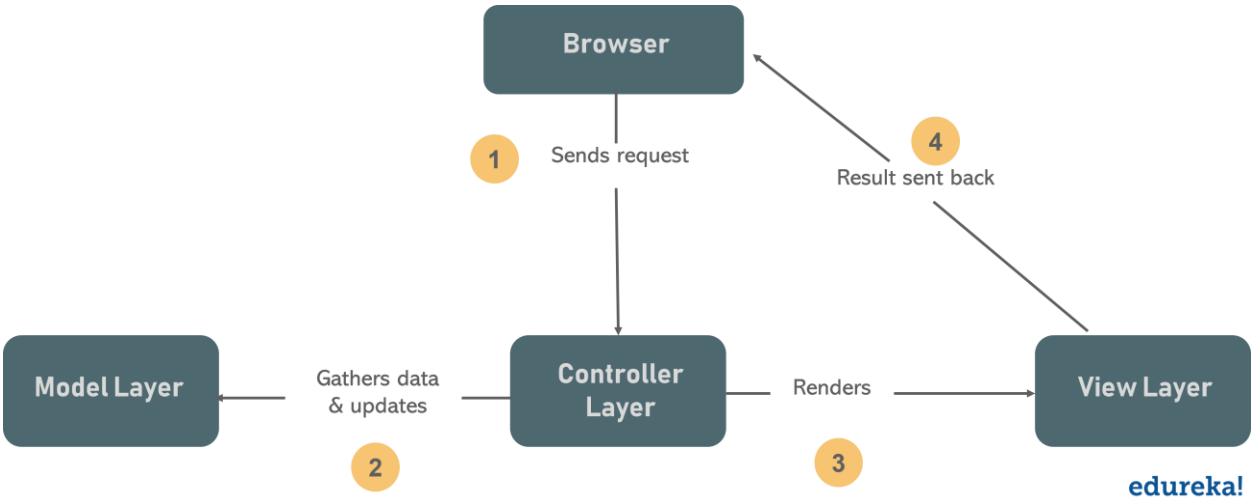
What is MVC Architecture in Java?

Model designs based on MVC architecture follow the MVC design pattern and they separate the application logic from the user interface when designing software. As the name implies MVC pattern has three layers, which are:

- ***Model*** – Represents the business layer of the application
- ***View*** – Defines the presentation of the application
- ***Controller*** – Manages the flow of the application



In Java Programming context, the Model consists of simple Java classes, the View displays the data and the Controller consists of servlets. This separation results in user requests being processed as follows:

1. The browser on the client sends a request for a page to the controller present on the server
2. The controller performs the action of invoking the model, thereby, retrieving the data it needs in response to the request
3. The controller then gives the retrieved data to the view
4. The view is rendered and sent back to the client for the browser to display

Separating a software application into these three distinct components is a good idea for a number of reasons. Let's take a look at what those are.

Advantages of MVC Architecture in Java

MVC architecture offers a lot of advantages for a programmer when developing applications, which include:

- Multiple developers can work with the three layers (Model, View, and Controller) simultaneously
- Offers improved *scalability*, that supplements the ability of the application to grow
- As components have a low dependency on each other, they are easy to maintain
- A model can be reused by multiple views which provides reusability of code
- Adoption of MVC makes an application more expressive and easy to understand
- Extending and testing of the application becomes easy

Now you know why the MVC is the most popular design patterns in the web programming world. But, if you are still struggling to get your head around the concept of MVC, don't worry. We will dig deep into each of these layers and learn their purpose with the help of an example program.

Implementation of MVC using Java

To implement a web application based on MVC design pattern, we will create

- ***Course Class***, which acts as the *model layer*
- ***CourseView Class***, which defines the presentation layer (*view layer*)
- ***CourseContoller Class***, which acts as a *controller*

Now, let's explore these layers one by one.

