Programming in Java using the MVC architecture



[**Afzaal Ahmad Zeeshan**](http://www.codeproject.com/script/Membership/View.aspx?mid=10255428), 24 Feb 2015 [CPOL](http://www.codeproject.com/info/cpol10.aspx)

|  |  |
| --- | --- |
| http://cdn.codeproject.com.global.prod.fastly.net/script/Ratings/Images/stars-fill-md.png  http://cdn.codeproject.com.global.prod.fastly.net/script/Ratings/Images/stars-empty-md.png | 4.50 (12 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate this: | [vote 1vote 2vote 3vote 4vote 5](http://www.codeproject.com/Articles/879896/Programming-in-Java-using-the-MVC-architecture) |  | |

This article uses the MVC framework for building Java-based applications for desktop or for enterprise solutions.

* [**Download JavaHelloCollections.rar - 27.2 KB**](http://www.codeproject.com/KB/java/879896/JavaHelloCollections.rar)
* [**Download JavaHelloCollections.zip - 28.6 KB**](http://www.codeproject.com/KB/java/879896/JavaHelloCollections.zip)

Introduction

This article is about MVC framework in Java application development, from desktop applications for basic programs to enterprise solutions written in Java. In this article the basic concepts about MVC framework would b covered. MVC framework is used to separate the data access layer, business logic code and the graphical user interface that has to be defined and designed to let the user interact with the application. This application has three parts,

1. Model - this part of the framework is to store the data of the application, such as databases, text data, files and/or other web resources.
2. View - this is the graphical user interface of the application. That would contain different buttons, text boxes and other controls to let the user interact with the application to complete his projects depending on sort of the software he is using.
3. Controller - the actual back-end code constitutes the controller of the framework. A controller controls the data coming from the users, or going to the user from a model.

This sets up a condition of validation, because the stream of data (coming from or going to the user) is always validated on the controller. That is why it makes the data more consistent by removing any chance of invalid data entry or unauthorized data deletion from the application's data source.

Background of MVC

In this short section I would try to provide a basic overview of the MVC framework. This section is not for Java-specific framework, but for the MVC framework itself. Beginners might be interested in knowing what this MVC framework is, how it interacts, and how it makes the software development efficient. But if you already know the MVC framework then you don't really want to read this section, continue to the development of the application in the next section.

The MVC framework, is generally a term for a software architecture for implementing the user interfaces. You can separate the software's source code into three different layers.

1. Model
2. View
3. Controller

After that you can develop them separately. This way you don't have to scratch your head once your application reaches an enterprise level and your source code looks a bit messy and is unable or debug. In MVC you have all your three layers separate and this way you can control, develop, debug or add features to all of the layers separately.

A very good feature of the MVC framework is that is hides the data access layer from the users. That is, the data access layer or the data is never actually called directly by the user; from the interface. This way, the user has to perform the actions that he is allowed to. This feature allows the developers to create groups or roles of users that are allowed to access the data; such as Admins, Guests etc.

Another good thing about this framework is that it doesn't let the application get so complicated, and all the three segments of the application interfering with each other in a single source code package. So you always know where the problem occurred. Common examples of this would be,

1. When you know that the problem is in the data access layer. For example, in the database you've allowed null entries, so now spam users would be able to leave the data inconsistent and cause a problem of redundancy. You can edit the data access layer, so that you can define the columns to not have null values.
2. When the view is not showing the data correctly. Such as scenarios where binding of the data is not accurate, or the format is not correct like the format of the date and time values. You can simply just edit the View section of your application to make it work again, instead of editing the entire page containing different variables for back-end code and/or the data access code.
3. When the logic is not correctly defined. Sometimes the data is correct, the view is correctly bonded to the resources, but the logic of your application is not correct. This type of problem is hard to figure out, because developer has to think about where code went wrong by giving it a rough try or by debugging the code using the IDE and putting a few breakpoints at the locations.

These are a few of the good points of the MVC framework for developing the software applications. In visual representation, the MVC framework works in this way,

