HttpConnection hc;
hc = (HttpConnection)Connector.open(url);
```

to get the contact with the web site, then we get a `DataInputStream` from this `HttpConnection` object.

```
DataInputStream dis;
dis = new DataInputStream(hc.openInputStream());
```

Then we use the method `read()` to read the source code of this web site.

```
while((ic = dis.read())!=-1)           //read the source code
content = content + (char)ic ;
```

Finally we put all the source code on the screen:

```
f.append(content) ;
```

The result is showed as following :



Figure 10: Example using `URLConnection`

All the classes in this package can be organised as a network

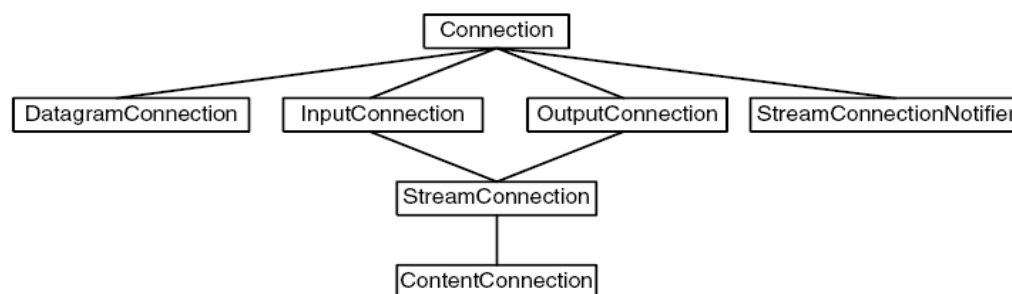


Figure 11 : The hierarchy architecture of the package `javax.microedition.io`

At the top of the interface hierarchy is the `Connection` interface, the most basic type of connection - all other connection types implement `Connection`. As you move down the hierarchy, connections become more complex and functional. For packet-based I/O the GCF defines `DatagramConnection`, and for stream-based I/O `InputConnection`, `OutputConnection`, `StreamConnection`, and `ContentConnection`. Note how `StreamConnection` implements both `InputConnection` and `OutputConnection`. At the bottom of the hierarchy is

## Packages in MIDP and CLDC

---

the `ContentConnection`, a special type of `StreamConnection` that provides content-specific information such as data length, content type, and data encoding. Finally, the `StreamConnectionNotifier` enables an application to wait for incoming stream connections, asynchronously.

### 5 Google maps on cell phones

#### 5.1 Introduction

This part is a project report. The goal of this project is to develop a MIDP application (Java Midlet) that can show google maps on mobile phones. This project intends to investigate, design, implementation and evaluation of the developed application using today mobile technologies.

#### 5.2 Project Aims

- Given a cellular phone. Enter the name of a city and show a map of the surroundings on the display.
- Navigate north, west, south and east.
- Zoom into and Zoom out of the map.

#### 5.3 Report Outline

The remainder of this part will be divided into 5 sections:

Section 5 analyzed the system requirements and the ideas to solve the main problems.

Section 6 considered the possible system architecture including the user interface system.

Section 7 discussed how the system was implemented including some important steps and main algorithm that I used.

Section 8 discussed how the system was tested and the result.

Section 9 is the last section of this report. It listed difficulties that I have encountered during the implementation and the future work that could be done to extend the system even further.

#### 5.4 System Analysis

Before software is designed, the first thing we should do is the requirement analysis.

For this midlet, that includes the following questions:

1) What is Google Maps and what can we do with it?

For this question, you can visit the following web address: [GoogleMaps].

