

- # Factory Design Pattern
- abstract factory pattern
 - factory method pattern

OODP 2022

https://www.tutorialspoint.com/design_pattern

- creational pattern
- one of the best ways to create an object.
- create objects without exposing the creation logic to the client
- refer to newly created object using a common interface.

https://www.tutorialspoint.com/design_pattern

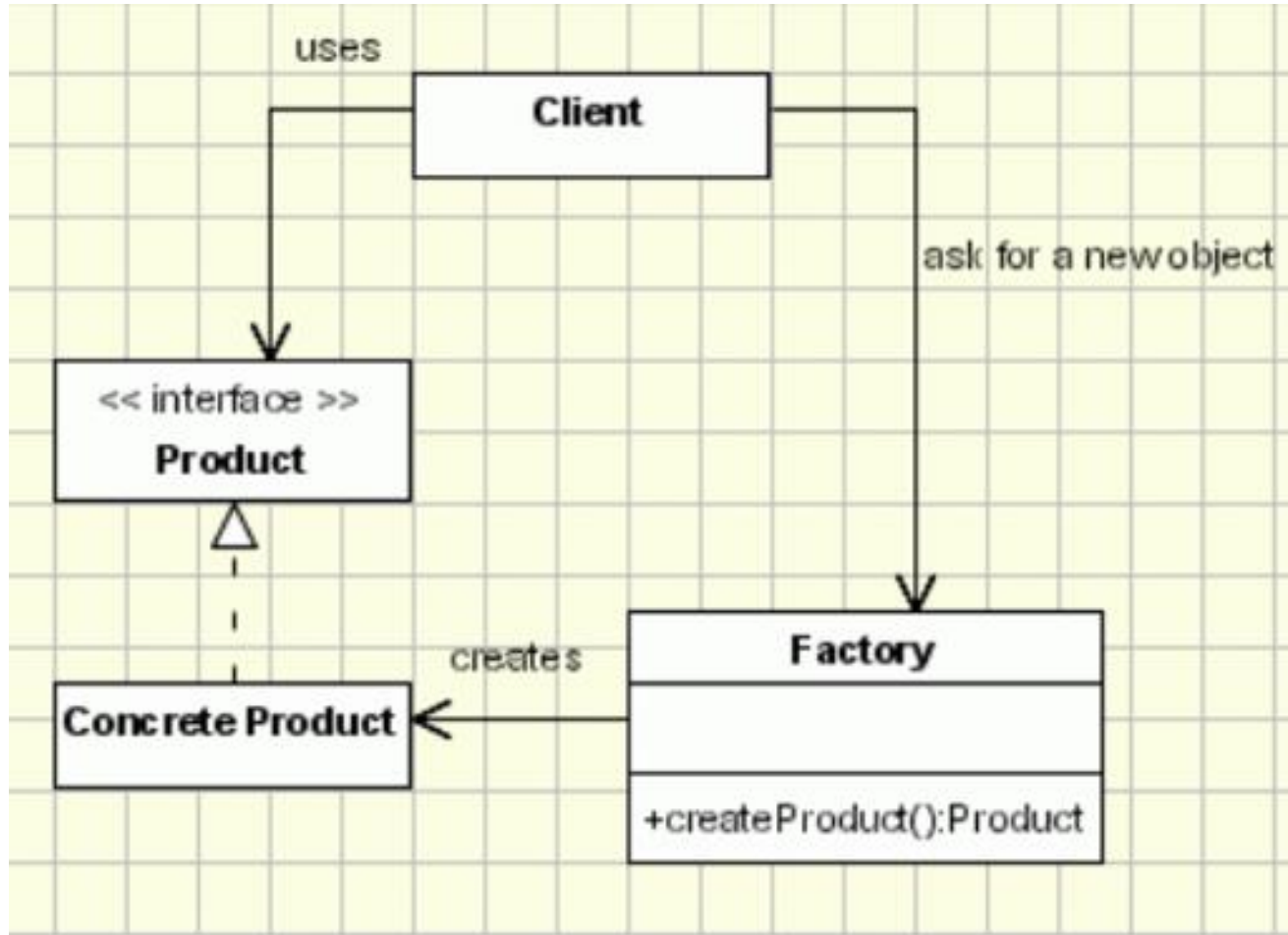
Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under **creational** pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create objects without exposing the creation logic to the client and refer to newly created object using a common interface.

