Практическая работа № 17. Разработка интерактивных программ на языке Джава с использованием паттерна MVC

Цель: введение в разработку программ с с использованием событийного программирования на языке программирования Джава с использованием паттерна MVC.

Теоретические сведения

Шаблон проектирования в программной инженерии — это метод решения часто возникающей проблемы при разработке программного обеспечения. Проектирование по модели указывает, какой тип архитектуры вы используете для решения проблемы или разработки модели.

Существует два *типа моделей проектирования*:

- *архитектура модели 1*
- *архитектура модели 2 (MVC).*

Архитектура MVC в Джава приложениях

Проекты моделей, основанные на архитектуре MVC (model-view-controller), следуют шаблону проектирования MVC и отделяют логику приложения от пользовательского интерфейса при разработке программного обеспечения.

Как следует из названия, шаблон MVC имеет три уровня:

- Модель — представляет бизнес-уровень приложения (model).
- Вид — определяет представление приложения (view).
- Контроллер — управляет потоком приложения (controller).

На рис. 17.1 представлена схема работы архитектуры MVC

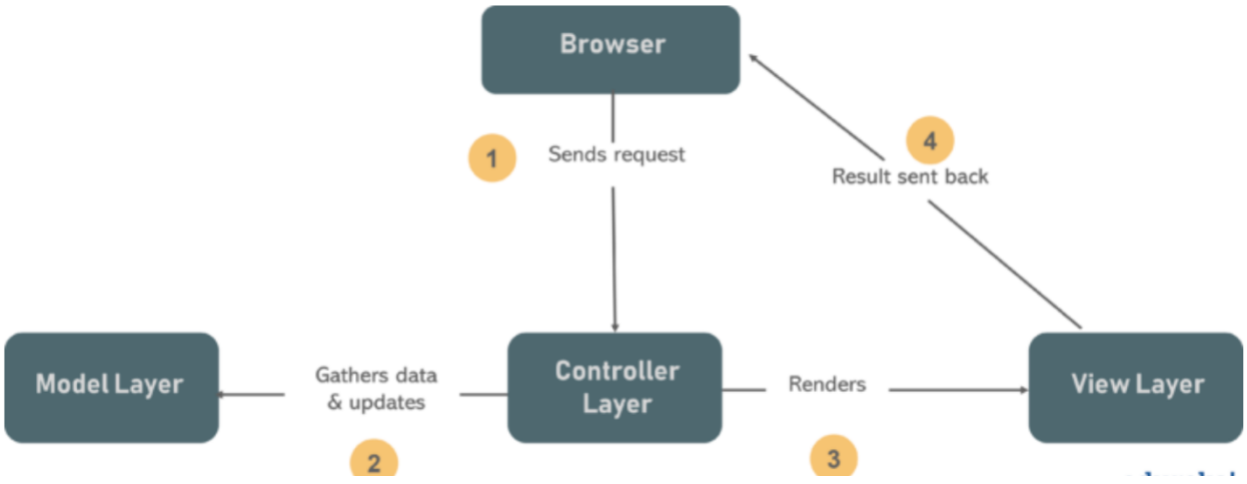

Рисунок 17.1. Архитектура MVC

В контексте программирования модель на Java состоит из простых классов Java, уровень представление отображает данные – то что видит пользователь, а контроллер состоит из сервлетов. Это разделение приводит к тому, что запросы пользователей обрабатываются следующим образом:

1. Браузер на клиенте отправляет запрос на страницу контроллеру, присутствующему на сервере.
2. Контроллер выполняет действие по вызову модели, тем самым извлекая необходимые ему данные в ответ на запрос.
3. Затем контроллер передает полученные данные в представление.
4. Представление визуализируется и отправляется обратно клиенту для отображения в браузере.

Разделение программного приложения на эти три отдельных компонента является хорошей идей по ряду причин.

Преимущества архитектуры MVC в Джава

Архитектура MVC предлагает множество преимуществ для программиста при разработке приложений, в том числе:

- Несколько разработчиков могут одновременно работать с тремя уровнями (модель, представление и контроллер).
- Предлагает улучшенную *масштабируемость* , которая дополняет возможность роста приложения.
- Поскольку компоненты мало зависят друг от друга, их легко поддерживать.
- Модель может быть повторно использована несколькими представлениями, что обеспечивает возможность повторного использования кода.
- Применение MVC делает приложение более выразительным и простым для понимания.
- Расширение и тестирование приложения становится проще

Шаблон MVC является самым популярным шаблоном проектирования для веб разработки.

Пример реализации паттерна MVC с использованием языка Джава

Чтобы реализовать веб-приложение на основе шаблона проектирования MVC, мы создадим следующие классы:

- Класс Course, который действует как слой модели
- Класс CourseView, определяющий уровень представления (слой представления)
- Класс CourseContoller, который действует как контроллер

Уровень слоя модели

В шаблоне проектирования *модель* MVC представляет собой уровень данных, который определяет бизнес-логику системы, а также представляет состояние приложения. Объекты модели извлекают и сохраняют состояние модели в базе данных. На этом уровне мы применяем правила к данным, которые в конечном итоге представляют концепции, которыми управляет наше приложение. Теперь давайте создадим модель с помощью класса Course см листинг 17.1

Листинг 17.1 Класс Course

```
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;
    }

}
```

Код прост для понимания и не требует пояснений. Он состоит факттически из методов геттеров и сеттеров для получения/установки сведений о курсе.