Google Maps is a powerful online mapping technology. By visiting this web address, you can search for a business location .For example, if you search for “Pizza in New York”, you can get a list of pizzeria in New York including contact information from the system and the map of them. On this map you can navigate, zoom in, zoom out and also drag it to view the adjacent sections immediately.

2) How I fulfill the supposed tasks

My main job is: Given a name of a city, and show the map of it. So the most important thing is to find out the relationship between the city name and the map image.

If you use Opera browser to visit the Google Maps, you will find that the map that

## Google maps on cell phones

---

presents you is not a whole block. It consists of several PNG images which consist of 256\*256 pixels. This is an example of the web address of such an image:

<http://mt0.google.com/mt?n=404&v=w2.11&x=0&y=0&zoom=17>

At this time, the zoom is at the largest. Now the whole world is a whole image. When the zoom minus 1, the whole world is separated into 4 images, the addresses of these 4 images are:

<http://mt0.google.com/mt?n=404&v=w2.11&x=0&y=0&zoom=16>

<http://mt0.google.com/mt?n=404&v=w2.11&x=1&y=0&zoom=16>

<http://mt0.google.com/mt?n=404&v=w2.11&x=0&y=1&zoom=16>

<http://mt0.google.com/mt?n=404&v=w2.11&x=1&y=1&zoom=16>

From this I can infer that if we get the address of the center image as following

<http://mt0.google.com/mt?n=404&v=w2.11&x=m&y=n&zoom=z>

The addresses of adjacent images are:

<http://mt0.google.com/mt?n=404&v=w2.11&x=m-1&y=n&zoom=z>

<http://mt0.google.com/mt?n=404&v=w2.11&x=m+1&y=n&zoom=z>

<http://mt0.google.com/mt?n=404&v=w2.11&x=m&y=n-1&zoom=z>

<http://mt0.google.com/mt?n=404&v=w2.11&x=m&y=n+1&zoom=z>

<http://mt0.google.com/mt?n=404&v=w2.11&x=m-1&y=n-1&zoom=z>

<http://mt0.google.com/mt?n=404&v=w2.11&x=m-1&y=n+1&zoom=z>

<http://mt0.google.com/mt?n=404&v=w2.11&x=m+1&y=n-1&zoom=z>

<http://mt0.google.com/mt?n=404&v=w2.11&x=m+1&y=n+1&zoom=z>

This is very important for the algorithm for navigating, zooming in and zooming out of our map. Now the question is if I know the name of the city, how I can know the address of the center image. Every location in this world has its own and single latitude and longitude. If I can find the relationship between the latitude longitude coordinate system and the coordinate system that the google maps use, we can find the address of the center image for a certain city.

In order to fulfill the supposed tasks, I must know

- a) How could we get the latitude and longitude of the city according its name.
- b) How to get the address of the center image if knowing the latitude and longitude.
- c) The way to get the adjacent image, navigate our map and zoom in, zoom out

3) This is a kind of complicated software. A single thread is not enough. We must allocate multiple threads to deal with all of the tasks.

### 5.5 System Design

#### 4.5.1 User interface system

The following flow diagrams show how the proposed system's user interface works.

The proposed system's user interface will be constructed using menus

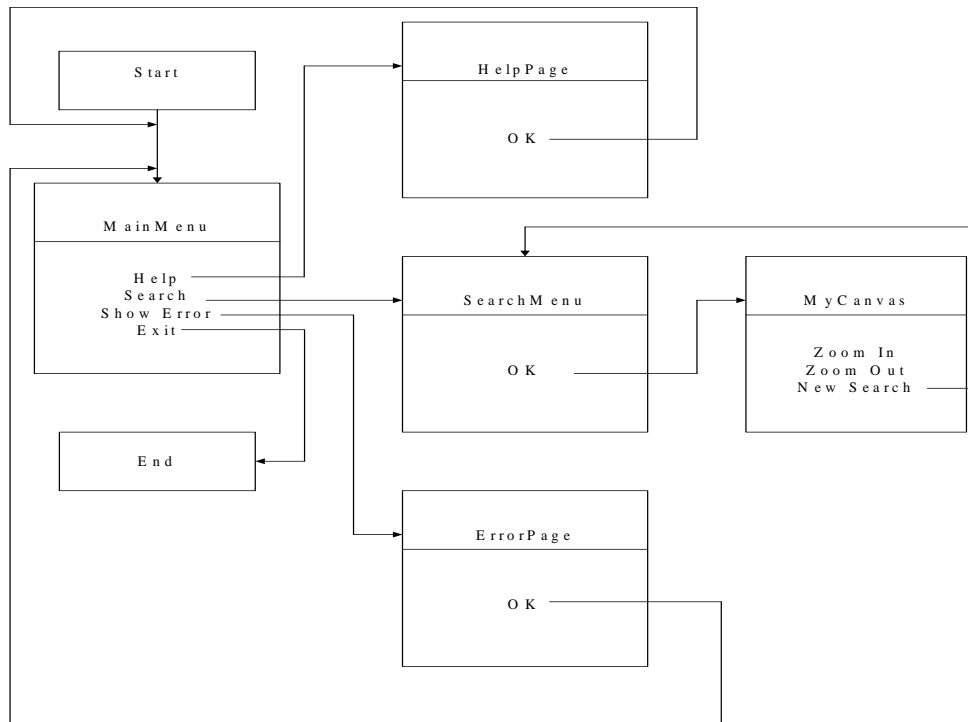


Figure 12 : Flow diagram of the user interface system

About the user interface system:

