package pk.cui.sc.fm;

import java.util.ArrayList;

public class Demo {

public static void main(String[] args) {

ArrayList<Shape> shapes = new ArrayList<Shape>();

CircleGeometry circleGeometry = new CircleGeometry();

SquareGeometry squareGeometry = new SquareGeometry();

shapes.add(circleGeometry.createShape());

shapes.add(squareGeometry.createShape());

shapes.add(squareGeometry.createShape());

for(Shape s: shapes){

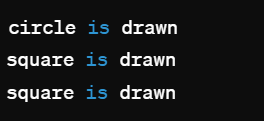
s.draw();

}

}

}

**OUTPUT:**

****

**And added another class of trianglefactory:**

package pk.cui.sc.fm;

import java.util.ArrayList;

public class Demo {

public static void main(String[] args) {

ArrayList<Shape> shapes = new ArrayList<Shape>();

GeometryFactory geometryFactory = new GeometryFactory();

shapes.add(geometryFactory.createShape("circle"));

shapes.add(geometryFactory.createShape("square"));

shapes.add(geometryFactory.createShape("square")); // Creating another square for demonstration

for(Shape s: shapes){

s.draw();

} }}

class GeometryFactory {

public Shape createShape(String type) {

if (type.equalsIgnoreCase("circle")) {

return new Circle();

} else if (type.equalsIgnoreCase("square")) {

return new Square();

} else if (type.equalsIgnoreCase("triangle")) {

return new Triangle();

}

return null;

}}

**Instead of four classifiers we have implemented the factory method with a single method:**

package pk.cui.sc.fm;

import java.util.ArrayList;

public class Demo {

public static void main(String[] args) {

ArrayList<Shape> shapes = new ArrayList<Shape>();

GeometryFactory geometryFactory = new GeometryFactory();

shapes.add(geometryFactory.createShape("circle"));

shapes.add(geometryFactory.createShape("square"));

shapes.add(geometryFactory.createShape("square")); // Creating another square for demonstration

for(Shape s: shapes){

s.draw();

}

}}

class GeometryFactory {

public Shape createShape(String type) {

if (type.equalsIgnoreCase("circle")) {

return new Circle();

} else if (type.equalsIgnoreCase("square")) {

return new Square();

} else if (type.equalsIgnoreCase("triangle")) {

return new Triangle();

}

return null;

}}

C# Code to Java:

// Empty vocabulary of actual object

interface IPerson {

String getName();

}

class Villager implements IPerson {

public String getName() {

return "Village Person";

}

}

class CityPerson implements IPerson {

public String getName() {

return "City Person";

}

}

enum PersonType {

Rural,

Urban

}

// Implementation of Factory - Used to create objects.

public class PersonFactory {

public IPerson getPerson(PersonType type) {

switch (type) {

case Rural:

return new Villager();

case Urban:

return new CityPerson();

default:

throw new UnsupportedOperationException();

}}}

Fcatory:

public interface IProduct {

String getName();

boolean setPrice(double price);

}

class Phone implements IProduct {

private double price;

public String getName() {

return "Apple TouchPad";

}

public boolean setPrice(double price) {

this.price = price;

return true;

}

}

/\* Almost same as Factory, just an additional exposure to do something with the created method \*/

abstract class ProductAbstractFactory {

protected abstract IProduct makeProduct();

public IProduct getObject() { // Implementation of Factory Method.

return this.makeProduct();

}

}

class PhoneConcreteFactory extends ProductAbstractFactory {

protected IProduct makeProduct() {

IProduct product = new Phone();

// Do something with the object after you get the object.

product.setPrice(20.30);

return product;

}

}

**Example Implementation:**

// Product interface

interface Button {

void render();

}

// Concrete products

class WindowsButton implements Button {

public void render() {

System.out.println("Rendering a Windows button.");

}

}

class MacOSButton implements Button {

public void render() {

System.out.println("Rendering a MacOS button.");

}

}

// Creator interface

interface Dialog {

Button createButton();

void renderWindow();

}

// Concrete creators

class WindowsDialog implements Dialog {

public Button createButton() {

return new WindowsButton();

}

public void renderWindow() {

Button button = createButton();

button.render();

}

}

class MacOSDialog implements Dialog {

public Button createButton() {

return new MacOSButton();

}

public void renderWindow() {

Button button = createButton();

button.render();

}

}

// Client code

public class Main {

public static void main(String[] args) {

Dialog windowsDialog = new WindowsDialog();

windowsDialog.renderWindow();

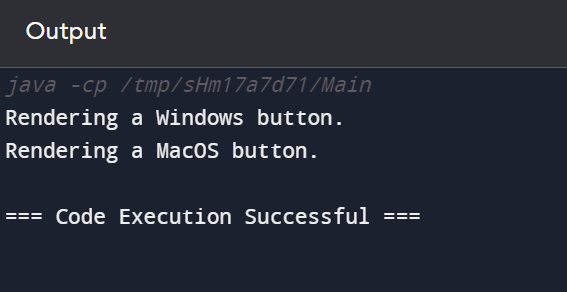
Dialog macOSDialog = new MacOSDialog();

macOSDialog.renderWindow();

}

}

**OUTPUT:**

****