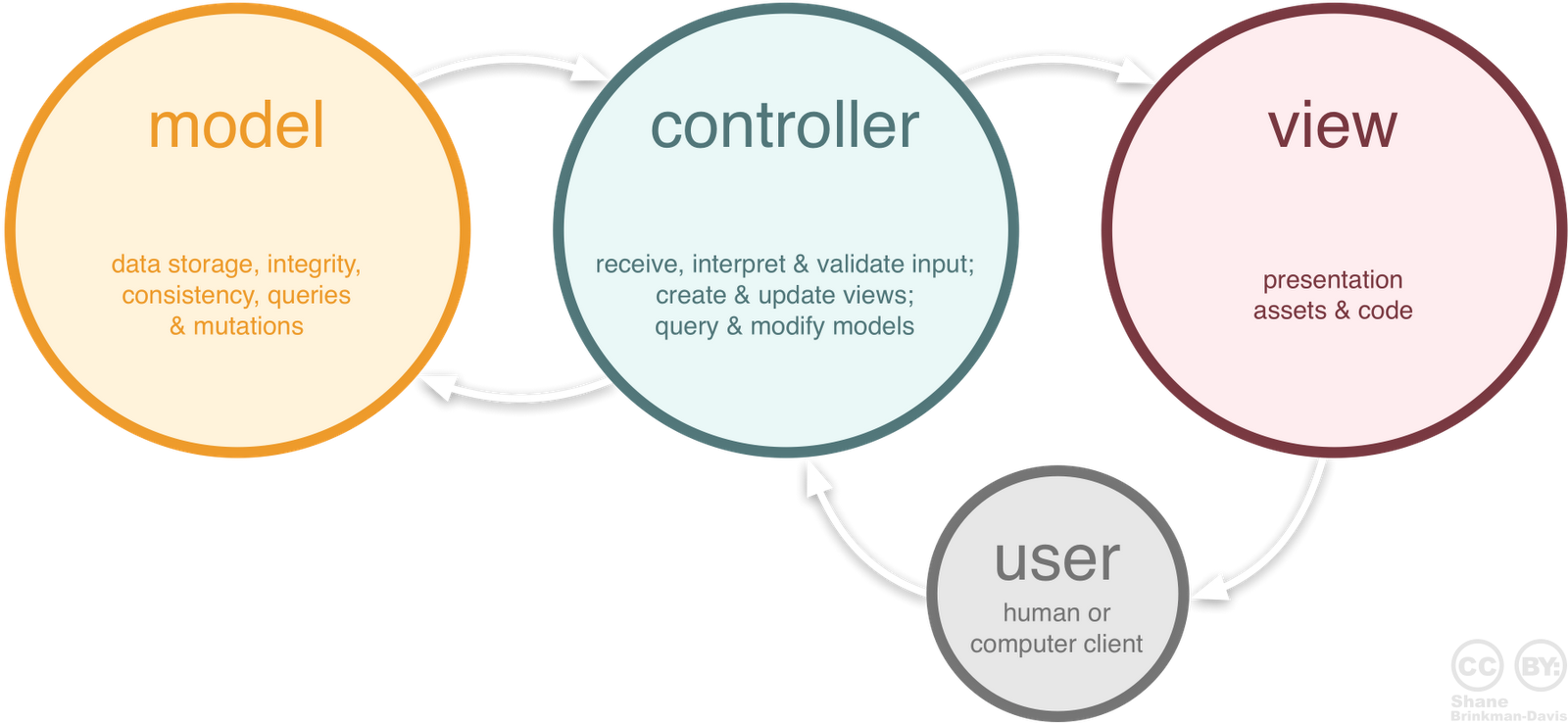


Image clarifies that the user is not able to connect to the data sources himself. Instead he has to interact with the top-level layers to access the data. Such as, the view, he interacts with buttons and the buttons then call the controller to perform an action on the data (*if required*). But the data is not directly accessible by the user. This makes the data source secure from potential users; *anyhow, there is always a threat of potential user*.

I wrote an article about the ASP.NET MVC framework, where I demonstrated the MVC framework before digging deeper into the ASP.NET's MVC framework; just like this one. So, if you're interested on learning more for MVC framework you might be interested in reading that article for one time; just the MVC section not the ASP.NET part. You can read that article [here](http://www.codeproject.com/Articles/871375/Understanding-ASP-NET-MVC-using-real-world-example), and you can also post any queries or questions about that article on it.

Building the application

Now comes the actual part of implementing this framework and building over Java-based application. In the steps coming, I will show you how you can (using this very framework) create applications starting from as simple as Hello world application to as much complex as enterprise level applications using Java.

**Setting the environment - *for Java beginners***

For developing Java applications, you are required to have an environment set up. Environment requires an IDE; for writing the source code and managing the required libraries into one single package, a few popular nowadays are