At first, the user should see the main menu. There are four choices available, if we choose “Help”, a HelpPage will be shown on the screen. It is a usage manual of this program. If we choose “show error”, the system will show us the happened errors. And we can choose “OK” from these two pages to get back to the main menu. If we choose “search” from the main menu, the search menu will be shown on the screen. There will be a text field so that the user can give the system the name of the city, and if we choose “OK” from this menu, a MyCanvas object will be shown on the screen. It is the google map of this city and its surroundings. It is also a menu, with the choices “Zoom In” and “Zoom Out” we can zoom into and zoom out of the map and if we choose “New Search”, a new Search Menu will appear to be ready for a new search.

The above flow diagram has shown all major screen displays.

### 4.5.2 Functional Part

The following sequence diagram show the process that from the name of the city that we give to get the map image that we need to be downloaded. It will be implemented through 4 classes: SearchMenu, CenterCalculator, MyCanvas, TileDownloader

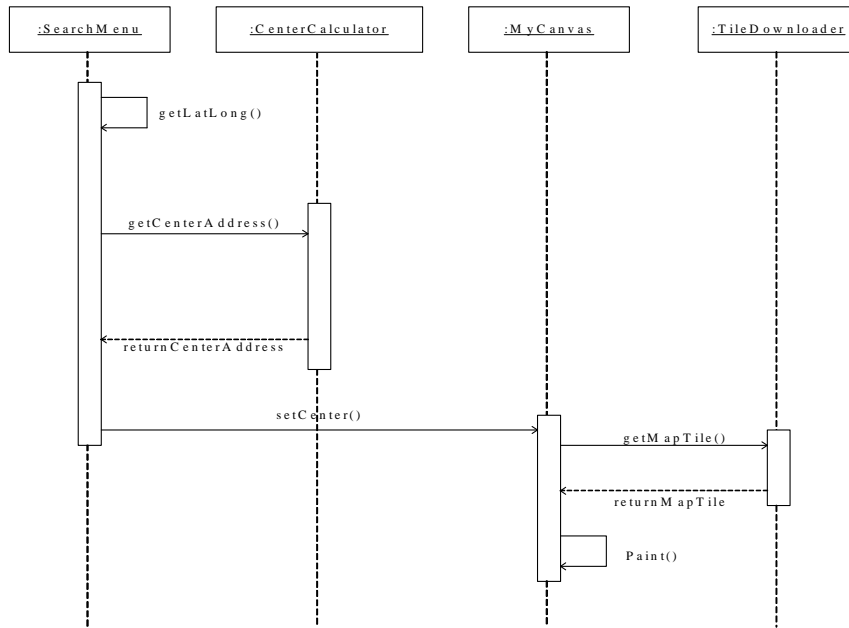


Figure 13: Sequence diagram of the process of getting map tiles

In the class `SearchMenu`, there will be a text field so that the user can give the city name for searching. Then in this class with a certain method the program will get the latitude and longitude values of this city. Then it passes these two values to the class `CenterCalculator` and gets in this class the address of the center map tile. Then it passes this address to the class `MyCanvas` through the method `setCenter()`. In the class `MyCanvas`, the addresses of all adjacent visible tiles will be calculated. And they will be sent to the class `TileDownloader`, this class will help to download all of the map tiles.

### 4.5.3 Major classes

The following is the class diagram of the whole major classes in our system



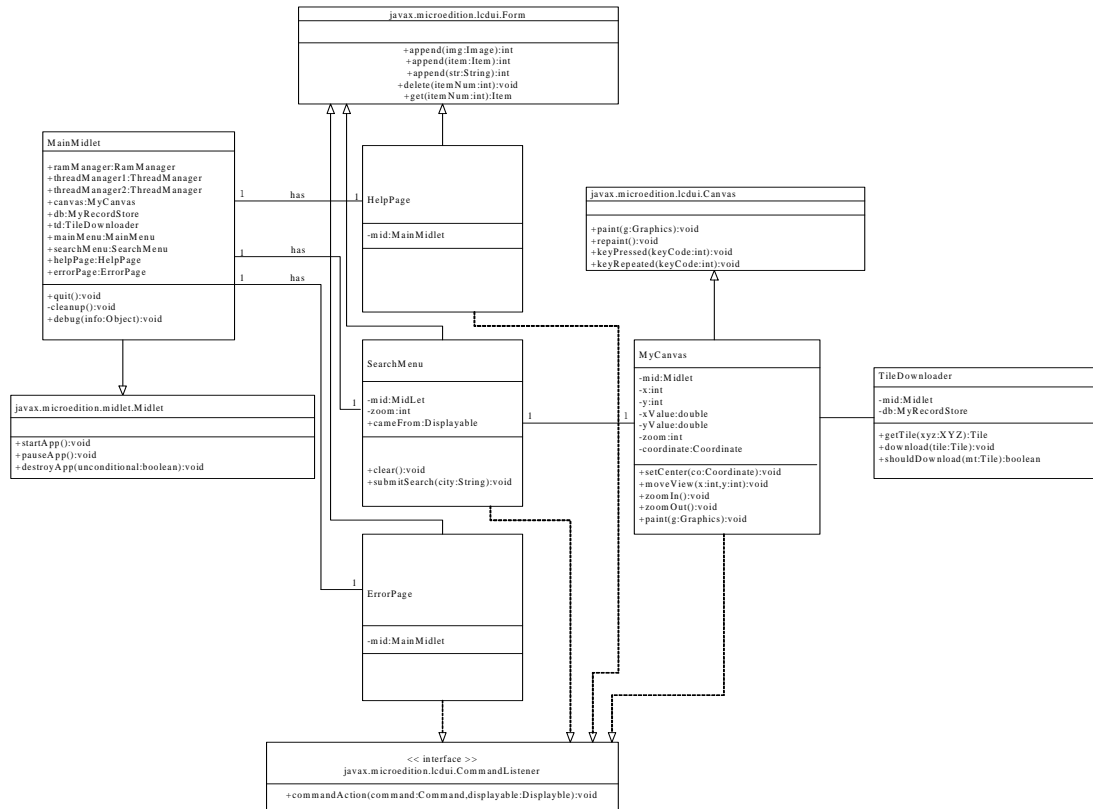


Figure 14 : Class diagram of all major classes of our system

## 5.6 System Implementation

In this Midlet suite are there 15 classes. They can be categorized into

### 1) The assistant classes

- Coordinate.java
- FixedByteArrayOutputStream.java
- LinkedHashMap.java
- MyRecordStore.java
- RamManager.java
- ThreadManager.java
- Tile.java
- XYZ.java

### 2) The major classes

- MainMidlet.java
- MainMenu.java
- HelpPage.java
- SearchMenu.java
- MyCanvas.java
- TileDownloader.java
- ErrorPage.java

## Google maps on cell phones

---

The classes in the first category deal with some assistant jobs, just like their name imply, they manage the RAM distribution, thread distribution, the images of the map storage and so on.

The classes in the second category are in charge of the so called “main jobs”. It includes all user interface classes and all of the classes that are involved in the progress that from the name of the city to get the address of the center map image just like we have discussed already. This paper put emphasis on this part and will explain the important steps of all classes in the order of flowing.

### **MainMidlet**

As we know, in one midlet package is there only one midlet. In this case, it is the class `MainMidlet`. It controls lifecycle of the whole project.

In the constructor of this class, all of the objects that are related to this midlet are initialized

