**Singleton pattern example:**

|  |
| --- |
| // Class 1  // Helper class  **class** Singleton  {      // Static variable reference of single\_instance of type Singleton  **private** **static** Singleton single\_instance = **null**;  **public** String s;        // private constructor restricted to this class itself  **private** Singleton()      {          s = "Singleton class";      }        // Static method to create instance of Singleton class  **public** **static** Singleton getInstance()      {  **if** (single\_instance == **null**)              single\_instance = **new** Singleton();    **return** single\_instance;      }  }    // Class 2  // Main class  **class** SingletonClass  {  **public** **static** **void** main(String args[])      {          // Instantiating Singleton class with variable x          Singleton x = Singleton.getInstance();            // Instantiating Singleton class with variable y          Singleton y = Singleton.getInstance();            // Instantiating Singleton class with variable z          Singleton z = Singleton.getInstance();            // Printing the hash code for above variable as declared          System.out.println("Hashcode of x is "+ x.hashCode());          System.out.println("Hashcode of y is "+ y.hashCode());          System.out.println("Hashcode of z is "+ z.hashCode());    **if** (x == y && y == z) {               System.out.println( "Three objects point to the same memory location on the heap i.e, to the same object");          }    **else** {              System.out.println("Three objects DO NOT point to the same memory location on the heap");          }      }  } |

**Output**

Hashcode of x is 558638686

Hashcode of y is 558638686

Hashcode of z is 558638686

Three objects point to the same memory location on the heap i.e, to the same object

**Factory pattern example:**

**interface** Currency

{

       String getSymbol();

}

// Concrete Rupee Class code

**class** Rupee **implements** Currency

{

       @Override

**public** String getSymbol() {

**return** "Rs";

       }

}

// Concrete SGD class Code

**class** SGDDollar **implements** Currency

{

       @Override

**public** String getSymbol() {

**return** "SGD";

       }

}

// Concrete US Dollar code

**class** USDollar **implements** Currency

{

       @Override

**public** String getSymbol() {

**return** "USD";

       }

}

// Factroy Class code

**class** CurrencyFactory

{

**public** **static** Currency createCurrency (String country) {

**if** (country. equalsIgnoreCase ("India")){

**return** **new** Rupee();

       }**else** **if**(country. equalsIgnoreCase ("Singapore")){

**return** **new** SGDDollar();

       }**else** **if**(country. equalsIgnoreCase ("US")){

**return** **new** USDollar();

        }

**throw** **new** IllegalArgumentException("No such currency");

       }

}

// Factory client code

**public** **class** Factory

{

**public** **static** **void** main(String args[]) {

              String country = args[0];

              Currency rupee = CurrencyFactory.*createCurrency*(country);

              System.*out*.println(rupee.getSymbol());

       }

}

# Abstract Factory Pattern

public interface Shape {

void draw();

}

public class RoundedRectangle implements Shape {

@Override

public void draw() {

System.out.println("RoundedRectangle draw method executed.");

}

}

public class RoundedSquare implements Shape {

@Override

public void draw() {

System.out.println("RoundedSquare draw method executed.");

}

}

public class Rectangle implements Shape {

@Override

public void draw() {

System.out.println("Rectangle draw method executed.");

}

}

public abstract class AbstractFactory {

abstract Shape getShape(String shapeType) ;

}

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;

}

}

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;

}

}

public class FactoryProducer {

public static AbstractFactory getFactory(boolean rounded){

if(rounded){

return new RoundedShapeFactory();

}else{

return new ShapeFactory();

}

}

}

public class AbstractFactoryPatternDemo {

public static void main(String[] args) {

//get shape factory

AbstractFactory shapeFactory = FactoryProducer.getFactory(false);

//get an object of Shape Rectangle

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

//call draw method of Shape Rectangle

shape1.draw();

//get an object of Shape Square

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

//call draw method of Shape Square

shape2.draw();

//get shape factory

AbstractFactory shapeFactory1 = FactoryProducer.getFactory(true);

//get an object of Shape Rectangle

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

//call draw method of Shape Rectangle

shape3.draw();

//get an object of Shape Square

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

//call draw method of Shape Square

shape4.draw();

}

}

**Output:**

Rectangle draw method executed

RoundedRectangle draw method executed

RoundedSquare draw method executed.