Abstract factory pattern, which provides an interface for creating **related** or dependent objects without specifying the objects' concrete classes,...

Factory method pattern, which allows a class to defer instantiation to subclasses.

Factory Pattern



```
class Person {  
    public String name;  
    private String gender;  
    public String getName() {  
        return name;  
    }  
    public String getGender() {  
        return gender;  
    }  
}
```

```
public class Female extends Person {  
    public Female(String fullNname) {  
        System.out.println("Hello Ms. "+fullNname);  
    }  
}
```

```
public class Male extends Person {  
    public Male(String fullName) {  
        System.out.println("Hello Mr. "+fullName);  
    }  
}
```

```

public class SalutationFactory {
    public static void main(String args[]) {
        SalutationFactory factory = new SalutationFactory();
        factory.getPerson(args[0], args[1]);
    }
    public Person getPerson(String name, String gender) {
        if (gender.equals("M"))
            return new Male(name);
        else if (gender.equals("F"))
            return new Female(name);
        else
            return null;
    }
}

```

```

I:\001 OODP Spr2010\designPatternExampleCode\factoryPattern\MrMrs>java SalutationFactory peter M
Hello Mr. peter

I:\001 OODP Spr2010\designPatternExampleCode\factoryPattern\MrMrs>java SalutationFactory peter F
Hello Ms. peter

```

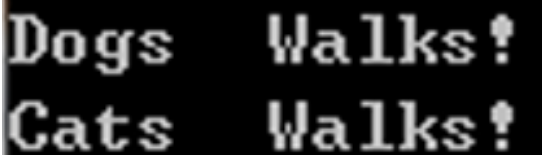
```
public abstract class Mammals {  
    public abstract String doWalking();  
}
```

```
public class Dog extends Mammals {  
    public String doWalking() {  
        return "Dogs  Walks!";  
    }  
}
```

```
public class Cat extends Mammals {  
    public String doWalking() {  
        return "Cats  Walks! ";  
    }  
}
```