```
canvas = new MyCanvas(this);
db = new MyRecordStore(this);
td = new TileDownloader(this);
mainMenu = new MainMenu(this);
searchMenu = new SearchMenu(this);
helpPage = new HelpPage(this);
errorPage = new ErrorPage(this);
```

In the `startApp()` of this class is there a line of code

```
display.setCurrent(mainMenu);
```

That means at this time, this `mainMenu` object is shown on the screen.

In the method `pauseApp()`, I invoked the method `cleanup()`

```
cleanup();
```

In the method `cleanUp()`, I invoked the methods

```
db.compact();
db.closeRMS();
```

These two lines of code mean if the midlet is in the paused state, its record store will be cleaned up and closed.

In the method `destroyApp()`, the following methods are invoked

```
if (!clean) cleanup();
notifyDestroyed();
```

## Google maps on cell phones

---

That means in its destroyed state, the midlet will clean up and closed its record store and notify the AMS to destroy it.

### **MainMenu**

The class `mainMenu` is subclass of the class `Form`. It implements also the interface `CommandListener`. Here is the major part of the constructor of `mainMenu` :

```
setCommandListener(this);
addCommand(new Command(SEARCH, Command.SCREEN, 1));
addCommand(new Command(HELP, Command.SCREEN, 2));
addCommand(new Command(EXIT, Command.EXIT, 4));
```

That means there will be some choices available on the screen, if we choose one, then the corresponding code in the method `commandAction` in this class will be invoked. For example, if we choose “search”, then these two lines of code will be executed:

```
mid.searchMenu.cameFrom = this;
mid.display.setCurrent(mid.searchMenu);
```

A `searchMenu` object will be shown on the screen.

If we choose “help”, a `HelpPage` will be shown on the display.

If we choose “show error”, an `ErrorPage` will be shown on the display.

### **HelpPage and ErrorPage**

These two classes are similar. They are both subclass of `Form`. The `HelpPage` presents us operation instructions of this program. Obviously it is very easy to implement. The `ErrorPage` shows us the errors and exceptions that appeared. First, in the class `MainMildlet`, we construct a vector and use it to collect all the exceptions

```
final Vector log = new Vector();
log.addElement(message + "\n" + exception + "\n-----\n");
```

Then in the class `MainMenu`, I enumerated the vector and store all the error messages into a `StringBuffer`, and then write all of the messages on the display:

```
StringBuffer sb = new StringBuffer();
for (Enumeration enumeration = mid.log.elements() ;
     enumeration.hasMoreElements() ; ) {
    Object o = enumeration.nextElement();
    sb.append(o.toString());
    sb.append('\n');
    if (sb.length() == 0)
        sb.append("No events logged.\nEverything is OK.");
}
```

Then I construct an `ErrorPage` object and put this error page on the screen

```
ErrorPage error = new ErrorPage(mid,sb);
mid.display.setCurrent(error);
```

In the constructor of the class `ErrorPage` is there such a line of code:

```
append(sb.toString());
```

That means the system will put all of the error messages that has happened until this moment on the screen.

### **SearchMenu**

The `searchMenu` class is also the subclass of `Form`. In this class, we can also get the latitude and longitude according to the name of the city that we gave. Its constructor is so:

```
setCommandListener(this);
addCommand(new Command(BACK, Command.BACK, 1));
addCommand(new Command(OK, Command.ITEM, 1));
append(city = new TextField("City Name", "", 200, TextField.ANY));
```

So a `TextField` will be on the screen. At beginning it is blank, what the user writes will then assigned to the variable `city`. And there also are several choices available. If we choose “OK” from them

```
String stringCity = city.getString();
submitSearch(stringCity);
```

will be executed. The program invokes the method `submitSearch()` and the parameter will be the city name that the user just gave.

In the method `submitSearch()`, first the latitude and longitude of the city will be found.

First let’s analyze the parameters that show in the address of a google map tile, for example if we search for the city “New York”, the address is

<http://maps.google.com/maps?hl=en&q=new+york>

After the question mark, “hl = en” means that the language that we use is English. “&q=new+york” is quite clear. And I found that if we put the parameter “&output=html” after this address, like

<http://maps.google.com/maps?hl=en&q=new+york&output=html>

We will find the latitude and longitude of corresponding city in the source code (javascript) of this webpage. They just follow the keywords “`map.setCenter(new GLatLng(`”. For the case “New York”, that would be:

## Google maps on cell phones

---

```
map.setCenter(new GLatLng(40.714167, -74.006389), zoom)
```

So in the method `submitSearch()`, first we replace all of the spaces in the name of the city with “+”.

```
String rep = city.replace((char)32, '+');
```

Then we determine the url for this city

```
final String url = "http://maps.google.com/maps?f=q&hl=en&q=" +  
rep + "&output=html";
```

And then we get the contact with the web page using `HttpConnection` in the package `javax.microedition.io`.

