Orchid International College

Faculty of Management

(TU)

A Lab Report

On

“Object Oriented Analysis and Design”

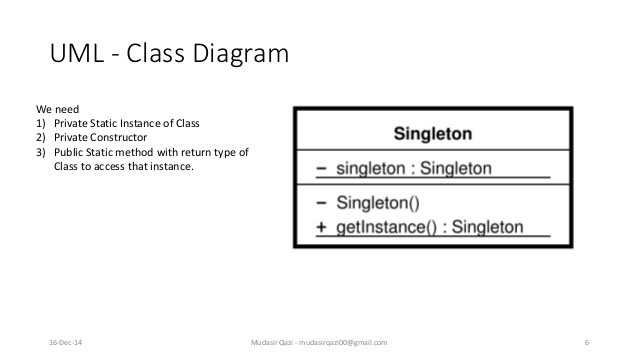
Submitted Date: 12 Feb, 2020

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Submitted By: | Submitted To: | | Name: Rakesh Bhatt | Department of Management | | TU Roll No.: 7449/16 |  | | Section: B |  | |  |

# **Chapter I Singleton Design Pattern**

## **Design Pattern Name: Singleton**

## **Descriptive Class Diagram**



## **Code in Java**

//Singleton.java

package ooad;

public class Singleton {

private static Singleton obj;

private Singleton() {

//do nothing

}

public static Singleton getInstance() {

if(obj == null) {

obj = new Singleton();

return obj;

}else{

return obj;

}

}//method ends

}//class ends

// TestSingleton.java

public class TestSingleton {

public static void main(String[] args) {

Singleton instance = Singleton.getInstance();

Singleton anotherInstance = Singleton.getInstance();

If(instance === anotherInstance){

System.out.println(“They both are same object”);

}else{

System.out.println(“They both are different object”);

}

}

Output:

They both are same object

# **Chapter II Factory Design Pattern**

## **Design Pattern Name: Factory**

## **Descriptive Class Diagram**



## **Code in Java**

package ooad;

// 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();

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

shape1.draw();

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

shape2.draw();

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

shape3.draw();

}

}

Output:

Inside Circle: draw() method.

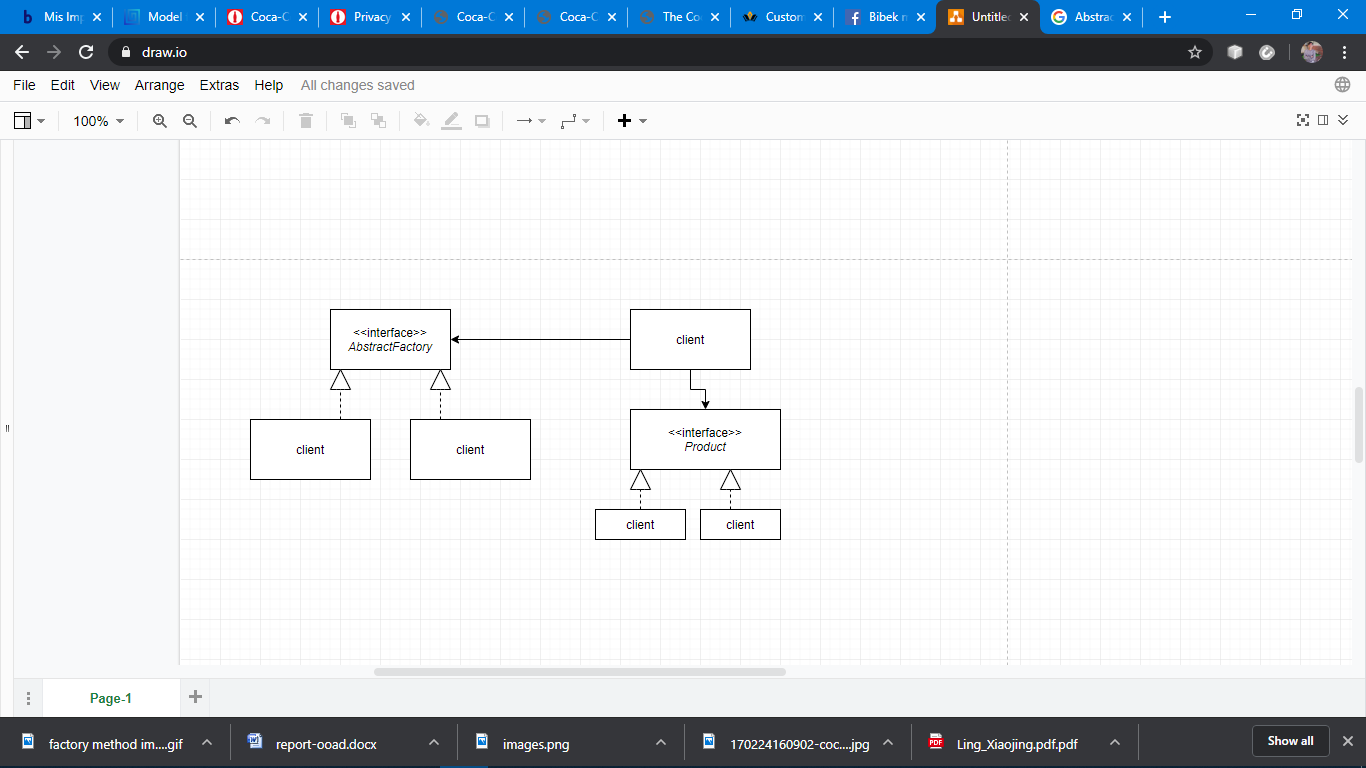
Inside Rectangle: draw() method.

