**Exercise 01.**

1. Implement an Interface called command with a method signature execute().

Java

Public interface command {

Void execute();

}

1. Implement an Interface called Light with method signatures on() and off().

Java

Public interface Light {

Void on();

Void off();

}

1. Create two concrete classes KitchenLight and LivingRoom wich implement Light interface.

Java

Public class Kitchenlight implements Light {

Public void on() {

System.out.println(“Kitchen light is ON”);

}

Public void off() {

System.out.println(“Kitchen light is OFF”);

}

}

Public class LivingRoomLight implements Light {

Public void on() {

System.out.println(“Living room light is ON”);

}

Public void off() {

System.out.println(“Living room light is OFF”);

}

}

1. Implement two concrete classes, LightOnCommand and LightOFFCommand, off the Command interface.

Java

Public class LightOnCommand implements Command {

Private Light light;

Public LightOnCommand(Light light) {

This.light = light

}

Public void execute() {

Light.on();

}

}

Public class LightOffCommand implements Command {

Private Light light;

Public LightOffCommand(Light light) {

This.light = light;

}

Public void execute() {

Light.off();

}

}

1. Now create a test class that checks every on and off command for each light.

Java

Public class CommandPattern Test {

Publc static void main(String[] args) {

Light KitchenLight = New KitchenLight();

Light livingRoomLight = New livingRoomLight();

Command KitchenLightOn = New LightOnCommand(KitchenLight);

Control KitchenLightOff = New LightOffCommand(KitchenLight);

Command livingRoomLightOn = New LightOnCommand(LivingRoomLight);

Command livingRoomLightOff = New LightOffCommand(LivingRoomLight);

KitchenLightOn.execute(); // Kitchen Light is ON

KitchenLightOff.execute(); // Kitchen Light is OFF

livingRoomLightOn.execute(); // Kitchen Light is ON

livingRoomLightOff.execute(); // Kitchen Light is OFF

}

}

**Exercise 02.**

1. Design an abstract class calles Beverages and extend it using Tea and Coffee concrete class.

Java

Public abstract class Beverages {

Final void prepareRecipe() {

boilWater();

brew();

pourlnCup();

addCondiments();

}

Abstract void brew();

Abstract void addCondiments();

Void BoileWater() {

System.out.println(“Boiling Water”);

}

Void pourlnCup() {

System.out.println(“Pouring into Cup”);

}

}

Public class Tea extends Beverages {

Void brew() {

System.out.println(“Steeping the Tea”);

}

Void addCondiments() {

System.out.println(“Adding Lemon”);

}

}

Public class Coffee extends Beverages {

Void brew() {

System.out.println(“Dripping Coffee Through Filter”);

}

Void addCondiments() {

System.out.println(“Adding Sugar And Milk”);

}

}

1. Test class that tries to the pattern of template method.

Java

Public class TemplateMethodTest {

Public static void main(String[] args){

Beverages Tea = New Tea();

Drink Tea = New Tea();

System.out.println(“Prepared Tea”);

Tea.PrepareRecipe();

System.out.println(“

Making Coffee.”);

Cofee.PrepareRecipe();

}

}

**Exercise 03**

1. Crete an Interface called shape with a method signature draw().

Java

Public interface Shape {

Void draw();

}

1. Implement 3-4 concreate class of shape such as Square,Circle,Triangle,Rectangle,etc.

Java

Public class Square implements Shape {

Public void draw() {

System.out.println(“Drawing a Square”);

}

}

Public class Circle implements Shape {

Public void draw() {

System.out.println(“Drawing a Circle”);

}

}

Public class Traingle implements Shape {

Public void draw() {

System.out.println(“Drawing a Traingle”);

}

}

Public class Rectangle implements Shape {

Public void draw() {

System.out.println(“Drawing a Rectangle”);

}

}

1. Create a ShapeFactory class with a method getShape() That accepts a string and returns a shape.

Java

Public class ShapeFactory {

Public Shape getShape(String ShapeType) {

If (shapeType ==null) {

Return null;

}

Switch (shapeType.to.UpperCase() {

Case “SQUARE”:

Return new Square();

Case “CIRCLE”:

Return new Circle();

Case “TRIANGLE”

Return new Triangle();

Default:

Return null;

}

}

}

1. Test class to test the pattern functionality.

Java

Public class AbstractFactoryTest [

Public static void main(String[] args) {

ShapeFactory ShapeFactory = New ShapeFactory();

Shape Shape1 = ShapeFactory.getShape(“SQUARE”);

Shape1.draw();//Drawing a Square

Shape Shape2 = ShapeFactory.getShape(“CIRCLE”);

Shape2.draw();//Drawing a Circle

Shape Shape3 = ShapeFactory.getShape(“TRIANGLE”);

Shape3.draw();//Drawing a Triangle

}

}