```
HttpConnection httpconnection =  
    (HttpConnection)Connector.open(url);  
InputStream in = httpconnection.openInputStream();
```

We use the following code to get the source code (javascript) of the webpage and convert it to a `String s`

```
StringBuffer ss = new StringBuffer();  
for (;;) {  
    int c = in.read();  
    if (c == -1) break;  
    ss.append((char)c);  
}  
String s = ss.toString();
```

The following is quite simple. We should analyze this `String s` and in this string find the location of the keyword „`GLatLng`” and then get the latitude and the longitude value that follow this keyword. The codes are quite much but the idea is quite clear. So I will not give the source code here.

Until now the latitude and longitude of the city are already known. What we should do next is to get the address of the center map image according to the latitude and longitude value. This step is implemented in the class `CenterCalculator`. The method `submitSearch` in the class `SearchMenu` invoked

```
Coordinate coordinate = new Coordinate(0,0);  
coordinate =  
CenterCalculator.getTileCoordinate(fiLatitude, fiLongitude, logV  
alue, zoom);
```

to transfer the latitude and longitude value to the class `CenterCalculator`.

### CenterCalculator

The coordinate system that google maps is using in its javascript is mercator system. When we know the latitude and longitude of certain city, we can calculate the x and y value of this city in mercator coordinate system. We already knew that each map tile is a 256\*256 pixels square. If we divide x and y value by 256, we can get the address of the center image. You can find the javascript implementation of this process here:

[JMapJS].

And I found the java variation of this algorithm in this page: [LaLoToTile].

The difference is this java code use `Math.log()` which is not available in the MIDP 2.0. So I use a PHP proxy to compute the logarithm and use again `HttpConnection` to get the value.

The class `CenterCalculator` gets the address of center image for certain city and give this value back to the class `searchMenu`. The class `searchMenu` then invokes the following two methods:

```
mid.canvas.setCenter(coordinate, zoom);
mid.display.setCurrent(mid.canvas);
```

Now the object `MyCanvas` is shown on the screen and which map tiles should be displayed are determined by the method `setCenter()`.

### MyCanvas

The algorithm to navigate the map and zoom in, zoom out is implemented in this class. Several methods are involved in this process including `paint()`, `moveView()`, `zoomIn()`, `zoomOut()`. The most important method is `paint()`. The following are major codes of `paint()`.

First, we get the x and y value for the center image:

```
this.xValue = coordinate.x;
this.yValue = coordinate.y;
```

Then we magnify this value

```
this.x = (((int)xValue << 7) + (w/2) ) << zoom;
this.y = (((int)yValue << 7) + (h/2) ) << zoom;
```

This is because if I want to navigate the map, I can not simply change the x and y value of the image. I mean if I replace x with x+1, then the whole image will be replaced. The canvas of the cell phone will be repainted with a totally different map tile. Such change is too distinct. I magnify x and y value so that the change will be not so great. The way that I magnify the x and y value I borrowed it from `Gmapviewer`.

[GMapViewer].

Then in the method `paint()`:

## Google maps on cell phones

---

```
final int sx = (x >> zoom) - (w / 2);
```

This sx means the screen x position.

```
final int xpos = sx >> 7;
```

The xpos represents the x value of center iamge.

```
final int xoff = (2*sx) % 256;
```

The xoff is the position from which that I draw the image. For example, when xoff = 8, I will draw the image from the 8<sup>th</sup> pixel. The whole image is 256 pixels. So the weight of the image that is shown on the screen will be 248. That is useful when I navigate the map. “% 256 “means that this value should be fewer than 256. Because if it is larger than 256, we can totally repaint the canvas with the image whose x value is x+1.

```
final int tw = 2;  
final int th = 3;
```

Because the weight of the cell phone’s screen is 240, the weight of the image is 256, the height of the screen is 289, the height of the image is 256, so the maximal tiles that can show on the screen at the same time is  $2*3 = 6$ .

```
XYZ[] visible = new XYZ[tw * th];  
int nVisible = 0;  
for (int ty = 0; ty < th; ty++) {  
    for (int tx = 0; tx < tw; tx++) {  
        XYZ xyz = new XYZ(xpos + tx, ypos + ty, zoom);  
        if (computeTileVisiblePixels(xyz, sx, sy, w, h) > 0) {  
            visible[nVisible++] = xyz;  
        }  
    }  
}
```

I create an XYZ array and store all XYZ of the visible tiles into it. Whether the tile is visible we use the method `computeTileVisiblePixels(xyz, sx, sy, w, h)`. Then in a for circle, I first get all of the map tiles

```
Tile mt = mid.td.getTile(xyz);
```

And then draw all the visible tiles

## Google maps on cell phones

---

```
int tx = (xyz.x * 256) - ((xpos * 256) + xoff);
int ty = (xyz.y * 256) - ((ypos * 256) + yoff);
g.drawImage(image, tx, ty, Graphics.TOP|Graphics.LEFT);
```

Also worth noting is the method `moveView(int movx, int movy)` implement the navigation of the map. In this method, we change the value of `x` and `y`, so that `sx` and `sy` will be changed