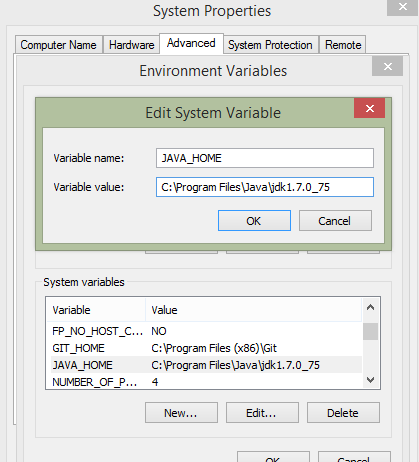
1. [Eclipse for Java](https://projects.eclipse.org/releases/luna) - Eclipse is greatly used Java IDE, it supports other languages too, but since we're talking about Java application so I highlighted only Java.
2. [NetBeans by Oracle](https://netbeans.org/downloads/) - NetBeans is an IDE developed under Oracle license and is mostly used by beginners for the sake that it helps them create a GUI by drag-and-drop feature. *I am also using NetBeans for Java development, for Android development I prefer Eclipse. You can choose on your own as your taste would like.*

and the Java SDK. Now it depends on which level of application you're going to developer, there are multiple type of Java SDK available out there for you and you can choose from them. I am using Java SE SDK and it fulfills my requirements. A list of the Java SDK available for download, and popular, are,

1. [Java SE](http://www.oracle.com/technetwork/java/javase/overview/index.html) - Standard edition. Perfect for small applications, like desktop or other applications that are not much complex.
2. [Java EE](http://www.oracle.com/technetwork/java/javaee/index.html) - Enterprise edition. Should be preferred over Java SE if you're going to develop an enterprise application and the program should be complex.
3. [Java ME](http://www.oracle.com/technetwork/java/javame/index.html) - Millennium edition. It is used to develop mobile applications, such as .jar applications and other projects that are to be run over mobile devices. It should be used because it requires less memory and a small heap size for the application to run, thus suitable for mobile application programming.
4. Then they continue to all the way to TV, Cards and so on and so forth. Learn more [here](http://www.oracle.com/technetwork/java/index.html).

You can download any SDK, the Java language would be similar. So the code in my article would be a good fit you in any condition. You can download and install the SDK on your machine. Once this step has been taken you can continue to the programming.

***The installation of the IDE*** would require a *JAVA\_HOME* system variable to be declared. Simple task; a bit confusing for the beginners. Go to properties of my computer, and inside it advanced system settings, Environment variables. Inside it create a new system variable and name it *JAVA\_HOME* and the value must be the location where the bin folder is located at.



Once this step is done. The IDE would automatically continue setting the required libraries and would install. You can then continue to create a new application and start making an MVC-based Java application.

**Creating the application**

*The source code is relative to my IDE, yours might be different.*

Next step is to create the application. Depending on your IDE, create a new application; a simple Java based application.

*You should take care of one thing*, unlike C# in Java your class name and the file name containing the class must match in order for the programs to run. If they don't Java would complain, saying, the class must exist in its own file. So, if you want to stick to the article, you should name your application's files as I have (I will mention the names of the packages and the files along with the code they contain) or you can create your own project and then edit the names of the classes and/or packages.

When you create the application, in Eclipse it were empty project whereas in NetBeans it created the package (with exact same name of the application) and create a class with a Main function. In my case I named the project to be JavaHelloCollections. If you want to follow the article, name it as it is, otherwise just follow the concept of "*first package in the application*". The class file in this package (depending on your IDE) would be located under the package, open it.

Hide   Copy Code

*/\**

*\* To change this license header, choose License Headers in Project Properties.*

*\* To change this template file, choose Tools | Templates*

*\* and open the template in the editor.*

*\*/*

package javahellocollections;

*/\*\**

*\**

*\* @author AfzaalAhmad*

*\*/*

public class JavaHelloCollections {

*/\*\**

*\* @param args the command line arguments*

*\*/*

public static void main(String[] args) {

*// TODO code application logic here*

}

}

This is the very basic application, that *doesn't do anything at all but compiles and does run*. Up to this stage the application has been created but lacks the MVC pattern of programming. That stage comes after this one. So, to create the business logic of our application we're actually not going to edit this file, we're going to define the MVC pattern of our application and then we're going to use that pattern to make sure that the user interacts with the view, view interacts with the controller and the controller provides with the data from the model after making calls to the model.

Implementing the MVC pattern

As far as now, we've set our environment, created the application that *does run*. Now it is time for us to implement the MVC pattern in our application and make sure that application follows the rules.

In Java we create packages just like we create namespace in C#, C++ or other object-oriented programming languages. Packages allow us to make a collection of similar classes, or classes that are required in the same order or for the same cause. In our application we require to create an MVC pattern, for that we can create three different packages, one for Model, one for View and one for Controller. After that we can add all of our Models, Views and Controllers in these packages and use them in different packages where ever needed.

This way, we can separate all of the three sections from one another, yet we can call them in our sections; by importing the packages. This way, the model would be left to be used by the controller, view should make a call to the controller to get or update the data.

*In this article, I will be using a file example to read and write the data to the file. You can use your own data source; database, web resource etc.*

Since we're going to develop the application first, and then notice the changes by running the application. Then there actually is no requirement to follow, to create the view first then create a controller to render the view and then to create a model to add some data. We can start by creating any section and then merge them into one single application and then run it.

