Creating applications for multiple Nokia Java ME platforms

This article explains how to write code that will allow you to target multiple Nokia Java ME platforms in the same codebase, using class *inheritance* and *abstraction*.

15 Sep 2013

Introduction

When targeting multiple Nokia Java ME platforms we need to tailor the user experience for each platform, because they differ in capabilities, features, and user interface paradigms.

This article shows how to to develop applications that target multiple Nokia Java ME platforms while keeping code in only one project. The technique used is based on `class` *inheritance* and *abstraction*. We will focus on gathering as much code as we can in only a single *parent* class and leaving child classes handle the uniqueness of each platform.

To help demonstrate and better understand this article we will create a simple application that can post tweets on Twitter consisting of a single `Form`.

Series 40 Touch and Type

Series 40 Full Touch

Nokia Asha Software Platform 1.0

MainForm

In general, depending on the application we develop, there will be many reasons to differentiate our code according to different platforms. In our case we only need to differentiate the way the commands show up on the screen. The *parent* `MainForm` will look like this.

```
import javax.microedition.io.ConnectionNotFoundException;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextField;


public abstract class MainForm extends Form
{
 TextField tweetText;


 public MainForm(String title)
 {
  super(title);
```


(C) Copyright Nokia 2014. All rights reserved.

```
   tweetText=new TextField("Tweet", "", 140, TextField.ANY);
   append(tweetText);

   setCommands();
  }


  protected abstract void setCommands();


  protected void onPost()
  {
   String tweet="https://twitter.com/intent/tweet?text="+tweetText.getString();

   try
   {
    MainMidlet.midlet.platformRequest(tweet);
   }
   catch (ConnectionNotFoundException e)
   {
    e.printStackTrace();
   }
  }

  protected void onAbout()
  {

  }

  protected void onExit()
  {
   MainMidlet.midlet.notifyDestroyed();
  }

 }
```

`MainForm`'s UI consists of a single `TextField` to hold the tweet. When the user is happy with his tweet he will have to activate a `postCommand` to post it on Twitter. The tweet will be posted by using the `Midlet.platformRequest` passing a URL.

Now, we only made room for changes through the `abstract setCommands` method. Also let's say that we target the latest Nokia platforms: *Touch&Type*, *Full Touch* and the *New Asha* phones.

Touch&Type implementation

On touch and type we have to define an "Exit" and "About" command, as they are required by the style guidelines. We also require a "Post" command. Note that the titles of the commands cannot be too long on Touch&Type devices so our `postCommand`'s text will be just "Post".

(C) Copyright Nokia 2014. All rights reserved.

Series 40 Touch and Type

The implementation is given below:

```
package implementations;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;

import MainForm;

public class MainForm_TouchType extends MainForm implements CommandListener
{
 Command postCommand;
 Command aboutCommand;
 Command exitCommand;


 public MainForm_TouchType(String title)
 {
  super(title);
 }

 protected void setCommands()
 {
  postCommand=new Command("Post", Command.OK, 0);
  addCommand(postCommand);

  aboutCommand=new Command("About", Command.OK, 1);
  addCommand(aboutCommand);

  exitCommand=new Command("Exit", Command.EXIT, 0);
  addCommand(exitCommand);
```


(C) Copyright Nokia 2014. All rights reserved.

```
  setCommandListener(this);
 }

 public void commandAction(Command c, Displayable d)
 {
  if (c==postCommand)
   onPost();
  else if (c==aboutCommand)
   onAbout();
  else if (c==exitCommand)
   onExit();
 }

}
```

Full Touch implementation

For Series 40 Full Touch the `postCommand` is implemented as an `IconCommand`, a beautiful UI feature introduced in the platform adding great value to the user experience. Adding the `exitCommand` will display the back button at the bottom-right of the screen and the `aboutCommand` will be hidden in the upper-left menu button.