```
int speed = 8;
    movx *= speed;
    movy *= speed;
    x = x + (movx << zoom);
    y = y + (movy << zoom);
```

The `xoff` and `yoff` will be a multiple of 8, but between 0 and 256. For example if we move to right 1 step, the `xoff` will be +8, and it seems that the image move to left 8 pixels. We implement move in this way.

### **TileDownloader**

We already learned that we use method `getTile()` in the class `MyCanvas` to get the map tiles:

```
Tile mt = mid.td.getTile(xyz);
```

This method is from the class `TileDownloader`. In this method we tried to find the corresponding map tiles on the record store. If we could not find it, we create a new map tile and put it on the store:

```
mapTile1 = (Tile)db.get(xyz);
    if (mapTile1 == null) {
        mapTile1 = new Tile(xyz);
        db.put(mapTile1);
    }
```

Then we invoke the method,

```
download(mapTile);
```

In the method `download()`, we get contact with the web using class `HttpConnection`

```
String url = "http://mt.google.com/mt?n=404&v=w2.10&x=" + x +
            "&y=" + y + "&zoom=" + zoom ;
ContentConnection conn = null;
InputStream in = null;
```



## Google maps on cell phones

---

```
conn = (ContentConnection)Connector.open(url);  
in = conn.openInputStream();
```

And use the method `createImage()` in the class `Image` to get the map image from certain page:

```
Image imageSource = Image.createImage(in);  
mapTile.image = imageSource;
```

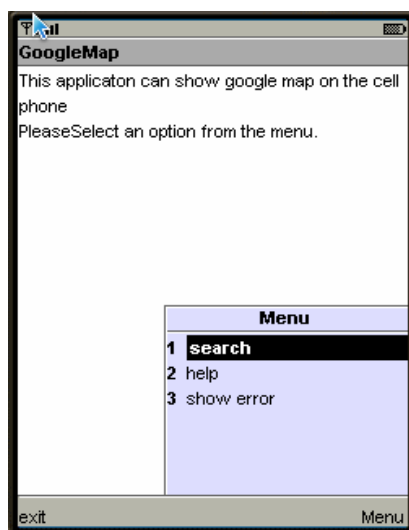
### 5.7 About the assistant classes

Here is the brief description of the functions of all assistant classes. For detail refer to the documentation of my midlet suite.

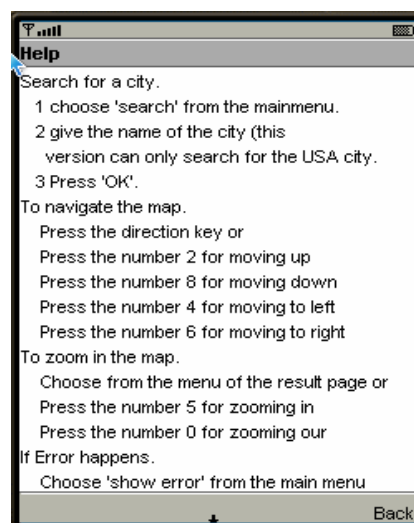
- .Coordinate.java                   The coordinate of certain place in google maps
- .Dialog.java                       All the dialog that appears between two screen objects.
- .FixedByteArrayOutputStream.java   Manage the bytes of Map Tile
- . LinkedHashMap.java               Save the Map Tile
- . MyRecordStore.java               RecordStore of this midlet suite
- . RamManager.java                  Manag the distribution of limited resource
- . ThreadManager.java               Manage the distribution of threads
- . Tile.java                         Represent the Tile of Map
- . XYZ.java                         The group of x,y,z value of a Map Tile

### 5.8 Test the program

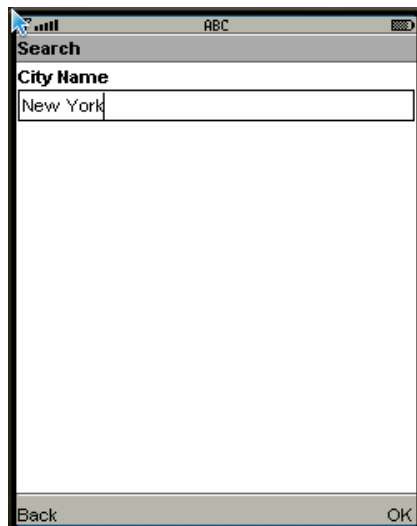
The easiest way that we test this software is: Using user requirements as test cases and check the system functionalities against each requirement to see if they are successful or not. The following screen shots are the result that I search for the city “New York” on the emulator.



1 MainMenu



2 HelpPage



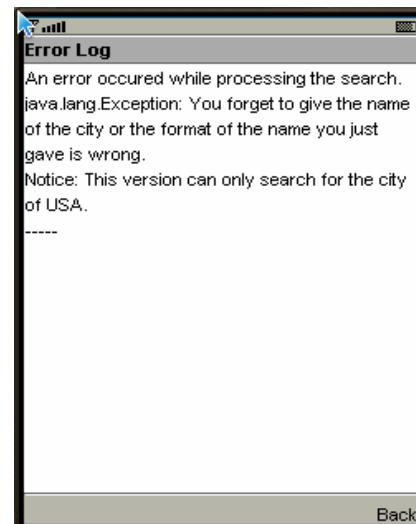
3 SearchMenu



4 MyCanvas (Map)



5 MyCanvas (Menu)



6 ErrorPag

Figure 15 : Screen shots on the emulator

From above screen shots we can see that all supposed requirements are properly fulfilled on emulator. I test my program also on actual device NOKIA 6280. It also works very well just like on the emulator.

## 5.9 Conclusion

### 5.9.1 Difficulties

The whole program took me nearly one and a half month. The reason that I took so much time is: during the implementation, I have encountered the following difficulties:

- 1) Knowledge deficiency in PHP

## Google maps on cell phones

---

At first, I tried to analyze the whole system of the GmapViewer. It uses PHP to solve many tasks like find the latitude and longitude of certain location, convert it to mercator coordinate system, and get the map tile from the google maps server. But I have too few knowledge about PHP. It failed again and again because of unknown reasons. For example, it took me 2 days long to know that the server that I use to put my PHP file does not allow that I open other url address.

### 2) Finding the 3 major algorithm

The above 3 algorithms seem easy but each took me a lot of time. For example, on the web site of google maps, I gave the name of the city and get a map of it. But from the source code of the map page I can not find the latitude and longitude value. So I tried to found another web site to compute these two values according to certain city name. It turned out too complicated. Occasionally I found if I put “&output=html” this parameter to the address, these two values are included in the source code. Thus I solved this problem.

### 3) Problems with debugging

When the whole system already worked very well on the WTK emulator, I tried to run it on actual device. But it did not work. I could not know exactly why. Finally I found that there was a choice on the NOKIA emulator which is called “diagnostic”. I run this “diagnostic” and found the reason that the system did not work. It took too much RAM. The process of rendering my program is tedious. This is because I did not exactly know which step took so much RAM. I checked step by step and finally I found that when I analyzed the String that I got from the javascript soured code to get the latitude and longitude values, I used always the class String.