**Creating the Model**

Let us start by creating a model for our application. This model would provide the data to our application. One thing you should consider is, that the model is not just a class to define the structure of the data in our application, like we do in the database applications. The structure has to be defined there, in simple application there is no need. This ambiguity is generally defined and created by the Entity framework on .NET framework, where you define a class for the structure of your database table and then fill a collection of objects of that class from the database. A model is the part which contains your data. It can provide you with the data, and can update the data; even the *controller makes a call to model to update itself* which means that controller sends the data to the model and the model then updates the data on the disk.

First, create a new *package,* remember to create a new package under the same project and not the same package. These would be two different packages for the same project and we will be using them in our application as where required. Name this package, JavaMVCModels. Once done, create a new class file under it. This class would act as our model for application. Name it, HelloWorldModel. Open the file, and alter the file. Our scenario is,

1. Allow the user to load the data from a data source; such as a file.
2. Allow the user to update the data in the file.

Inside the class, write this code.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

*/\**

*\* To change this license header, choose License Headers in Project Properties.*

*\* To change this template file, choose Tools | Templates*

*\* and open the template in the editor.*

*\*/*

package JavaMVCModels;

import java.io.\*;

import java.nio.file.Files;

import java.nio.file.Paths;

*/\*\**

*\**

*\* @author AfzaalAhmad*

*\*/*

public class HelloWorldModel {

public String getData() throws FileNotFoundException, IOException {

if(!(new File("F:\\file.txt").isFile())) {

*// Create -- Make sure file exists -- the file before continuing*

Files.createFile(Paths.get("F:\\file.txt"));

}

String data;

*// We will be using a try-with-resource block*

try (BufferedReader reader = new BufferedReader(

new FileReader("F:\\file.txt"))) {

*// Access the data from the file*

*// Create a new StringBuilder*

StringBuilder string = new StringBuilder();

*// Read line-by-line*

String line = reader.readLine();

string.append("<html>");

*// While there comes a new line, execute this*

while(line != null) {

*// Add these lines to the String builder*

string.append(line);

string.append("<br />");

*// Read the next line*

line = reader.readLine();

}

string.append("</html>");

data = string.toString();

} catch (Exception er) {

*// Since there was an error, you probably want to notify the user*

*// For that error. So return the error.*

data = er.getMessage();

}

*// Return the string read from the file*

return data;

}

public boolean writeData(String data) throws IOException, FileNotFoundException

{

*// Save the data to the File*

try (BufferedWriter writer = new BufferedWriter(

new FileWriter("F:\\file.txt"))) {

*// Write the data to the File*

writer.write(data);

*// Return indicating the data was written*

return true;

} catch (Exception er) {

return false;

}

}

}

Above code can be explained in the following few stages:

1. First of all is the declaration of the package in which this class exists. JavaMVCModels is the package which would contain the class file.
2. The class name and the file name is similar.
3. The class contains two methods, one to extract the data and one to save the data from the user.

The above code makes a use of the File APIs of the Java language. You can find File, BufferedWriter, BufferedReader and other classes in the Java documentations and I do not need to explain them. They're used to store the data in the files, or to extract the data from the files.

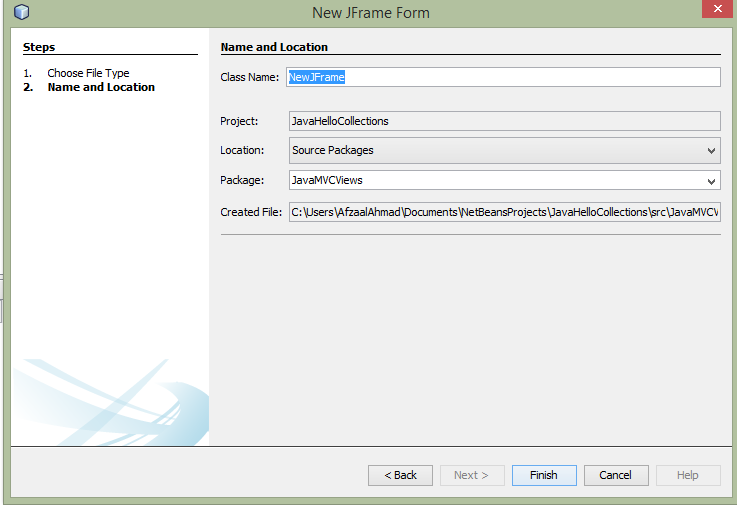
The Java SE 7, you can avail a new functionality *try-with-resources* block which automatically cleans and disposes the resources. The similar one is being used here, otherwise I would have used a finally block to manually call the .close() function. This is well described on the [Java documentations](http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html).

