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

State Structural Pattern

The State pattern is a behavioral design pattern that allows an object to alter its behavior when its internal state changes. The pattern enables an object to behave differently based on its internal state, and it encapsulates state-specific behavior into separate classes.

Intent:

- The State pattern is used to model state-dependent behavior in an object-oriented manner.
- It allows an object to change its behavior when its internal state changes, without changing its class.

Key Components:

1. **Context**: It's the object that has an internal state and whose behavior changes based on that state. It maintains a reference to the current state object and delegates state-specific requests to that object.
2. **State**: It's an interface or an abstract class that defines a set of methods that encapsulate the behavior associated with a particular state of the Context. Concrete State classes implement these methods to provide state-specific behavior.

Structure:

- **Context**: Maintains a reference to a State object that represents the current state. It delegates state-specific requests to the current State object.
- **State**: Defines an interface or an abstract class for encapsulating the behavior associated with a particular state of the Context.
- **Concrete State**: Implements the behavior associated with a particular state of the Context.

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:

```
// State interface
interface DrivingMode {
    void accelerate();

    void brake();
}

// Concrete State classes
class EcoMode implements DrivingMode {
    @Override
    public void accelerate() {
        System.out.println("Driving smoothly in Eco Mode");
    }

    @Override
    public void brake() {
        System.out.println("Braking gently in Eco Mode");
    }
}

class SportMode implements DrivingMode {
    @Override
    public void accelerate() {
        System.out.println("Driving aggressively in Sport Mode");
    }

    @Override
    public void brake() {
        System.out.println("Braking sharply in Sport Mode");
    }
}

class NormalMode implements DrivingMode {
    @Override
    public void accelerate() {
        System.out.println("Driving steadily in Normal Mode");
    }

    @Override
```

```
    public void brake() {
        System.out.println("Braking normally in Normal Mode");
    }
}

// Context class
class Car {
    private DrivingMode currentMode;

    public Car(DrivingMode mode) {
        this.currentMode = mode;
    }

    public void setDrivingMode(DrivingMode mode) {
        System.out.println("Switching to " + mode.getClass().getSimpleName());
        this.currentMode = mode;
    }

    public void accelerate() {
        currentMode.accelerate();
    }

    public void brake() {
        currentMode.brake();
    }
}

// Client code
public class Pr13_State {
    public static void main(String[] args) {
        Car car = new Car(new NormalMode());

        car.accelerate();
        car.brake();

        car.setDrivingMode(new EcoMode());
        car.accelerate();
        car.brake();

        car.setDrivingMode(new SportMode());
        car.accelerate();
        car.brake();
    }
}
```

Output:

```
Driving steadily in Normal Mode
Braking normally in Normal Mode
Switching to EcoMode
Driving smoothly in Eco Mode
Braking gently in Eco Mode
Switching to SportMode
Driving aggressively in Sport Mode
Braking sharply in Sport Mode
```

Observation & Learning:

The state design pattern helps objects change their behaviour based on their internal state. In this experiment, we saw how it works by separating each state's behaviour into its own class. This makes it easier to manage and update the behaviour without messing up the rest of the code. By using this pattern, we learned that we can keep our code organized, flexible, and easier to work with.

Conclusion:

The experiment showed us that the state design pattern is a useful tool for managing how objects behave in software. It keeps things tidy and makes it simple to add or change behaviours without causing chaos in the code. Overall, it's a smart way to design software systems that are easy to understand and maintain.