The Model Layer

In the MVC design pattern, the *model* is the data layer which defines the business logic of the system and also represents the state of the application. The model objects retrieve and store the state of the model in a database. Through this layer, we apply rules to data, which eventually represents the concepts our application manages. Now, let's create a model using *Course Class.*

```
package MyPackage;

public class Course {
	private String CourseName;
	private String CourseId;
	private String CourseCategory;

	public String getId() {
	   return CourseId;
	}

	public void setId(String id) {
	   this.CourseId = id;
	}

	public String getName() {
	   return CourseName;
	}

	public void setName(String name) {
	   this.CourseName = name;
	}

	public String getCategory() {
		return CourseCategory;
	    }

	public void setCategory(String category) {
		this.CourseCategory = category;
	    }

   }
```

The code is easy to understand and is self-explanatory. It consists of functions to get/set course details.

The View Layer

This layer of the MVC design pattern represents the output of the application or the user interface. It displays the data fetched from the model layer by the controller and presents the data to the user whenever asked for. It receives all the information it needs from the controller and it doesn't need to interact with the business layer directly. Let's create a view using *CourseView Class.*

```
package MyPackage;

public class CourseView {
      public void printCourseDetails(String CourseName, String CourseId, String CourseCategory
         System.out.println("Course Details: ");
         System.out.println("Name: " + CourseName);
         System.out.println("Course ID: " + CourseId);
         System.out.println("Course Category: " + CourseCategory);
      }
   }
```

This code is simply to print the values to the console. Next up we have the controller of the web application.

The Controller Layer

The Controller is like an interface between Model and View. It receives the user requests from the view layer and processes them, including the necessary validations. The requests are then sent to model for data processing. Once they are processed, the data is again sent back to the controller and then displayed on the view. Let's create *CourseContoller Class* which acts as a controller.

```
package MyPackage;

public class CourseController {
      private Course model;
      private CourseView view;

      public CourseController(Course model, CourseView view){
         this.model = model;
         this.view = view;
      }

      public void setCourseName(String name){
         model.setName(name);
      }

      public String getCourseName(){
         return model.getName();
```

```
17      }
18
19      public void setCourseId(String id){
20         model.setId(id);
21      }
22
23      public String getCourseId(){
24         return model.getId();
25      }
26
27      public void setCourseCategory(String category){
28            model.setCategory(category);
29      }
30
31          public String getCourseCategory(){
32             return model.getCategory();
33      }
34      public void updateView(){
35         view.printCourseDetails(model.getName(), model.getId(), model.getCategory());
36      }
37   }
38
```

A cursory glance at the code will tell us that this controller class is just responsible for calling the model to get/set the data and updating the view based on that. Now let's have a look at how all of these are tied together.

Main Java Class

Let's call this class "MVCPatternDemo.java". Check out the code below.

```
1
2  package MyPackage;
3
4  public class MVCPatternDemo {
5        public static void main(String[] args) {
6           //fetch student record based on his roll no from the database
7           Course model  = retriveCourseFromDatabase();
8
9           //Create a view : to write course details on console
10          CourseView view = new CourseView();
11
12          CourseController controller = new CourseController(model, view);
13
14          controller.updateView();
15
          //update model data
```

```
         controller.setCourseName("Python");
         System.out.println("nAfter updating, Course Details are as follows");

         controller.updateView();
      }

      private static Course retriveCourseFromDatabase(){
         Course course = new Course();
         course.setName("Java");
         course.setId("01");
         course.setCategory("Programming");
         return course;
      }
   }
```

The above class fetches the course data from the function using which user enters the set of values. It then pushes those values into the Course model. Then, it initializes the view we had created earlier in the article. Further, it also invokes the *CourseController* class and binds it to the *Course* class and the *CourseView* class. The *updateView()* method which is a part of the controller then updates the course details on the console. Check out the output below.

Output

```
Course Details:
Name: Java
Course ID: 01
Course Category: Programming

After updating, Course Details are as follows
Course Details:
Name: Python
Course ID: 01
Course Category: Programming
```

The MVC Architecture provides an altogether new level of modularity to your code which makes it a lot more readable and maintainable. This brings us to the end of this article. Hope you are clear with all that has been shared with you.