The model doesn't need to interact with the view or the controller, instead the controller would trigger the functions of the model that is why we're not going to write any other code, any code apart from getting/setting the data of our application. That is the exact reason I am not importing any other package in this class (apart from that Java IO API for data access), otherwise in other classes you will see that I will be making an import to other classes and packages as whole.

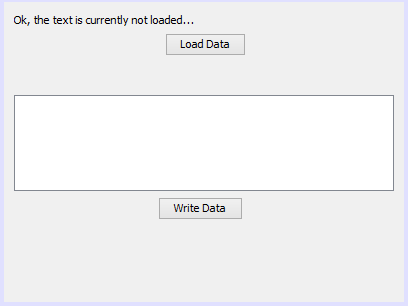
**Creating the view - *using the NetBeans***

*Note: If you're using Eclipse IDE, then you can simply write the code from my View, Eclipse doesn't support drag-and-drop for Swing GUI like NetBeans does.*

Now the time has come for the View portion of the application. A similar process should be taken under considerations. A new package, with name JavaMVCViews would be created and a class for HelloWorldView would be created. But in a slight different manner. You should create the package first. Once you've created it. Create a new swing JFrame form. Fill the details, and then continue. In my case the name was HelloWorldView.



NetBeans allows the developer to use a tool box, to drag-and-drop the controls over the window box and create the GUI for their applications. You can use the same, and create a simple GUI that would look like the following one.



This is our view for the application, and contains enough controls for our application to *work like it has to*. View's work is just to display what controller has allowed it to. This view contains a few controls, that controller *should be able to interact with* to display them or hide them on will.

For developers, here is the code that you can write in your application to create the similar view.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

*/\**

*\* To change this license header, choose License Headers in Project Properties.*

*\* To change this template file, choose Tools | Templates*

*\* and open the template in the editor.*

*\*/*

package JavaMVCViews;

*// Call the import*

import JavaMVCControllers.\*;

*/\*\**

*\**

*\* @author AfzaalAhmad*

*\*/*

public class HelloWorldView extends javax.swing.JFrame {

*// Create the class for the Controller*

private HelloWorldController controller = new HelloWorldController();

*/\*\**

*\* Creates new form HelloWorldView*

*\*/*

public HelloWorldView() {

initComponents();

}

*/\*\**

*\* This method is called from within the constructor to initialize the form.*

*\* WARNING: Do NOT modify this code. The content of this method is always*

*\* regenerated by the Form Editor.*

*\*/*

@SuppressWarnings("unchecked")

*// <editor-fold defaultstate="collapsed" desc="Generated Code">*

private void initComponents() {

myLabel = new javax.swing.JLabel();

loadData = new javax.swing.JButton();

jScrollPane1 = new javax.swing.JScrollPane();

myMessage = new javax.swing.JTextArea();

writeData = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT\_CURSOR));

setName("myFrame"); *// NOI18N*

myLabel.setText("Ok, the text is currently not loaded...");

loadData.setText("Load Data");

loadData.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseClicked(java.awt.event.MouseEvent evt) {

loadDataMouseClicked(evt);

}

});

myMessage.setColumns(20);

myMessage.setRows(5);

jScrollPane1.setViewportView(myMessage);

writeData.setText("Write Data");

writeData.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseClicked(java.awt.event.MouseEvent evt) {

writeDataMouseClicked(evt);

}

});

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(myLabel, javax.swing.GroupLayout.DEFAULT\_SIZE, 380, Short.MAX\_VALUE)

.addComponent(jScrollPane1))

.addContainerGap())

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGap(0, 0, Short.MAX\_VALUE)

.addComponent(loadData)

.addGap(158, 158, 158))))

.addGroup(layout.createSequentialGroup()

.addGap(154, 154, 154)

.addComponent(writeData)

.addGap(0, 0, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(myLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(loadData)

.addGap(39, 39, 39)

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(writeData)

.addContainerGap(82, Short.MAX\_VALUE))

);

pack();

}*// </editor-fold>*

private void loadDataMouseClicked(java.awt.event.MouseEvent evt) {

*// TODO add your handling code here:*

try {

String data = controller.getMessage();

myLabel.setText(data);

myLabel.setVisible(true);

} catch (Exception er) {

}

}

private void writeDataMouseClicked(java.awt.event.MouseEvent evt) {

*// TODO add your handling code here:*

String message = myMessage.getText();

controller.writeMessage(message);

}

*/\*\**

*\* @param args the command line arguments*

*\*/*

public static void main(String args[]) {

*/\* Set the Nimbus look and feel \*/*

*//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">*

*/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.*

*\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html*

*\*/*

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(HelloWorldView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(HelloWorldView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(HelloWorldView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(HelloWorldView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

*//</editor-fold>*

*/\* Create and display the form \*/*

java.awt.EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

new HelloWorldView().setVisible(true);

}

});

}

*// Variables declaration - do not modify*

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JButton loadData;

public javax.swing.JLabel myLabel;

private javax.swing.JTextArea myMessage;

private javax.swing.JButton writeData;

*// End of variables declaration*

}

The above code is to create the View. In NetBeans, the IDE hides all of the fuzzy code, and just displays the code that is required. In Eclipse you would be required to write it all. So that is why, you can use this code in the Eclipse to build the GUI.

