Network

Mobile fundamental and programming

Third class

The main steps to build first application in J2ME

م.م علاء الدين عباس عبد الحسن

Sunday 6/10/2013

The main library in J2ME are:

javax.microedition.lcdui: for user interface ,windows ,bottoms and textboxs.
javax.microedition.rms :storage methods on the main storage unit in the mobile phone.
javax.microedition.midlet : MIDP and application environment.
javax.microedition.io: for communication framework
java.io :for input,output tool.
java.lang: for variables type.

java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.Integer
java.lang.Long
java.lang.Short

Build the first program

In java and C we see that execution steps are begin from the MAIN() function , where in J2ME is little deferent and well begin from the STARTAPP() function for simple reason, that's it when we write a program in java or c its enough to write the main function to say to the compiler to begin from here .where in J2ME we extends class form MIDlet which founded in javax.microedition.midlet and there are three abstract functions (destroyApp , pauseApp , startApp) and one of programming

fundamental that the abstract function in extended class should be completely written . for this reasons the main program well be extended from the class MIDlet and well be start form STARTAPP() function .

The MIDP Model and Lifecycle.

The MIDlet forms the application framework that executes on CLDC devices under the Mobile Information Device Profile (MIDP). Every application must extend the MIDlet class found in the javax.microedition.midlet package. The application management software (AMS) manages the MIDlet itself. The AMS is a part of the device's operating environment and guides the MIDlet through its various states during the execution process. Unlike desktop or server applications, MIDlets should not have a public static void main() method. MIDlets are initialized when the AMS provides the initial class needed by CLDC to start the MIDlet. The AMS then guides the MIDlet through its various changes of state. We shall look at these states next.

MIDlet States

Once a MIDlet has been instantiated, it resides in one of three possible states. A state is designed to ensure that the behavior of an application is consistent with the expectations of the end-users and device manufacturer.

Initialization of the application should be short; it should be possible to put an application in a non-active state; and it should also be

possible to destroy an application at any time. Therefore, three valid MIDlet states exist:

PAUSED

The MIDlet has been initialized, but is in a dormant state. This state is entered in one of four ways:

• after the MIDlet has been instantiated by the AMS invoking its constructor; if an exception occurs, the destroy state is entered

• from the ACTIVE state, if the pauseApp() method is called by the AMS

• from the ACTIVE state, if the startApp() method has been called but an exception has been thrown

• from the ACTIVE state, if the notifyPaused() method has been invoked and successfully returned.

When a well-written MIDlet is paused, it should generally release any shared resources.

ACTIVE

The MIDlet is functioning normally. This state is entered after the AMS has called the startApp() method. The startApp() method can be called on more than one occasion during the MIDlet lifecycle.

DESTROYED

The MIDlet has released all resources and terminated. This state, which can only be entered once, is entered for the following two reasons:

• the destroyApp(boolean unconditional) method has been called by the AMS and returned successfully; if the unconditional argument is false a MIDletStateChangedException may be thrown and the MIDlet will not move to the DESTROYED state; the implementation of the destroyApp() method should release all resources and terminate any running threads

• when the notifyDestroyed() method successfully returns; the application should release all resources and terminate any running threads prior to calling notifyDestroyed().

MIDlet Lifecycle Methods

The javax.microedition.midlet.MIDlet abstract class defines three lifecycle methods:

• pauseApp() – this method is called by the AMS to indicate to the MIDlet that it should enter the PAUSED state, releasing all shared resources and becoming passive

• startApp() – this method is invoked by the AMS to signal to the MIDlet that it has moved from the PAUSED to the ACTIVE state. The application should acquire any resources it requires to run and then set the current display

• destroyApp() – this method is called by the AMS to indicate to the MIDlet that it should enter the DESTROYED state; all persistent and state data should be saved and all resources that have been acquired during its lifecycle should be released at this point;

generally, a well-written MIDlet will start up in the state it was in prior to being shut down.

