SADP Practical Questions and Answers.txt

Q1) Write a java program to implement Singleton pattern for multithreading?

-->

(Main.java)

```
package slip2;
class Singleton{
   private static volatile Singleton uniqueInstance=null;

   private Singleton(){};

   public static Singleton getInstance(){
      if(uniqueInstance==null)
      {
         uniqueInstance=new Singleton();
      }
      return uniqueInstance;
   }
}
public class Main{
  public static void main(String[] args) {
    Singleton s = Singleton.getInstance();
    System.out.println("Object Created Sucessfully!!!");
  }

}
```

Q2) Write a java program to implement I/O decorator for converting uppercase letters to lower case letters.

-->

(LowerCaseInputStream.java)

```
package slip1;
import java.io.*;

public class LowerCaseInputStream extends FilterInputStream{
   public LowerCaseInputStream(InputStream in)
   {
      super(in);
   }
```

```
    public int read() throws IOException
    {
        int c=in.read();
        return (c==-1?-1:Character.toLowerCase((char)c));
    }

    public int read(byte[] b,int off,int len) throws IOException
    {
        int length=in.read(b,off,len);
        for(int i=off;i<len;++i)
        {
            b[i]=(b[i]>=65 && b[i]<=90)?(byte)(b[i]+32):b[i];
        }
        return length;
    }
}
```

(Slip1.java)

```
package slip1;

import java.io.*;
public class Slip1{
    public static void main(String[] args)
    {
        try
        {
            int c;
            InputStream in=new LowerCaseInputStream(new BufferedInputStream(new FileInputStream("a.txt")));
            while((c=in.read())>0)
            System.out.print((char)c);
        }
        catch(IOException io)
        {
            io.printStackTrace();
        }
    }
}
```

Q3). Factory Method Shape Program (TutorialsPoint)?

->

Step 1

Create an interface.

(Shape.java)

```
public interface Shape {
   void draw();
}
```

Step 2

Create concrete classes implementing the same interface.

(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.");
   }
}
```

Step 3

Create a Factory to generate object of concrete class based on given information.

(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;
   }
}
```

Step 4

Use the Factory to get object of concrete class by passing an information such as type.

(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();
   }
}
```

Q4). Adapter Pattern Bird Program (geeksforgeeks) ?
https://www.geeksforgeeks.org/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();
    }
}
```