*The above code was a result of the drag-and-drop actions. I****didn't****write anything apart from****function calls****.*

**Creating the controller**

The controller is the very vital section of our application. It contains the back-end code and the logic, for connecting the application's views (indirectly the user; because the user is going to use the view to interact with the application) with the models (the data source of our applications).

Similar process should be repeated, create a new package, name it JavaMVCControllers, create a new class and name it HelloWorldController. This class would control all of the logic of our application, so it is vital that this application has enough functions and flexibility to handle all of the events and requirements for our application. Such as, whenever the view needs to do something the controller must have a function defined to be triggered as well as for the model, to load the data from model into the view or to load the data from the view to the model; to let the model save the data in the disk.

Once the class file has been generated write the following code to it.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

*/\**

*\* To change this license header, choose License Headers in Project Properties.*

*\* To change this template file, choose Tools | Templates*

*\* and open the template in the editor.*

*\*/*

package JavaMVCControllers;

import JavaMVCViews.\*;

import JavaMVCModels.\*;

*/\*\**

*\**

*\* @author AfzaalAhmad*

*\*/*

public class HelloWorldController {

public void startApplication() {

*// View the application's GUI*

HelloWorldView view = new HelloWorldView();

view.setVisible(true);

}

public String getMessage() {

try {

HelloWorldModel model = new HelloWorldModel();

return model.getData();

} catch (Exception er) {

return "There was an error.";

}

}

public boolean writeMessage(String message) {

try {

HelloWorldModel model = new HelloWorldModel();

return model.writeData(message);

} catch (Exception er) {

return false;

}

}

}

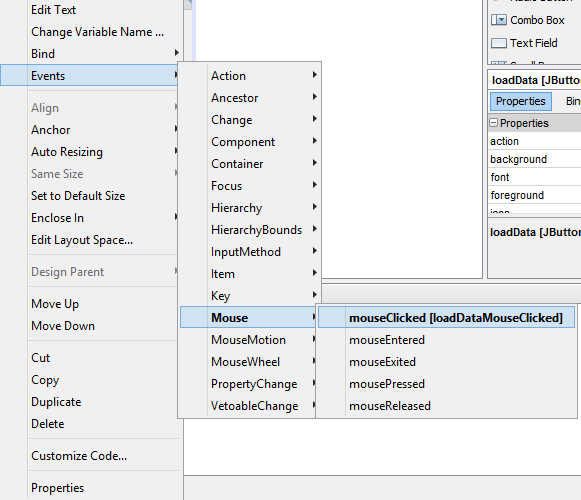
The controller here access the view package as well as the model package in our application. It contains three methods, one is the startApplication which starts the application's GUI by calling the view and setting it to be visible.

Other two functions are just to write the data to the model, or to extract the data from the model and returning it to the view. The main function is the startApplication function. We would call this function, to actually start the application. Why is it so important? In Java's Swing API, a JFrame is set to be visible once its setVisible property is set to be true. That is why, inside this function, the GUI is visible and the application now looks like *it just started*. It can be thought of as the *entry point* of our *MVC application* inside our *HelloJavaCollections project*.

**Handling the events in the View**

There was a step left in the View that I skipped for a later time, *time has come*. In Swing, you can handle the events, such as mouse, keyboard or other calls that are made to the control. In our application, we're interested in the mouse events only to handle the mouse click event on the buttons of our.

In NetBeans you can right click over the button (in the GUI builder), and the in the events, under the mouse click on mouseClick event.



The GUI builder will automatically add the code to handle the event, and will let you write the code to execute once the trigger is made. For example, in our code, once the trigger was handled; event to handle it was created, the GUI builder would take you to the source code panel, and would let you write the code under the event. The two events in our application are,

Hide   Copy Code

private void loadDataMouseClicked(java.awt.event.MouseEvent evt) {

*// TODO add your handling code here:*

try {

String data = controller.getMessage();

myLabel.setText(data);

myLabel.setVisible(true);

} catch (Exception er) {

}

}

private void writeDataMouseClicked(java.awt.event.MouseEvent evt) {

*// TODO add your handling code here:*

String message = myMessage.getText();

controller.writeMessage(message);

}

These events would execute once the user click on the buttons; depending on which button he clicks on.

