Using the Java ME PureMVC Framework

PureMVC is a free, open source, lightweight framework for creating applications based upon the Model-View-Contoller pattern. It has been ported to many languages including Java and can be implemented into a Java ME application. This article shows you how to use it.

Introduction

05 Jun 2011

The MVC (Model-View-Controller) pattern is an architectural pattern for software engineering. The pattern enables the programmer to separate the UI (presentation layer), application data and business logic making it much easier to maintain code and accommodate changes.

- Model - The Model is a representation of data specifically used in an application.
- View - The UI of an application. It renders the model for display or interaction.
- Controller - Handles events from user interaction and typically invokes changes on the Model.

PureMVC

PureMVC is a free, open source, lightweight framework for creating applications based upon the Model-View-Contoller pattern. It has been ported to many languages including Java and can be implemented into a Java ME application. Visit http://www.puremvc.org to learn about the framework, best practices and to download the latest version. PureMVC is also available for Python.

Using PureMVC with Java ME

PureMVC is used to build the basis of an application in a well-structured manner. This enables the programmer to easily port the application they are building to many devices and platforms. It also makes it easier to maintain, expand and alter the project.

Download the example Login application.

This example can be used as the basis to any mobile Java project. Currently the application uses version 0.2 of the PureMVC Java port. Here is a quick overview of how some of the framework operates with the Login demo. To gather a full understanding of PureMVC visit the website and study the well written documentation. The examples are written in ActionScript 3, but the theory applies to all languages supported.

The PureMVC framework works on 4 singletons.

- The Model
- The View
- The Controller
- The Facade

All applications start with an ApplicationFacade. It is the central point of communication for the framework.

```
private ApplicationFacade facade = ApplicationFacade.getInst();
...
    protected void startApp()throws MIDletStateChangeException
    {
  this.facade.startup(this);
    }
...
```

ApplicationFacade.java

```
package org.puremvc.java.demos.j2me.login;
import org.puremvc.java.patterns.facade.Facade;
import org.puremvc.java.patterns.observer.Notification;
import org.puremvc.java.demos.j2me.login.controller.StartupCommand;
import org.puremvc.java.demos.j2me.login.controller.ProcessLogin;
import org.puremvc.java.demos.j2me.login.LoginExample;
public class ApplicationFacade extends Facade
{
 public static final String STARTUP = "startup";
 public static final String LOGIN = "login";
 public static final String SUBMIT_LOGIN = "submitLogin";
 public static final String LOGIN_SUCCESSFUL = "loginSuccessful";
 public static final String LOGIN_FAIL = "loginFail";
 public static final String MAIN = "main";

 //Startup command notifications
     public static final String PREP_MODEL = "prepModel";
     public static final String PREP_VIEW = "prepView";

 private static ApplicationFacade instance = null;

 public static LoginExample midlet;

 public static ApplicationFacade getInst()
 {
  if(instance == null)
  {
   instance = new ApplicationFacade();
  }

  return instance;
 }

 protected void initializeController()
```


(C) Copyright Nokia 2013. All rights reserved.

```
  {
   super.initializeController();

   registerCommand(STARTUP, StartupCommand.class);
   registerCommand(SUBMIT_LOGIN, ProcessLogin.class);
  }

  public void startup(LoginExample midlet)
  {
   this.midlet = midlet;
   notifyObservers(new Notification(STARTUP, null, null));
  }
 }
```

Communication through the framework works by registering commands, mediators and proxies.

ApplicationFacade.java

```
registerCommand(SUBMIT_LOGIN, ProcessLogin.class);
```

PrepViewCommand.java

```
this.facade.registerMediator(new LoginScreenMediator());
```

PrepModelCommand.java

```
this.facade.registerProxy(new ItemDataProxy());
```

Every UI screen has a mediator attached to it. The mediator interacts with the UI and the framework. Mediators respond to calls by registering a notification interest in a call and handle any calls through the handleNotification method.

LoginScreenMediator.java

```
public String[] listNotificationInterests()
{
 return new String[] {ApplicationFacade.LOGIN, ApplicationFacade.LOGIN_FAIL};
}


public void handleNotification(INotification note)
//Variables can be passed along through INotification note.
{
 if(note.getName().equals(ApplicationFacade.LOGIN))
 {
  ApplicationFacade.midlet.fetchDisplay().setCurrent(getLoginScreen());
 }
 else if(note.getName().equals(ApplicationFacade.LOGIN_FAIL))
 {
  getLoginScreen().loginFail();

 }
}
```

Mediators can also make a call out to the framework by using facade.notifyObservers.

```
...
 this.facade.notifyObservers(new Notification(ApplicationFacade.SUBMIT_LOGIN, details,
 null));
```

NOTE: details is a string array. Any variable can be passed along through a notification

A control listens to calls when it is registered.

```
registerCommand(SUBMIT_LOGIN, ProcessLogin.class);
```

Data can be stored using a Value Object (ItemDataVO.java) and accessed through a proxy(ItemDataProxy.java)

Value objects can be populated and added to a proxy in the following manner.

ProcessLogin.java

```
//Add some ItemDataVO objects using proxy
ItemDataProxy itemProxy = (ItemDataProxy)facade.retrieveProxy(ItemDataProxy.NAME);
for(int i=0; i<url.length;i++)
{
 ItemDataVO item = new ItemDataVO(url[i], data[i]);
 itemProxy.addItem(item);
}
```

Again this is only a brief overview of the PureMVC framework. Visit http://www.puremvc.org and read the documentation to fully understand the use of the framework. Also study the attached Login demo codebase. The Login Demo is free for you to use, alter and build upon. Please help the PureMVC community by contributing any new and better ways you discover in the use of this framework with Java ME. You can also help by providing more demos and tutorials. I hope you enjoy creating better structured and easier to maintain applications utilising the PureMVC framework.

Internal link

- Model-View-Controller application architecture

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.