

Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

class Utility{

public static final String RECTANGLE = "RECTANGLE";

public static final String CIRCLE = "CIRCLE";

public static final String SQUARE = "SQUARE";

public static final String COLOR = "COLOR";

public static final String SHAPE = "SHAPE";

public static final String RED = "RED";

public static final String GREEN = "GREEN";

public static final String BLUE = "BLUE";

}

interface Shape{

public void draw();

}

class Rectangle implements Shape{

public void draw() {

System.out.println("inside rectangle draw");

}

}

class Square implements Shape{

public void draw() {

System.out.println("inside square draw");

}

}

class Circle implements Shape{

public void draw() {

System.out.println("inside circle draw");

}

}

interface Color{

public void fill();

}

class Red implements Color{

public void fill() {

System.out.println("inside red fill method");

}

}

class Green implements Color{

public void fill() {

System.out.println("inside green fill method");

}

}

class Blue implements Color{

@Override

public void fill() {

System.out.println("inside blue fill method");

}

}

abstract class AbstractFactory{

abstract Color getColor(String color);

abstract Shape getShape(String shape);

}

class ShapeFactory extends AbstractFactory{

public Shape getShape(String shape) {

switch(shape) {

case Utility.RECTANGLE : return new Rectangle();

case Utility.CIRCLE : return new Circle();

case Utility.SQUARE : return new Square();

}

return null;

}

public Color getColor(String color) {

return null;

}

}

class ColorFactory extends AbstractFactory{

public Color getColor(String color) {

switch(color) {

case Utility.RED : return new Red();

case Utility.GREEN :return new Green();

case Utility.BLUE : return new Blue();

}

return null;

}

public Shape getShape(String shape){

return null;

}

}

class FactoryProducer{

public static AbstractFactory getFactory(String choice) {

switch(choice) {

case Utility.COLOR : return new ColorFactory();

case Utility.SHAPE : return new ShapeFactory();

}

return null;

}

}

public class AbstractFactoryPatternDemo {

public static void main(String[] args) {

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

Shape shapeCircle = shapeFactory.getShape(Utility.CIRCLE);

shapeCircle.draw();

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

Color redColor = colorFactory.getColor(Utility.RED);

redColor.fill();

}

}