Part A	
Class B Tech CSE 2nd Year	**Sub: Design Pattern Lab**
Aim: Write a program to implement the Model View Control Design Pattern	
Prerequisite: Basics of Object Oriented Programming	
Outcome: To impart knowledge of Design Pattern techniques	
Theory: Explain about the Model View Control Design Pattern.	

Model View Control

The Model-View-Controller (MVC) design pattern is a widely used architectural pattern for developing user interfaces. It separates an application into three interconnected components: Model, View, and Controller, each with distinct responsibilities.

Model: Represents the application's data and business logic. It encapsulates data manipulation, storage, and validation operations. The Model notifies registered observers (typically views) about changes in the data.

View: Represents the user interface components of the application. It displays the data from the model to the user and sends user input to the controller for processing. Views observe changes in the model and update themselves accordingly.

Controller: Acts as an intermediary between the model and the view. It processes user input, performs operations on the model, and updates the view accordingly. Controllers handle user actions, such as button clicks or menu selections, and update the model or view as necessary.

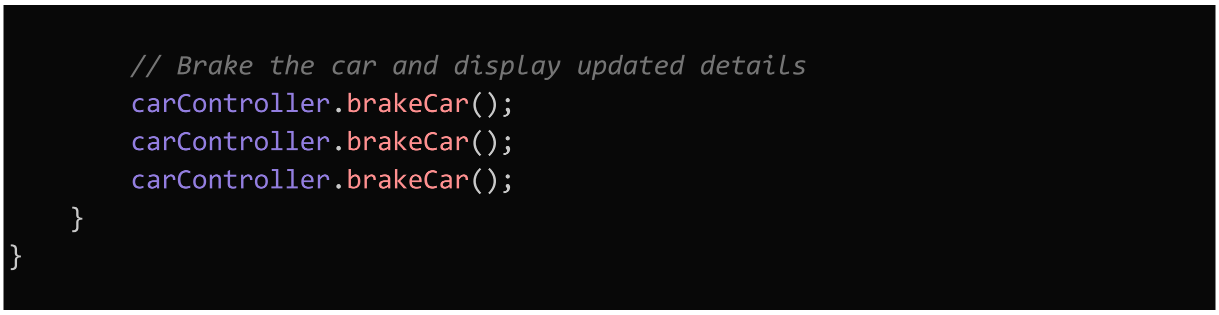
Key Characteristics:

- **Separation of Concerns:** MVC separates the concerns of data manipulation, user interface presentation, and user input handling into distinct components, making the application easier to understand, maintain, and scale.
- **Modularity and Reusability:** Each component (Model, View, Controller) can be developed independently, promoting code modularity, reusability, and testability.
- **Flexibility:** MVC allows for different implementations of each component, providing flexibility in choosing the most suitable technologies or frameworks for each part of the application.

Part B
Steps: Write procedure/code here.
Output: Paste Output here.
Observation & Learning: Give your observation and learning about the experiment.
Conclusion: Give your conclusion for the experiment.

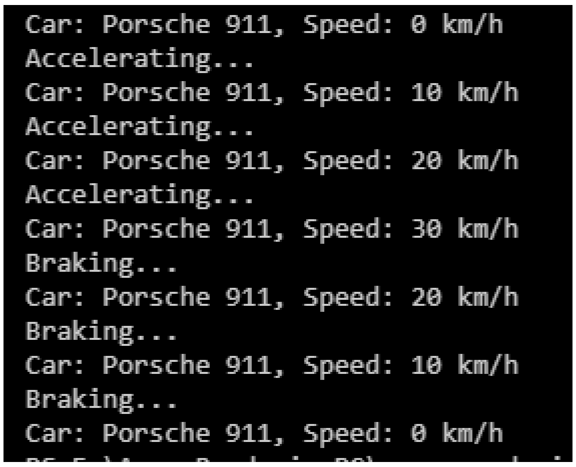
Code:

```java
// Model: Car
class Car {
    private String brand;
    private String model;
    private int speed;

    public Car(String brand, String model) {
        this.brand = brand;
        this.model = model;
        this.speed = 0;
    }

    public String getBrand() {
        return brand;
    }

    public String getModel() {
        return model;
    }

    public int getSpeed() {
        return speed;
    }

    public void accelerate() {
        speed += 10;
    }

    public void brake() {
        if (speed > 0) {
            speed -= 10;
        }
    }
}

// View: CarView
class CarView {
    public void displayCarDetails(String brand, String model, int speed) {
        System.out.println("Car: " + brand + " " + model + ", Speed: " + speed
+ " km/h");
    }
}

// Controller: CarController
class CarController {
    private Car model;
    private CarView view;

    public CarController(Car model, CarView view) {
        this.model = model;
        this.view = view;
    }

    public void updateView() {
        view.displayCarDetails(model.getBrand(), model.getModel(),
model.getSpeed());
    }

    public void accelerateCar() {
        System.err.println("Accelerating...");
        model.accelerate();
        updateView();
    }

    public void brakeCar() {
        System.err.println("Braking...");
        model.brake();
        updateView();
    }
}

// Main class
public class Pr15_MVC {
    public static void main(String[] args) {
        // Create model, view, and controller
        Car carModel = new Car("Porsche", "911");
        CarView carView = new CarView();
        CarController carController = new CarController(carModel, carView);

        // Display initial car details
        carController.updateView();

        // Accelerate the car and display updated details
        carController.accelerateCar();
        carController.accelerateCar();
        carController.accelerateCar();
```



```
        // Brake the car and display updated details
        carController.brakeCar();
        carController.brakeCar();
        carController.brakeCar();
    }
}
```

Output:



```
Car: Porsche 911, Speed: 0 km/h
Accelerating...
Car: Porsche 911, Speed: 10 km/h
Accelerating...
Car: Porsche 911, Speed: 20 km/h
Accelerating...
Car: Porsche 911, Speed: 30 km/h
Braking...
Car: Porsche 911, Speed: 20 km/h
Braking...
Car: Porsche 911, Speed: 10 km/h
Braking...
Car: Porsche 911, Speed: 0 km/h
```

Observation & Learning:

- **Separation of Concerns**: Implementing MVC helped in clearly separating the concerns of data manipulation (Model), user interface presentation (View), and user input handling (Controller). This separation made the codebase more organized and easier to understand.

- **Modularity and Reusability**: Each component (Model, View, Controller) could be developed and tested independently, promoting modularity and reusability. For instance, if there's a need to change the user interface, only the View component needs to be modified, leaving the Model and Controller unaffected.

- **Flexibility and Scalability**: MVC's flexibility allowed for different implementations of each component. For example, different views (GUI, command-line interface, web interface) can be easily implemented without changing the underlying model or controller logic. This flexibility makes it easier to scale and adapt the application to changing requirements.

Conclusion:

The MVC design pattern is a powerful architectural pattern widely used in software development to structure applications, particularly those with user interfaces. By separating concerns into Model, View, and Controller components, MVC promotes code organization, modularity, and maintainability, making it a popular choice for building scalable and maintainable software systems.