1. Write a Java Program to implement 1/O Decorator for converting uppercase letters to lower case letters.

import java.io.\*;

public class LowercaseDecorator extends FilterReader { public LowercaseDecorator(Reader in) { super(in);

}

@Override

public int read() throws IOException {

int c = super.read();

return (c == -1) ? c : Character.toLowerCase((char) c);

}

@Override

public int read(char[] cbuf, int off, int len) throws IOException { int numCharsRead = super.read(cbuf, off, len); for (int i = off; i < off + numCharsRead; i++) { cbuf[i] = Character.toLowerCase(cbuf[i]);

}

return numCharsRead;

}

public static void main(String[] args) throws IOException { StringReader sr = new StringReader("HELLO WORLD"); LowercaseDecorator lsd = new LowercaseDecorator(sr); int c;

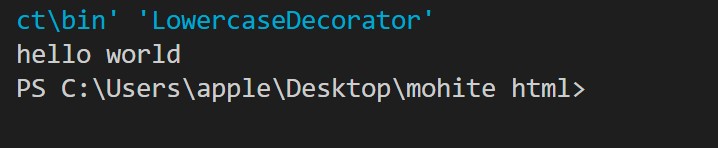
while ((c = lsd.read()) != -1) { System.out.print((char) c);

}

}

}

# OUTPUT:-



2. Write a Java Program to implement Factory method for Pizza Store with createPizza, orederPizza, prepare. Bake, cut, box. Use this to create variety of pizza's like NyStyleCheesePizza

abstract class Pizza { public abstract void prepare(); public abstract void bake(); public abstract void cut(); public abstract void box();

}

class NYStyleCheesePizza extends Pizza { public void prepare() { System.out.println("Preparing NY Style Cheese Pizza"); } public void bake() { System.out.println("Baking NY Style Cheese Pizza"); } public void cut() { System.out.println("Cutting NY Style Cheese Pizza"); } public void box() { System.out.println("Boxing NY Style Cheese Pizza"); }

}

abstract class PizzaStore { public Pizza orderPizza(String type) { Pizza pizza = createPizza(type); pizza.prepare(); pizza.bake(); pizza.cut(); pizza.box(); return pizza;

}

protected abstract Pizza createPizza(String type);

}

class NYPizzaStore extends PizzaStore {

@Override

protected Pizza createPizza(String type) { if (type.equals("cheese")) { return new NYStyleCheesePizza();

}

return null;

}

}

public class ex {

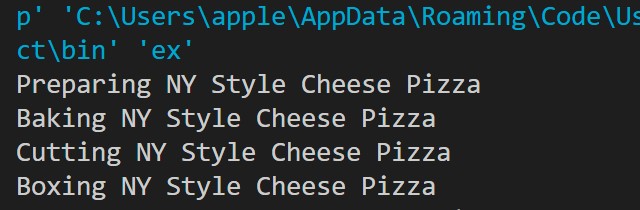
public static void main(String[] args) {

PizzaStore store = new NYPizzaStore(); store.orderPizza("cheese");

}

}

# OUTPUT:-



Write a Java Program to implement Singleton pattern for multithreading.

public class Singleton {

private static Singleton instance;

private Singleton() { }

public static synchronized Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

public static void main(String[] args) {

Runnable task = () -> {

Singleton singleton = Singleton.getInstance();

System.out.println(singleton);

};

Thread t1 = new Thread(task);

Thread t2 = new Thread(task);

t1.start(); t2.start();

}

}

Write a Java Program to implement Adapter pattern for Enumeration

Iterator

import java.util.\*;

class EnumerationAdapter implements Iterator<Object> { private Enumeration<?> enumeration;

public EnumerationAdapter(Enumeration<?> enumeration) {

this.enumeration = enumeration;

}

@Override public boolean hasNext() {

return enumeration.hasMoreElements();

}

@Override public Object next() {

return enumeration.nextElement();

}

}

public class AdapterPatternDemo { public static void main(String[] args) { Vector<String> vector = new Vector<>();

vector.add("One"); vector.add("Two"); vector.add("Three");

Enumeration<?> enumeration = vector.elements();

Iterator<?> iterator = new EnumerationAdapter(enumeration);