Series 40 Full Touch

The implementation is shown below:

```
package implementations;
```

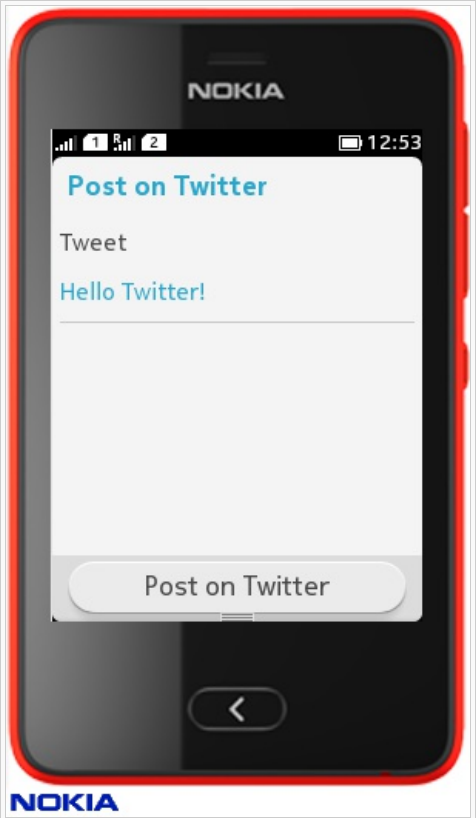

(C) Copyright Nokia 2014. All rights reserved.

```
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Image;

import com.nokia.mid.ui.IconCommand;

import MainForm;

public class MainForm_FullTouch extends MainForm implements CommandListener
{
 IconCommand postCommand;
 Command aboutCommand;
 Command exitCommand;


 public MainForm_FullTouch(String title)
 {
  super(title);
 }

 protected void setCommands()
 {
  try
  {
   postCommand=new IconCommand("Post", Image.createImage("/images/twitter.png"), null,
Command.OK, 0);
   this.addCommand(postCommand);
  }
  catch (Exception ex)
  {
   ex.printStackTrace();
  }

  aboutCommand=new Command("About", Command.OK, 1);
  addCommand(aboutCommand);

  exitCommand=new Command("Exit", Command.EXIT, 0);
  addCommand(exitCommand);

  setCommandListener(this);
 }

 public void commandAction(Command c, Displayable d)
 {
  if (c==postCommand)
   onPost();
  else if (c==aboutCommand)
   onAbout();
  else if (c==exitCommand)
   onExit();
 }

}
```


(C) Copyright Nokia 2014. All rights reserved.

New Asha implementation

In the Nokia Asha Software platform the most important `command` (the one with the highest priority) is displayed at the bottom covering screen's total width. This means that we have more space for a more descriptive command title.



Nokia Asha Software Platform 1.0

Our `exitCommand` will be bound to the single hardware back button of the device. Also our `aboutCommand`, as a secondary command, will be hidden and showed up only when the user swipes up from the bottom of the screen. The implementation is given below:

```
package implementations;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;

import MainForm;

public class MainForm_NewAsha extends MainForm implements CommandListener
{
 Command postCommand;
 Command aboutCommand;
 Command exitCommand;


 public MainForm_NewAsha(String title)
 {
  super(title);
 }

 protected void setCommands()
 {
  postCommand=new Command("Post on Twitter", Command.OK, 0);
```


(C) Copyright Nokia 2014. All rights reserved.

```
  addCommand(postCommand);

  aboutCommand=new Command("About", Command.OK, 1);
  addCommand(aboutCommand);

  exitCommand=new Command("Exit", Command.EXIT, 0);
  addCommand(exitCommand);

  setCommandListener(this);
 }

 public void commandAction(Command c, Displayable d)
 {
  if (c==postCommand)
   onPost();
  else if (c==aboutCommand)
   onAbout();
  else if (c==exitCommand)
   onExit();
 }

}
```

FormFactory

Now that we have differentiated our code and covered all the targeting platforms we will create the `FormFactory` class to allow our application to instantiate the appropriate implementation according to the device it is running on. We can get the platform by using the code found in this article.

```
import implementations.MainForm_FullTouch;
import implementations.MainForm_NewAsha;
import implementations.MainForm_TouchType;


public class FormFactory
{
 public static MainForm createMainForm(String title)
 {
  if (ApplicationGlobal.currentPlatform==ApplicationGlobal.TouchType)
   return new MainForm_TouchType(title);
  else if (ApplicationGlobal.currentPlatform==ApplicationGlobal.FullTouch)
   return new MainForm_FullTouch(title);
  else if (ApplicationGlobal.currentPlatform==ApplicationGlobal.NewAsha)
   return new MainForm_NewAsha(title);
  else
   return new MainForm_TouchType(title);
 }
}
```

The `createMainForm` method is responsible of instantiating the appropriate implementation and we will use it like below.

```
MainForm mainForm=FormFactory.createMainForm("Post on Twitter");
Display.getDisplay(this).setCurrent(mainForm);
```


(C) Copyright Nokia 2014. All rights reserved.

Tip: There is a saying in application development about multiple platforms: *Don't be afraid of IF and be afraid of IF*.

What this means is that *if-then-else* is appropriate when the statement can remain simple. If there are too many conditions then our code becomes less versatile and more difficult to maintain.

Related sources

To take it a step further, we would need to set the icon of our application to match each platform's guidelines. But we cannot control that from our code because that information is stored in the application's .jad file. Check this article for a way to do so using Nokia IDE for Java ME (Eclipse): Automatically edit project configuration in Nokia IDE with Eclipse

(C) Copyright Nokia 2014. All rights reserved.