```
String s;
for (;;) {
    int c = in.read();
    if (c == -1) break;
    s = s + (char)c;           // Using the class String
}
```

If I use the class StringBuffer,

```
StringBuffer ss = new StringBuffer();
for (;;) {
    int c = in.read();
    if (c == -1) break;
    ss.append((char)c);      //Using the class StringBuffer
}
String s = ss.toString();
```

this RAM problem will not happen. After I solve the problem, I found the reason on

## Google maps on cell phones

---

google: Every time the method is invoked,

```
s = s + (char)c;
```

I construct a new String object and that will take too much RAM.

### 4) Unsupported functions in API

As mentioned before, J2ME is a subset of J2SE API. This means some of the classes will be left out. For example in the class `java.lang.Math` is there no method `Math.log()`. I must implement it by myself.

### 5.9.2 Extension Suggestion

Many other functions can also be implemented in the future version. For example

1) We can search for a certain location in a certain city. Just like in google maps. We can search for “pizza” in the city “New York”. The server will give us a list of addresses. We can choose one and show it in our map. Actually the job is just to find the latitude and longitude values for those addresses. `GMapView` already implemented this task. It finds the values through the PHP proxy. If I had time, I can also find those values using J2ME itself.

2) The satellite image of map can also be shown on the screen of cell phones. Only the address systems are changed. This is an example of a satellite map tile address:

```
http://kh1.google.com/kh?n=404&v=6&t=tqtsr
```

These are the addresses of all adjacent tiles:

```
http://kh2.google.com/kh?n=404&v=6&t=tqstq
```

```
http://kh0.google.com/kh?n=404&v=6&t=tqtrs
```

```
http://kh1.google.com/kh?n=404&v=6&t=tqtsr
```

```
http://kh3.google.com/kh?n=404&v=6&t=tqtrt
```

```
http://kh3.google.com/kh?n=404&v=6&t=tqtrt
```

```
http://kh1.google.com/kh?n=404&v=6&t=tqsqt
```

```
http://kh0.google.com/kh?n=404&v=6&t=tqtsq
```

You can find the difference is after „t=“. If we can find the relationship between them, we can show the whole satellite map on the screen.

### 6 Conclusion

Today cell phones are not only used as traditional “telephone”. Many improvements are developed or are being worked. For example

- Translation function will be useful.
- With the development of wireless and 3-G technology, many new entertainment and communication services are now becoming available.
- Many phones already have speech recognition in a form of voice dialing.
- With time, image scanning may develop into full 3D modeling.
- Many Operators try to develop television on cell phones.

Due to these future prospects, we need a good program language for mobile phones. It proved that not only for developers but also for mobile phone manufacturers is J2ME a good choice. Unlike software running on PCs, the mobile phone industry does not use one operating system or platform. That means that the developers have had to know every phone platform before writing code. J2ME, like its brothers, promised software developers the hope of achieving device independence for applications running on mobile phones. It promised to lower development costs by offering the capability of “write once, run anywhere“. It also simplified application creation by providing a platform that was already familiar to millions of Java programmers. To mobile phone manufacturers, J2ME offered a development community that could deliver new functionality for their devices.

It seems that J2ME landscape is more complicated than that of J2SE, but the complexity is due to the underlying diversity of the devices that J2ME must support. Actually, I have found that it takes little effort to learn my way around a particular configuration and profile.

In this thesis I tried to provide a careful study of J2ME. This is an interesting program language with very good future. I think the best way to learn to program with J2ME is to analyze the source codes that other people already have developed and try to rewrite the same program from very beginning.

### A References

- [JCP] Java Community Process Official Web Site  
<http://www.jcp.com>
- [MIDP2.0] What Is New In MIDP 2.0  
<http://developers.sun.com/techttopics/mobility/midp/articles/midp20/>
- [JSR] List of JSRs by JCP Technology  
<http://jcp.org/en/jsr/tech?listBy=1&listByType=platform>
- [SVGMobile] Mobile SVG Profiles  
<http://www.w3.org/TR/SVGMobile/>
- [JSR226] Scalable 2D Vector Graphics API for J2ME  
<http://www.jcp.org/en/jsr/detail?id=226>
- [RIJSR226] RI Binary for JSR 226 API for J2ME  
<http://www.forum.nokia.com/info/sw.nokia.com/id/5305ba6b-e943-42cb-8bff-83d5960a9df4.html>
- [JSR226Nokia] Learning Program from Sun about JSR226  
<http://developers.sun.com/learning/javaoneonline/2005/mobility/TS-7105.pdf>
- [JSDK] Java SE 6 Beta  
<http://java.sun.com/javase/downloads/index.html>
- [WTK] The Sun Java Wireless Toolkit 2.3 Beta  
<http://java.sun.com/products/sjwtoolkit/>
- [ECLIPSE] Eclipse SDK 3.1.2  
<http://www.eclipse.org/downloads/>
- [EclipseME] EclipseME 1.5.0  
[http://sourceforge.net/project/showfiles.php?group\\_id=86829](http://sourceforge.net/project/showfiles.php?group_id=86829)
- [EclipseMEIn] Installing EclipseMe itself  
<http://eclipseme.org/docs/installEclipseME.html>
- [MidCreation] Create Java Midlet  
<http://eclipseme.org/docs/createMidlet.html>
- [GoogleMaps] Google Maps  
<http://maps.google.com/>
- [JMapJS] Goolge Map Javascript Source Code  
<http://maps.google.com/mapfiles/maps.31.js>
- [LaLoToTile] Java Source Code for Converting Latitude and Longitude to Tile  
[http://www.mapki.com/index.php?title=Lat/Lon\\_To\\_Tile](http://www.mapki.com/index.php?title=Lat/Lon_To_Tile)
- [GMapViewer] A Java Midlet to Show Google Maps Which Was Developed by Sreid.  
<http://www.sreid.org/GMapViewer/>
- [Top2002] Topley, K.  
J2ME in a Nutshell

## Reference

---

- O' Reilly Edition March 2002
- [Wang2001] Wang, S  
Introduction to Java Midlet Development  
25, 08, 2001
- [Wells2004] Wells, M. J.  
J2ME. Game. Programming  
2004.
- [MIDPAPI] Sun Microsystems  
MIDP APIs for Wireless Applications  
February 2001
- [Pro2002] Provost W. W  
Wireless Programming Using J2ME and MIDP  
Revision 1.0. 2002
- [WTKGui] Sun Microsystems  
User' s Guide J2ME Wireless Toolkit  
October 2004
- [Day2001] Day, B.  
Developing Wireless Applications using the J2ME  
2001

**B Figure list**

Figure 1: Three Editions of Java.....5  
Figure 2: JSR 226 AP..... 10  
Figure 3: Standard Development Process of Java Midlet ..... 11  
Figure 4: WTK Properly integrated into EclipseME ..... 13  
Figure 5: Hello.jar on Cell Phone Emulator ..... 14  
Figure 6: Example of JAD File..... 15  
Figure 7: LCDUI Example (TextBox) .....20  
Figure 8 : Canvas Example .....22  
Figure 9: Example using RecordStore .....25  
Figure 10: Example using HttpURLConnection.....27  
Figure 11 : The hierarchy architecture of the package javax.microedition.io .....27  
Figure 12 : Flow diagram of the user interface system.....31  
Figure 13: Sequence diagram of the process of getting map tiles .....32  
Figure 14 : Class diagram of all major classes of our system.....33  
Figure 15 : Screen shots on the emulator.....42



### C Content of CD-ROM

Environment	Software that are needed for developing Midlet
Environment/JSE	JDK-6-beta for windows
Environment/ECLIPSE	ECLIPSE3-3.1.2 for windows
Environment/ECLIPSEME	ECLIPSEME-1.5.0
Environment/WTK	Sun-wireless-toolkit-2_3
Environment/Nokia	Nokia carbide1.0.1( development suite from nokia) S40_DP20_SDK ( SDK and Emulator )
Environment/SIEMENS	SDK M55 Emulator
Environment/SVG	JSR-226-specification JSR-226-implementation ( from nokia ) Tinylinemidpsdk Tinylinemidp.jar Tinylinens60.jar
Files	The study material that I found in the internet
Files/Books	The reference books
Files/Papers	The papers
Files/web	The web address of internet resource
Myfiles	All the files that i write myself
Myfiles/powerpoint	Powerpoint
Myfiles/thesis	Thesis in word and pdf
Myfiles/GoogleMaps src	All the source code of my midlet suite
Myfiles/GoogleMaps doc	The HTML document of all classes
Myfiles/GoogleMaps	The Google Maps midlet suite
Myfiles/GoogleMaps deploy	JAD and JAR file
Source Code	The source code I found on the web
Source Code/App	A phone book midlet
Source Code/Game	15 midlet games
Source Code/GoogleMap	The source code of GoogleMap

## Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. All information derived from the published or unpublished work of others has been acknowledged in the text and the list of references.

Signature: .....

Student's full name: .....

Date: .....

Faculty/ School: .....