Diving deeper, explaining more

Now the code has been set. The application is now set to run, work as we want it to. The controller is running and would run the view to render the application's GUI, and the user can interact to load the data or to update the data in the file that is at the back-end as the data source.

Now, let us see how the application runs, what functions are called first and how the MVC pattern works in our small *Hello world* application. Starting from the first stage and moving all the way to the last stages in our application...

***Running* the application**

When you run the application, as default (if you didn't change the behavior) the default entry point is set to the JavaHelloCollections class's Main function. The source code in the top of the article, is the one I'm talking about. If you will run the application it won't show anything, because the function is empty and it would terminate *without doing anything real*. So, change the application and introduce the MVC pattern of our application. Like this,

Hide   Copy Code

*/\**

*\* To change this license header, choose License Headers in Project Properties.*

*\* To change this template file, choose Tools | Templates*

*\* and open the template in the editor.*

*\*/*

package javahellocollections;

import JavaMVCControllers.\*;

*/\*\**

*\**

*\* @author AfzaalAhmad*

*\*/*

public class JavaHelloCollections {

*/\*\**

*\* @param args the command line arguments*

*\*/*

public static void main(String[] args) {

*// TODO code application logic here*

HelloWorldController controller = new HelloWorldController();

*// Start the application*

controller.startApplication();

}

}

What I did is, just added the controller, and triggered the startApplication function in it. Now once this would execute, the application would run; the *GUI would be visible*.

**Under the**startApplication

Now let us see what happened when the startApplication was triggered. As soon as the startApplication was triggered the control moved over to the Controller; controller now controls the application and every trigger must be made through controller now, View and Model both depend on the controller now. It is not a good approach to let all of these three interact, it would kill the main purpose of the MVC pattern. Inside the Controller, the function is something like this,

Hide   Copy Code

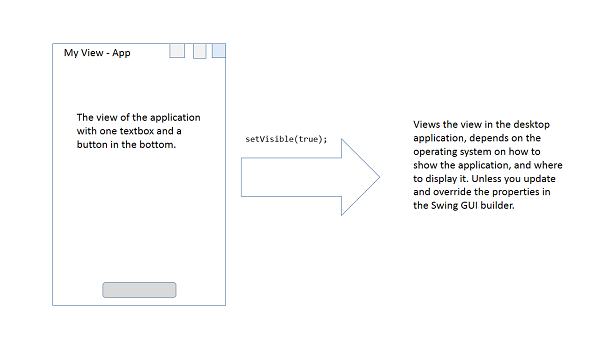
public void startApplication() {

*// View the application's GUI*

HelloWorldView view = new HelloWorldView();

view.setVisible(true);

}

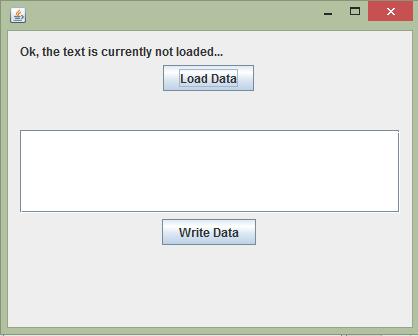


**View has been loaded**

Up to this stage, the control is still in the hands of controller, but not directly. Directly it seems as if the control were in the hands of the user using the application. Because no other code would execute until the user asks the application to perform something, like close application, load data or save the data and so on. Once he orders something, then the controller would again come in action and would perform the action returning the control back in the hands of user.

This method goes on, until the user interacts with the view himself. Now, let us run our application and see what our application show us.

*Note that the default look and feel set by the NetBeans is*[*Nimbus*](http://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/nimbus.html)*, I didn't bother editing it.*



Now that the view has been loaded, let us tinker with it a little. Should we? Smile | :)

**Loading/Writing the data**

Now the user clicked on the load data button, the button was attached to an event that handles the click. Inside that event we're going to call the controller, to provide us with some data. We will be using that data from controller to load into the JLabel we're having; the one showing "Ok, the text is currently not loaded...".

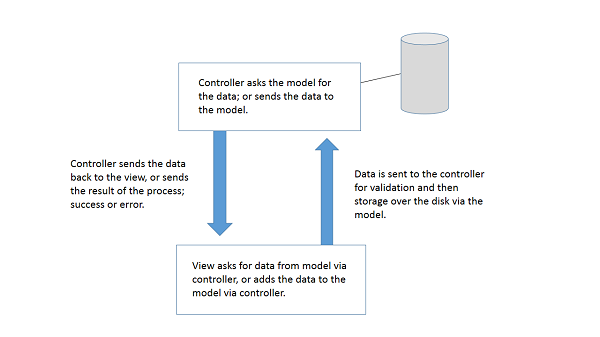
Hide   Copy Code

String data = controller.getMessage();

myLabel.setText(data);

myLabel.setVisible(true);

To make sure that the changes are visible, you call this function to recreate the view and show it to the user.