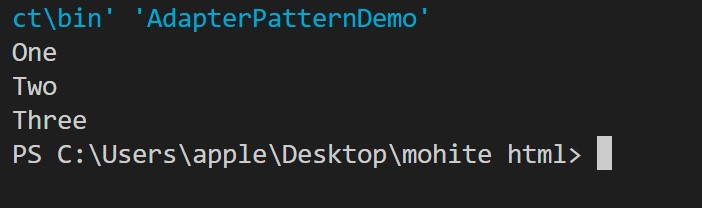
while (iterator.hasNext()) { System.out.println(iterator.next());

}

}

}

# OUTPUT:-



## Write a Java Program to implement command pattern to test Remote Control( to ON and OFF lightV ON and OFF ceiling Fan)

interface Command { void execute();

}

class Light { public void on() { System.out.println("Light is ON"); } public void off() { System.out.println("Light is OFF"); }

}

class CeilingFan { public void on() { System.out.println("Ceiling Fan is ON"); } public void off() { System.out.println("Ceiling Fan is OFF"); }

}

class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override public void execute() {

light.on();

}

}

class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override public void execute() { light.off();

}

}

class CeilingFanOnCommand implements Command {

private CeilingFan ceilingFan;

public CeilingFanOnCommand(CeilingFan ceilingFan) {

this.ceilingFan = ceilingFan;

}

@Override public void execute() { ceilingFan.on();

}

}

class CeilingFanOffCommand implements Command { private CeilingFan ceilingFan;

public CeilingFanOffCommand(CeilingFan ceilingFan) {

this.ceilingFan = ceilingFan;

}

@Override public void execute() {

ceilingFan.off();

}

}

class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

public class CommandPatternDemo {

public static void main(String[] args) {

Light light = new Light();

CeilingFan ceilingFan = new CeilingFan();

Command lightOn = new LightOnCommand(light);

Command lightOff = new LightOffCommand(light);

Command ceilingFanOn = new CeilingFanOnCommand(ceilingFan);

Command ceilingFanOff = new CeilingFanOffCommand(ceilingFan);

RemoteControl remote = new RemoteControl(); remote.setCommand(lightOn); remote.pressButton();

remote.setCommand(lightOff);

remote.pressButton();

remote.setCommand(ceilingFanOn);

remote.pressButton();

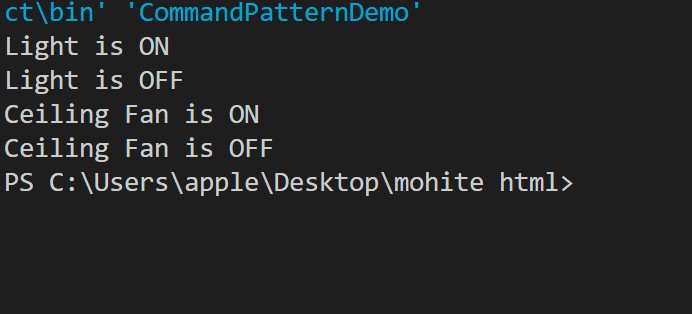
remote.setCommand(ceilingFanOff);

remote.pressButton();

}

}

# OUTPUT:-



**6.** **Write a Java Program to implement Strategy Pattern for Duck Behavior.**

## Create instance variable that holds current state of Duck from there, we just need to handle all Flying Behaviors and Quack Behaviour

public

interface FlyBehavior { void fly();

}

interface QuackBehavior {

void quack();

}

class FlyWithWings implements FlyBehavior {

public void fly() { System.out.println("I'm flying with wings!"); }

}

class FlyNoWay implements FlyBehavior {

public void fly() { System.out.println("I can't fly."); }

}

class Quack implements QuackBehavior {

public void quack() { System.out.println("Quack!"); }

}

class MuteQuack implements QuackBehavior {

public void quack() { System.out.println("<< Silence >>"); }

}

abstract class Duck { protected FlyBehavior flyBehavior;

protected QuackBehavior quackBehavior;

public Duck(FlyBehavior fb, QuackBehavior qb) { flyBehavior = fb; quackBehavior = qb;

}

public void performFly() {

flyBehavior.fly();

}

public void performQuack() { quackBehavior.quack();

}

public void swim() {

System.out.println("All ducks float!");

}

public abstract void display();