Inside Square: draw() method.

# **Chapter III Abstract Factory Design Pattern**

## **3.1. Design Pattern Name: Abstract Factory**

## **3.2. Descriptive Class Diagram**



## **3.3. Code in Java**

package ooad;

// 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.");

}

}

// RoundedRectangle.java

public class RoundedRectangle implements Shape {

@Override

public void draw() {

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

}

}

// RoundedSquare.java

public class RoundedSquare implements Shape {

@Override

public void draw() {

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

}

}

// AbstractFactory.java

public abstract class AbstractFactory {

abstract Shape getShape(String shapeType) ;

}

//ShapeFactory.java

public class ShapeFactory extends AbstractFactory {

@Override

public Shape getShape(String shapeType){

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

return new Rectangle();

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

return new Square();

}

return null;

}

}

// RoundedShapeFactory.java

public class RoundedShapeFactory extends AbstractFactory {

@Override

public Shape getShape(String shapeType){

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

return new RoundedRectangle();

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

return new RoundedSquare();

}

return null;

}

}

// FactoryProducer.java

public class FactoryProducer {

public static AbstractFactory getFactory(boolean rounded){

if(rounded){

return new RoundedShapeFactory();

}else{

return new ShapeFactory();

}

}

}

// AbstractFactoryPatternDemo.java

public class AbstractFactoryPatternDemo {

public static void main(String[] args) {

AbstractFactory shapeFactory = FactoryProducer.getFactory(false);

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

shape1.draw();

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

shape2.draw();

AbstractFactory shapeFactory1 = FactoryProducer.getFactory(true);

Shape shape3 = shapeFactory1.getShape("RECTANGLE");

shape3.draw();

Shape shape4 = shapeFactory1.getShape("SQUARE");

shape4.draw();

}

}

Output:

Inside Rectangle: draw() method.

Inside Square: draw() method.

Inside RoundedRectangle: draw() method.

Inside RoundedSquare: draw() method.

# **Chapter IV Adapter Design Pattern**

## **4.1. Design Pattern Name: Adapter**

## **4.2. Descriptive Class Diagram**



## **4.3. Code in Java**

package com.ooad.demo;

// Bird.java

interface Bird {

public void fly();

public void makeSound();

}

// Sparrow.java

class Sparrow implements Bird {

public void fly() {

System.out.println("Flying");

}

public void makeSound() {

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

}

}

// ToyDuck.java

interface ToyDuck {

public void squeak();

}

// PlasticToyDuck.java

class PlasticToyDuck implements ToyDuck {

public void squeak() {

System.out.println("Squeak");

}

}

//BirdAdapter.java

class BirdAdapter implements ToyDuck {

Bird bird;

public BirdAdapter(Bird bird) {

this.bird = bird;

}

public void squeak() {

bird.makeSound();

}

}

//AdapterDemo.java

class AdapterDemo {

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();

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

birdAdapter.squeak();

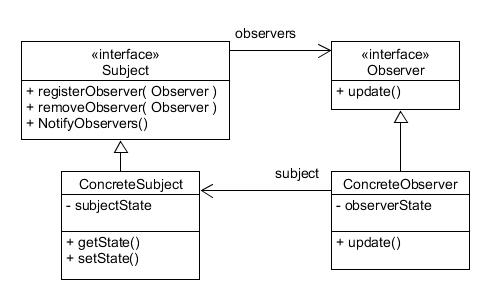
}

}

# **Chapter V Observer Design Pattern**

## **5.1. Design Pattern Name: Observer**

## **5.2. Descriptive Class Diagram**



## **5.3. Code in Java**

package com.ooad.demo;

// Student.java

import java.util.ArrayList;

import java.util.List;

public class Subject {

private List<Observer> observers = new ArrayList<Observer>();

private int state;

public int getState() {

return state;

}

public void setState(int state) {

this.state = state;

notifyAllObservers();

}

public void attach(Observer observer){

observers.add(observer);

}

public void notifyAllObservers(){

for (Observer observer : observers) {

observer.update();

}

}

}

//Observer.java

public abstract class Observer {

protected Subject subject;

public abstract void update();

}

// BinaryObserver.java

public class BinaryObserver extends Observer{

public BinaryObserver(Subject subject){

this.subject = subject;

this.subject.attach(this);

}

@Override

public void update() {

System.out.println( "Binary String: "+ Integer.toBinaryString(subject.getState()));

}

}

// OctalObserver.java

public class OctalObserver extends Observer{

public OctalObserver(Subject subject){

this.subject = subject;

this.subject.attach(this);

}

@Override

public void update() {

System.out.println( "Octal String: " + Integer.toOctalString( subject.getState() ) );

}

}

// HexaObserver.java

public class HexaObserver extends Observer{

public HexaObserver(Subject subject){

this.subject = subject;

this.subject.attach(this);

}

@Override

public void update() {

System.out.println( "Hex String: " + Integer.toHexString(subject.getState() ).toUpperCase() );

}

}

// ObserverPatternDemo.java

public class ObserverPatternDemo {

public static void main(String[] args) {

Subject subject = new Subject();

new HexaObserver(subject);

new OctalObserver(subject);

new BinaryObserver(subject);

System.out.println("First state change: 15");

subject.setState(15);

System.out.println("Second state change: 10");

subject.setState(10);

}

}

Output:

First state change: 15

Hex String: F

Octal String: 17

Binary String: 1111

Second state change: 10

Hex String: A

Octal String: 12

Binary String: 1010