The above image that I drew, demonstrates how the controller, view and the model interact when the user wants to transmit the data, or to receive the data. Let us assume the first event, loading the data, view makes a request to the controller to provide the data. The controller then calls the model for the data, if the data is accessed then the data is returned back to the view otherwise the error; errors can be multiple, ranging from unauthorized user, or disk errors. So it depends on many factors what sort of data would be returned.

**Inside the**getMessage

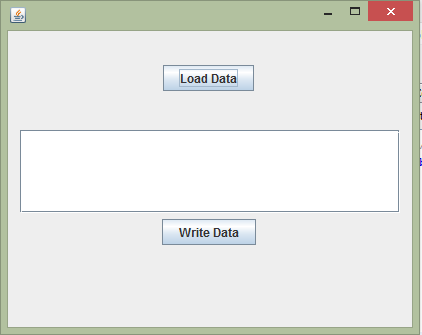
Inside the getMessage function, the call is now made to the model. To load the data into our stream. The code for that in our application is as,

Hide   Copy Code

HelloWorldModel model = new HelloWorldModel();

return model.getData();

Now this would return the data, based on the same above stream that I drew. The data would be captured by the controller, passed to the view for rendering. Let us see, what happens in our code; in our application, the file doesn't exist up to now, so there won't be anything displayed to the user because the file is empty after the creation.



Now letting the user to write some data in the file. Then we will handle the same event once again, but with an opposite traffic. This time the data won't be coming from data source to the view via controller. But to the *data source from the view via the controller*.

**Writing the data**

Now, suppose the user filled the value, and clicked the *write data* button. What would happen? The controller would again come into action, to perform the data store actions. You shouldn't connect your view to your data source directly.

Hide   Copy Code

String message = myMessage.getText();

controller.writeMessage(message);

The above code is simple, it gets the data from the text field, and then requests the controller to write the message on the disk. Let us see what is under the hood there.

**Inside the writeMessage**

Inside this function, the model is again called, but in an opposite manner. The data passed is then stored on the disk; inside that file.

Hide   Copy Code

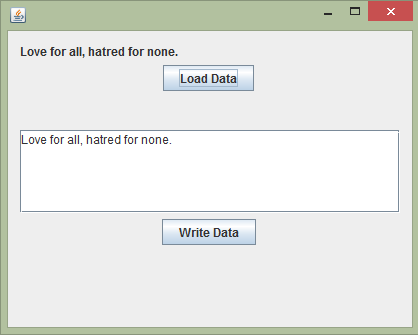
HelloWorldModel model = new HelloWorldModel();

return model.writeData(message);

Now the model would store the data, according to the methods and logic specified in the model class.

*I have not talked about the methods in the model, because they're about the Java language itself, and the Java IO API and have nothing to do with the MVC pattern.*

Once the data is stored on the file, you can again request the data. Once again, following the MVC connection, the data comes from the data source via the controller all the way down to the view. The view rendered the data on the screen and then shows the data.



The data once came back, had something in it. So the swing framework viewed it without any problem at all.

Points of Interest

In this article you learned about the MVC framework (pattern) of software development, how one can implement this framework in his software development, to distribute the source code into three different layers.

1. Model - for the data.
2. View - for the user interface.
3. Controller - for the back-end code.

In Java, you're required to install the SDK for Java and then an IDE supporting Java programming language to develop applications. Application that is developed, can have any programming architecture used in it. In our case we used the MVC framework, there are some others too like MVVM.

In Java, you combine and merge the similar classes into *packages* like you create a namespace in C# or C++. These packages are useful in distributing the code of same category in different packages under different name.

You can create as many package in an application as you want and then you can reference them in your projects. Classes of same project, do not need to import the classes, but the classes outside of your package need to make a reference to your package before they can access your classes.

In a MVC pattern, the data is not directly accessible by the user, instead the controller handles all the requests from and to the model in order to prove the validity and the consistency of the data. The view can trigger different commands, using which we can force the application to perform actions.

In MVC, you split the source code so that once the application reaches the level of enterprise solution you do not have to scratch your head or pull your hairs just because your code looks fuzzy now. MVC allows you to maintain the code separately, focusing over one section at a time. Either that one is a view, a model or the controller itself. You can debug your application more fast by knowing which of the division needs your attention at the moment and leaving other two as they are.

History

First post of this article.

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License (CPOL)](http://www.codeproject.com/info/cpol10.aspx)