}

class MallardDuck extends Duck { public MallardDuck() {

super(new FlyWithWings(), new Quack());

}

public void display() {

System.out.println("I'm a real Mallard duck!");

}

}

class ModelDuck extends Duck { public ModelDuck() {

super(new FlyNoWay(), new MuteQuack());

}

public void display() {

System.out.println("I'm a model duck.");

}

}

public class StrategyPatternDemo { public static void main(String[] args) { Duck mallard = new MallardDuck();

mallard.display(); mallard.performFly(); mallard.performQuack();

Duck model = new ModelDuck();

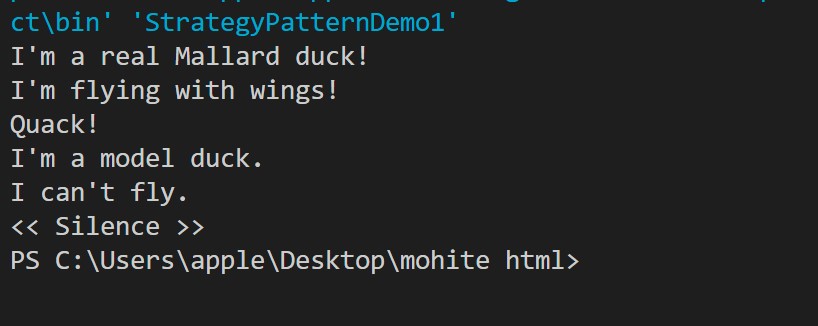
model.display(); model.performFly();

model.performQuack();

}

}

# OUTPUT:-



## 7. Write a Java Program to implement Decorator Pattern for interface Car to define the assemble() method and then decorate it to Sports car and Luxury Car

interface Car { String assemble();

}

class BasicCar implements Car { public String assemble() {

return "Basic Car";

}

}

abstract class CarDecorator implements Car {

protected Car car;

public CarDecorator(Car car) { this.car = car;

}

public String assemble() {

return car.assemble();

}

}

class SportsCar extends CarDecorator {

public SportsCar(Car car) {

super(car);

}

public String assemble() {

return super.assemble() + " + Sports Car Features";

}

}

class LuxuryCar extends CarDecorator {

public LuxuryCar(Car car) {

super(car);

}

public String assemble() {

return super.assemble() + " + Luxury Car Features";

}

}

public class DecoratorPatternDemo {

public static void main(String[] args) { Car sportsCar = new SportsCar(new BasicCar());

System.out.println(sportsCar.assemble());

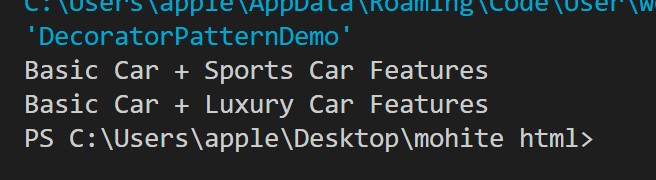
Car luxuryCar = new LuxuryCar(new BasicCar());

System.out.println(luxuryCar.assemble());

}

}

# OUTPUT:-



8. Write a Java Program to implement Abstract Factory Pattern for Shape interface

interface Shape {

void draw();

}

class Circle implements Shape {

@Override public void draw() {

System.out.println("Drawing Circle");

}

}

class Rectangle implements Shape {

@Override public void draw() {

System.out.println("Drawing Rectangle");

}

}

class Square implements Shape {

@Override public void draw() {

System.out.println("Drawing Square");

}

}

interface ShapeFactory {

Shape createShape();

}

class CircleFactory implements ShapeFactory {

@Override public Shape createShape() {

return new Circle();

}

}

class RectangleFactory implements ShapeFactory {

@Override public Shape createShape() { return new Rectangle();

}

}

class SquareFactory implements ShapeFactory {

@Override public Shape createShape() { return new Square();

}

}

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

ShapeFactory circleFactory = new CircleFactory(); Shape circle = circleFactory.createShape();

circle.draw();

ShapeFactory rectangleFactory = new RectangleFactory(); Shape rectangle = rectangleFactory.createShape();

rectangle.draw();

ShapeFactory squareFactory = new SquareFactory(); Shape square = squareFactory.createShape();

square.draw();

}

}