**1. Write a Java Program to implement Singleton pattern for multithreading.**

public class Main {

public static void main(String ar[]) {

Test1 t = new Test1();

Test1 t2 = new Test1();

Test1 t3 = new Test1();

Thread tt = new Thread(t);

Thread tt2 = new Thread(t2);

Thread tt3 = new Thread(t3);

Thread tt4 = new Thread(t);

Thread tt5 = new Thread(t);

tt.start();

tt2.start();

tt3.start();

tt4.start();

tt5.start();

}

}

final class Test1 implements Runnable {

@Override

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println(Thread.currentThread().getName() + " : " +

Single.getInstance().hashCode());

}

}

}

class Single {

private final static Single sing = new Single();

private Single() {

}

public static Single getInstance() {

return sing;

}

}

**o/p - Thread-1 : 13362967**

**Thread-0 : 13362967**

**Thread-0 : 13362967**

**Thread-0 : 13362967**

**Thread-0 : 13362967**

**Thread-0 : 13362967**

**Thread-3 : 13362967**

**Thread-3 : 13362967**

**Thread-3 : 13362967**

**Thread-2 : 13362967**

**Thread-2 : 13362967**

**Thread-2 : 13362967**

**Thread-2 : 13362967**

**Thread-2 : 13362967**

**Thread-4 : 13362967**

**Thread-3 : 13362967**

**Thread-3 : 13362967**

**Thread-1 : 13362967**

**Thread-1 : 13362967**

**Thread-4 : 13362967**

**Thread-4 : 13362967**

**Thread-4 : 13362967**

**Thread-1 : 13362967**

**Thread-4 : 13362967**

**Thread-1 : 13362967**

**2. Lower case decorator pattern**

import java.io.\*;

import java.util.\*;

class LowerCaseInputStream extends FilterInputStream

{

public LowerCaseInputStream(InputStream in) {

super(in);

}

public int read() throws IOException {

int c = super.read();

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

}

public int read(byte[] b, int offset, int len) throws IOException {

int result = super.read(b, offset, len);

for (int i = offset; i < offset+result; i++) {

b[i] = (byte)Character.toLowerCase((char)b[i]);

}

return result;

}

}

public class Main {

public static void main(String[] args) throws IOException

int c;

try {

InputStream in =

new LowerCaseInputStream(

new BufferedInputStream(

new FileInputStream("test.txt")));

while((c = in.read()) >= 0) {

System.out.print((char)c);

}

in.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**o/p - abcde**

**3. Factory pattern shape program**

**Create an interface.**

**Shape.java**

public interface Shape {

void draw();

}

**Rectangle.java**

public class Rectangle implements Shape {

@Override

public void draw() {

System.out.println("Inside Rectangle::draw() method.");

}

}

**Square.java**

public class Square implements Shape {

@Override

public void draw() {

System.out.println("Inside Square::draw() method.");

}

}

**Circle.java**

public class Circle implements Shape {

@Override

public void draw() {

System.out.println("Inside Circle::draw() method.");

}

}

**ShapeFactory.java**

public class ShapeFactory {

//use getShape method to get object of type shape

public Shape getShape(String shapeType){

if(shapeType == null){

return null;

}

if(shapeType.equalsIgnoreCase("CIRCLE")){

return new Circle();

} else if(shapeType.equalsIgnoreCase("RECTANGLE")){

return new Rectangle();

} else if(shapeType.equalsIgnoreCase("SQUARE")){

return new Square();

}

return null;

}

}

**FactoryPatternDemo.java**

public class FactoryPatternDemo {

public static void main(String[] args) {

ShapeFactory shapeFactory = new ShapeFactory();

//get an object of Circle and call its draw method.

Shape shape1 = shapeFactory.getShape("CIRCLE");

//call draw method of Circle

shape1.draw();

//get an object of Rectangle and call its draw method.

Shape shape2 = shapeFactory.getShape("RECTANGLE");

//call draw method of Rectangle

shape2.draw();

//get an object of Square and call its draw method.

Shape shape3 = shapeFactory.getShape("SQUARE");

//call draw method of square

shape3.draw();

}

}

**o/p - Inside Circle::draw() method.**

**Inside Rectangle::draw() method.**

**Inside Square::draw() method.**

**4. Adapter pattern Bird program**

// Java implementation of Adapter pattern

interface Bird

{

// birds implement Bird interface that allows

// them to fly and make sounds adaptee interface

public void fly();

public void makeSound();

}

class Sparrow implements Bird

{

// a concrete implementation of bird

public void fly()

{

System.out.println("Flying");

}

public void makeSound()

{

System.out.println("Chirp Chirp");

}

}

interface ToyDuck

{

// target interface

// toyducks dont fly they just make

// squeaking sound

public void squeak();

}

class PlasticToyDuck implements ToyDuck

{

public void squeak()

{

System.out.println("Squeak");

}

}

class BirdAdapter implements ToyDuck

{

// You need to implement the interface your

// client expects to use.

Bird bird;

public BirdAdapter(Bird bird)

{

// we need reference to the object we

// are adapting

this.bird = bird;

}

public void squeak()

{

// translate the methods appropriately

bird.makeSound();

}

}

class Main

{

public static void main(String args[])

{

Sparrow sparrow = new Sparrow();

ToyDuck toyDuck = new PlasticToyDuck();

// Wrap a bird in a birdAdapter so that it

// behaves like toy duck

ToyDuck birdAdapter = new BirdAdapter(sparrow);

System.out.println("Sparrow...");

sparrow.fly();

sparrow.makeSound();

System.out.println("ToyDuck...");

toyDuck.squeak();

// toy duck behaving like a bird

System.out.println("BirdAdapter...");

birdAdapter.squeak();

}

}

**o/p - Sparrow...**

**Flying**

**Chirp Chirp**

**ToyDuck...**

**Squeak**

**BirdAdapter...**

**Chirp Chirp**