The Lifecycle Model

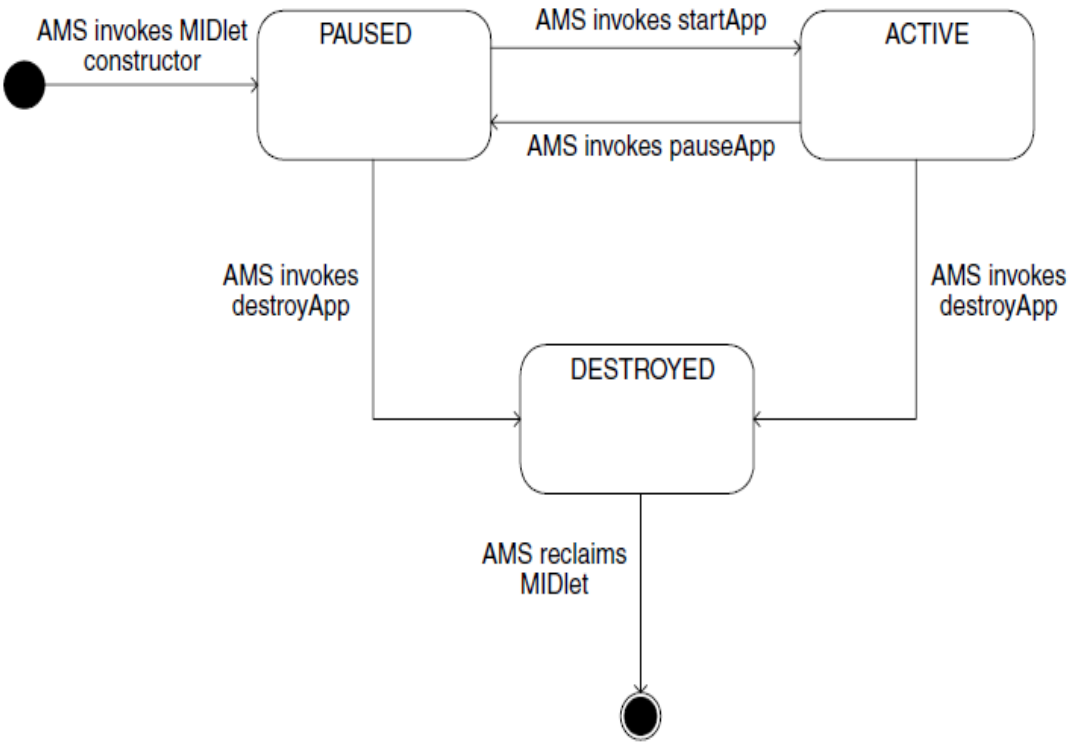
The various states of the MIDlet the Figure show how the AMS and the MIDlet interface combine to form the lifecycle of the MIDlet:

1. The AMS creates a new instance of aMIDlet. The MIDlet's constructor is called with no argument and the application is put into the PAUSED state. If any exception is thrown during this phase then the application is put into the DESTROYED state.
2. The AMS calls startApp() to move the MIDlet into the ACTIVE state. The MIDlet itself will at this point acquire any resources it needs and begin executing.
3. Once the application is running, the MIDlet can move to two other states:

• the MIDlet can be put into the PAUSED state by a call from the AMS to the pauseApp() method The MIDlet will cease to be in the ACTIVE state and choose to release some of the resources it currently holds. If the programmer requires the MIDlet to pause, then the MIDlet should first release shared resources (possibly stopping any running threads) and then call the notifyPaused() method.

• the MIDlet can move to the DESTROYED state The user or the AMS decides that the application no longer needs to be running. Game play may be finished, for example, or the AMS may have

decided that a process of a higher priority needs to claim the resources being used by the MIDlet.



The first program

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HelloMidlet extends MIDlet
{
private Display display;
public void HelloMidlet(){}
public void startApp(){
display =Display.getDisplay(this);
```

```
System.out.println ("Hello world ");
display.setCurrent(form);
}
public void pauseApp(){ }
public void destroyApp(boolean unconditional){}
}
```