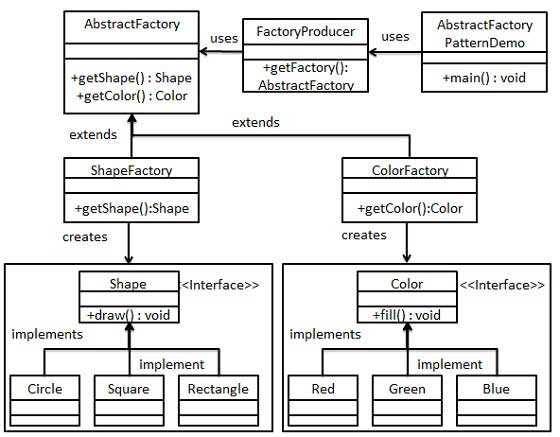
**Abstract Factory Design Pattern**

**Problem:**

The objective of this exercise is to implement the Abstract Factory design pattern.



**Steps:**

1. Create an interface called **Shape**. It defines a public method **draw**(). Create three subclasses (**Circle**, **Square**, and **Rectangle**) as indicated in the class diagram above. Each subclass implements draw() method by printing out a string as “Inside xxx.draw() method”. For example, in **Circle**, it should print out “inside Circle.draw() method”.

A screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

1. Create an interface called **Color**. It defines a public method **fill**(). Create three subclasses (**Red**, **Green**, and **Blue**) as indicated in the class diagram above. Each subclass implements fill() method by printing out a string as “Inside xxx.fill() method”. For example, in **Red**, it should print out “inside Red.fill() method”.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generatedA computer screen shot of a program code

Description automatically generatedCreate an Abstract class called **AbstractFactory** to get factories for Color and Shape Objects. Define two abstract methods:

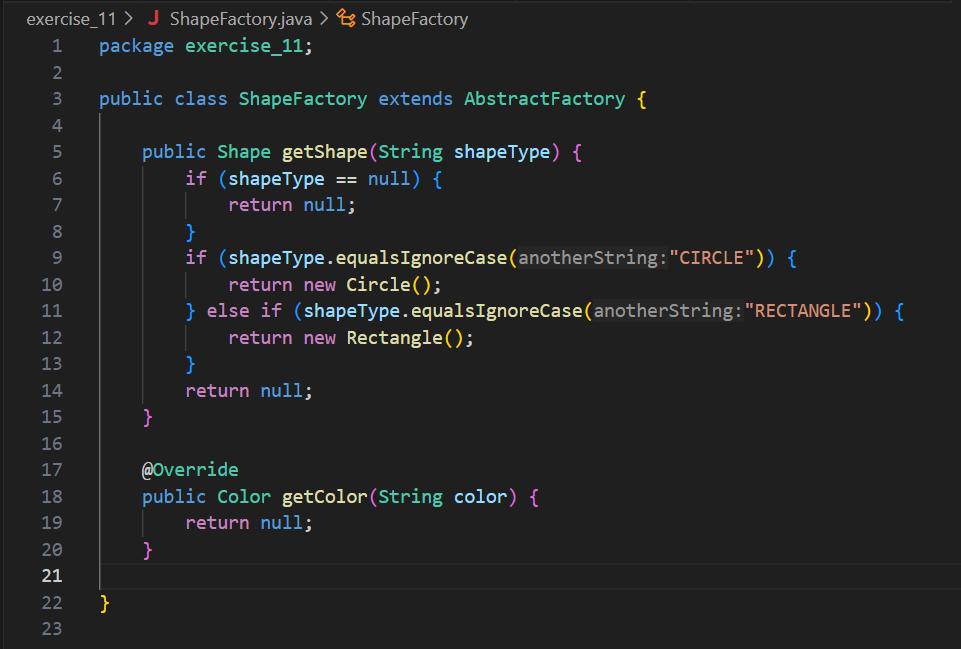
abstract Color getColor(String color);

abstract Shape getShape(String shape);

A screen shot of a computer code

Description automatically generated

1. Create Factory classes **ShapeFactory** and **ColorFactory** extending **AbstractFactory** to generate object of concrete class based on given information.
2. Implement **getShape** function in **ShapeFactory**, so that it checks the argument shapeType and create corresponding Shape objects. For example, if shapeType is “CIRCLE”, it should return new Circle().



Shape getShape(String shapeType){ ... … }

1. Implement **getColor** function in **ColorFactory**, so that it checks the argument colorType and create corresponding Color objects. For example, if colorType is “RED”, it should return new Red().

Color getColor(String colorType){ ... … }



1. Create a Factory generator/producer class called **FactoryProducer** to get factories by passing an information such as Shape or Color.

public class FactoryProducer {

public static AbstractFactory getFactory(String choice){

if(choice.equalsIgnoreCase("SHAPE")){

return new ShapeFactory();

}else if(choice.equalsIgnoreCase("COLOR")){

return new ColorFactory();

}

return null;

}

}

A screen shot of a computer program

Description automatically generated

1. Use the client below to check the results.

public class AbstractFactoryPatternDemo {

public static void main(String[] args) {

//get shape factory

AbstractFactory shapeFactory = FactoryProducer.getFactory("SHAPE");

//get an object of Shape Circle

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

//call draw method of Shape Circle

shape1.draw();

//get an object of Shape Rectangle

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

//call draw method of Shape Rectangle

shape2.draw();

//get an object of Shape Square

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

//call draw method of Shape Square

shape3.draw();

//get color factory

AbstractFactory colorFactory = FactoryProducer.getFactory("COLOR");

//get an object of Color Red

Color color1 = colorFactory.getColor("RED");

//call fill method of Red

color1.fill();

//get an object of Color Green

Color color2 = colorFactory.getColor("Green");

//call fill method of Green

color2.fill();

//get an object of Color Blue

Color color3 = colorFactory.getColor("BLUE");

//call fill method of Color Blue

color3.fill();

}

}

Upload your code to the Blackboard when you are done.