Experiment – 13

Aim –State design pattern (Behavioral pattern)

Concept -**State** is a behavioral design pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.

FoodOrderState.java

```java
//STATE INTERFACE

public interface FoodOrderState {
    void takeOrder(FoodOrder fo);
    void prepareOrder(FoodOrder fo);
    void deliveredOrder(FoodOrder fo);
}
```

FoodOrder.java

```java
//CONTEXT CLASS

public class FoodOrder {
    private FoodOrderState state;

    public FoodOrder() {
        state = new OrderingState();
    }

    public void setState(FoodOrderState state) {
        this.state = state;
    }

    public void takeOrder() {
        state.takeOrder(this);
    }

    public void prepareOrder() {
        state.prepareOrder(this);
    }

    public void deliveredOrder() {
        state.deliveredOrder(this);
    }
}
```

OrderingState.java

```java
public class OrderingState implements FoodOrderState{

    @Override
    public void takeOrder(FoodOrder fo) {
        fo.setState(new PreparingState());
        System.out.println("order taken");
    }

    @Override
    public void prepareOrder(FoodOrder fo) {
        System.out.println("order can not be prepared until it is taken");
    }

    @Override
    public void deliveredOrder(FoodOrder fo) {
        System.out.println("order can not be delivered until it is prepared");
    }
}
```

PreparingState.java

```java
public class PreparingState implements FoodOrderState {

    @Override
    public void takeOrder(FoodOrder fo) {
        System.out.println("order has already been taken");
    }

    @Override
    public void prepareOrder(FoodOrder fo) {
        fo.setState(new CookedState());
        System.out.println("order is being prepared");
    }

    @Override
    public void deliveredOrder(FoodOrder fo) {
        System.out.println("order cannot be delivered until it is cooked");
    }
}
```

CookedState.java

```java
public class CookedState implements FoodOrderState{

    @Override
    public void takeOrder(FoodOrder fo) {
        System.out.println("order has already been taken");
    }

    @Override
    public void prepareOrder(FoodOrder fo) {
        System.out.println("order has already been cooked");
    }

    @Override
    public void deliveredOrder(FoodOrder fo) {
        fo.setState(new DeliveredState());
        System.out.println("Order is being delivered");
    }
}
```

DeliveredState.java

```java
public class DeliveredState implements FoodOrderState{
    @Override
    public void takeOrder(FoodOrder fo) {
        System.out.println("order has already been taken");
    }

    @Override
    public void prepareOrder(FoodOrder fo) {
        System.out.println("order has already been cooked");
    }

    @Override
    public void deliveredOrder(FoodOrder fo) {
        System.out.println("order has already been delivered");
    }
}
```

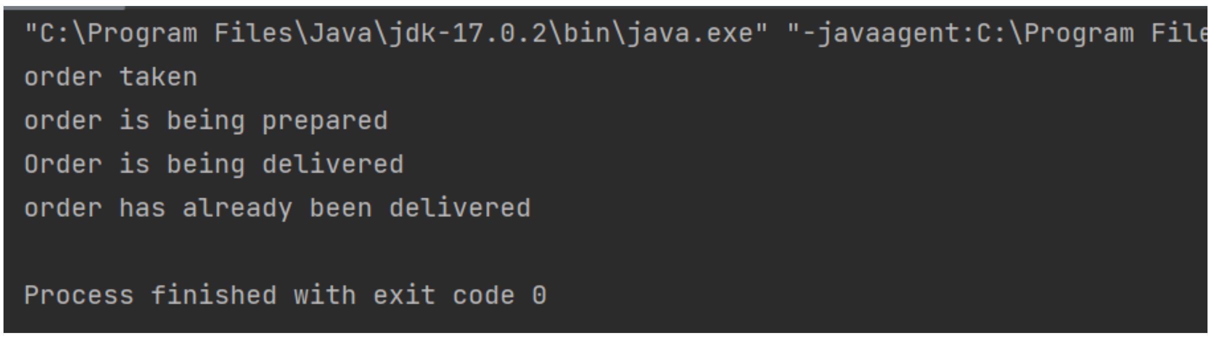
Main.java

```java
public class Main {
    public static void main(String[] args) {
        FoodOrder order = new FoodOrder();

        order.takeOrder();
        order.prepareOrder();
        order.deliveredOrder();

        // Attempt to transition from an invalid state
        order.deliveredOrder();
    }
}
```

OUTPUT :-



```
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:C:\Program File
order taken
order is being prepared
Order is being delivered
order has already been delivered

Process finished with exit code 0
```