Уровень слоя представления

Этот уровень шаблона проектирования MVC представляет выходные данные приложения или пользовательского интерфейса. Он отображает данные, полученные контроллером из уровня модели, и предоставляет данные пользователю по запросу. Он получает всю необходимую информацию от контроллера, и ему не нужно напрямую взаимодействовать с бизнес-уровнем. Давайте создадим представление с помощью класса CourseView см листинг 17.2.

Листинг 17.2 Класс *CourseView*.

```
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);
    }
}
```

Этот код просто выводит значения на консоль. Далее нам предстоит создать контроллер для отслеживания событий на уровне представления данных и изменения модели данных. Этот слой факттически отвечает за бизнес логику приложения.

Уровень слоя контроллера

Контроллер похож на интерфейс между моделью и представлением. Он получает пользовательские запросы от уровня представления и обрабатывает их, включая необходимые проверки. Затем запросы отправляются в модель для обработки данных. После обработки данные снова отправляются обратно в контроллер, а затем отображаются в представлении. Давайте создадим класс CourseContoller, который действует как контроллер см листинг 17.3.

Листинг 17.3 Класс CourseContoller

```
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();
    }

    public void setCourseId(String id){
        model.setId(id);
    }

    public String getCourseId(){
        return model.getId();
    }

    public void setCourseCategory(String category){
        model.setCategory(category);
```

```
    }

    public String getCourseCategory(){
        return model.getCategory();
    }
    public void updateView(){
        view.printCourseDetails(model.getName(),
model.getId(), model.getCategory());
    }
}
```

Из кода на листинге 17.3 видно на, что этот класс контроллера просто отвечает за вызов модели для получения/установки данных и обновления представления на основе этой информации. Теперь соединим все вместе, для этого напишем тестовый класс, назовем его MVCPatternDemo.java.

Листинг 17.4 Класс MVCPatternDemo

```
package myPackage;

public class MVCPatternDemo {
    public static void main(String[] args) {

        //fetch student record based on his roll no
from the database
        Course model  = retriveCourseFromDatabase();

        //Create a view : to write course details on
console
        CourseView view = new CourseView();

        CourseController controller = new
CourseController(model, view);

        controller.updateView();

        //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;
    }
}

}
```

Приведенный на листинге класс извлекает данные курса из функции с помощью которой пользователь вводит набор значений. Затем он помещает эти значения в модель курса. Затем он инициализирует новое представление, которое мы создали ранее. Кроме того, он также вызывает класс CourseController и связывает его с классом Course и классом CourseView, метод updateView() , который является частью контроллера, затем обновляет сведения о курсе на консоли.

Результат работы программы

```
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
```

Вывод. Архитектура MVC обеспечивает совершенно новый уровень модульности вашего кода, что делает его более читабельным и удобным в сопровождении.

Задания на практическую работу № 1

1. Напишите реализацию программного кода по UML диаграмме, представленной на рис.17.2 . Прграмма должна демонстрировать

использование паттерна MVC.

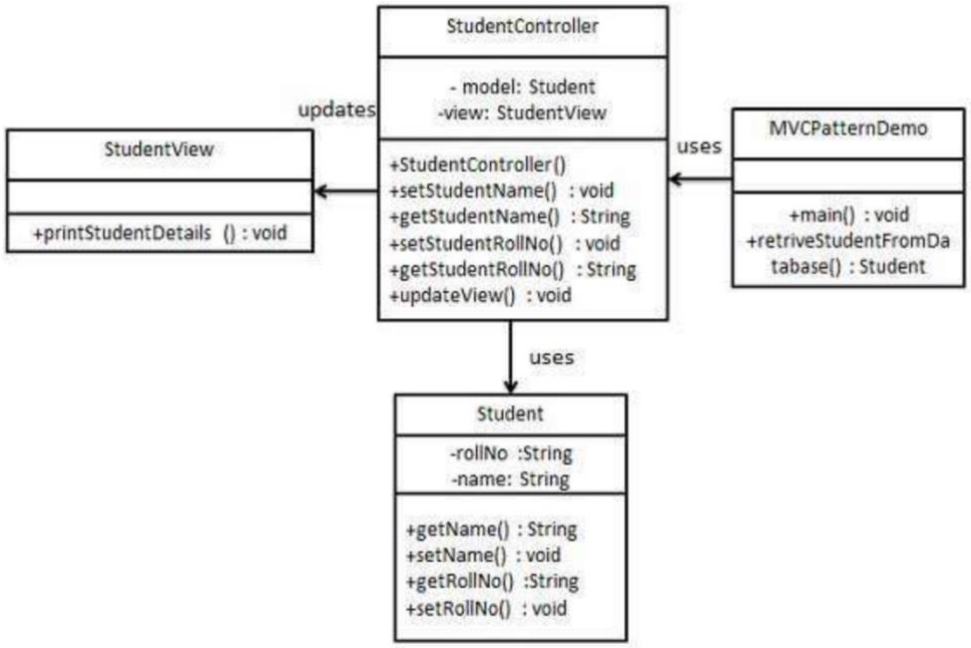

Рисунок 17.2. UML диаграмма классов проекта, реализующего MVC

2. Напишите реализацию программного кода, с использованием паттерна MVC для расчета заработной платы сотрудника предприятия. Предлагается использовать следующие классы.

- Класс Employee – сотрудник будет выступать в качестве слоя модели
- Класс EmployeeView будет действовать как слой представления.
- Класс EmployeeContoller будет действовать как уровень контроллера.

3. Вы можете написать программную реализацию, используя собственную идею, реализуя паттерн MVC. Выполнение задания предполагаеьт создание GUI.