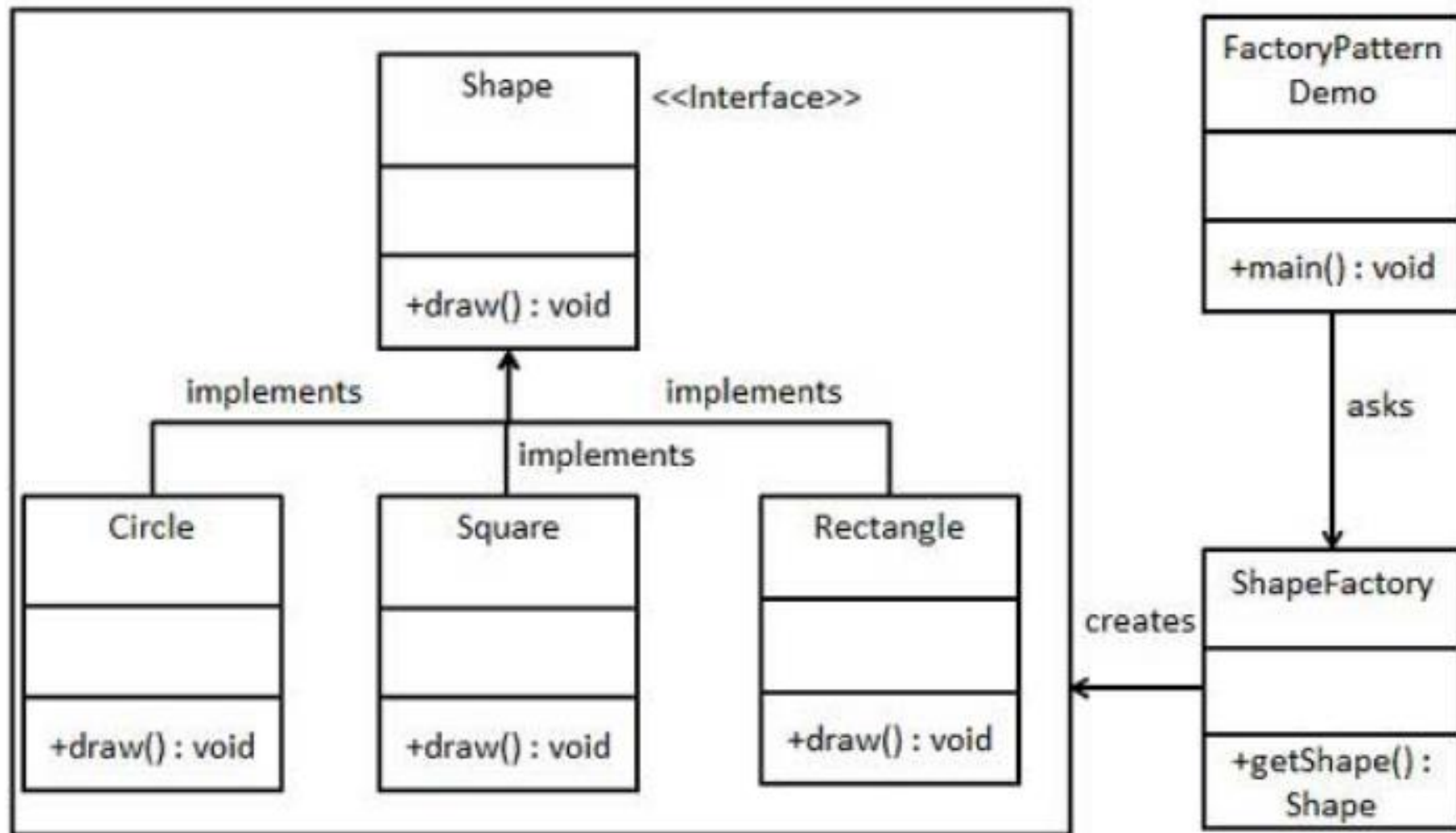
```
public class MammalsFactory {  
    public static Mammals  
        getMammalObject(String name) {  
        if (name.equalsIgnoreCase("Cat")){  
            return new Cat();  
        } else {  
            return new Dog();  
        }  
    }  
}
```

```
public class FactoryClient {  
    public static void main(String args[]) {  
        MammalsFactory mf = new MammalsFactory();  
        System.out.println(mf.getMammalObject("Dog").doWalking());  
        System.out.println(mf.getMammalObject("Cat").doWalking());  
    }  
}
```



```
Dogs  Walks!  
Cats  Walks!
```


Compare two factory examples.



Example A

```
public interface Shape {  
    void draw();  
}
```

```
public class Circle implements Shape {  
  
    @Override  
    public void draw() {  
        System.out.println("Inside Circle::draw() method.");  
    }  
}
```

```
public class Rectangle implements Shape {  
  
    @Override  
    public void draw() {  
        System.out.println("Inside Rectangle::draw()  
method.");  
    }  
}
```

```
public class Square implements Shape {  
  
    @Override  
    public void draw() {  
        System.out.println("Inside Square::draw() method.");  
    }  
}
```

```
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;  
    }  
}
```

```
public class FactoryPatternDemo {
```

FactoryShape.zip

```
    public static void main(String[] args) {  
        ShapeFactory shapeFactory = new ShapeFactory();
```

```
        //get an object of Circle and call its draw method.  
        Shape shape1 = shapeFactory.getShape("CIRCLE");
```

```
        //call draw method of Circle  
        shape1.draw();
```

```
        //get an object of Rectangle and call its draw method.  
        Shape shape2 = shapeFactory.getShape("RECTANGLE");
```

```
        //call draw method of Rectangle  
        shape2.draw();
```

```
        //get an object of Square and call its draw method.  
        Shape shape3 = shapeFactory.getShape("SQUARE");
```

```
        //call draw method of square  
        shape3.draw();
```

```
    }  
}
```

```
Inside Circle::draw() method.  
Inside Rectangle::draw() method.  
Inside Square::draw() method.
```

example B

```
public class AbstractFactoryPatternDemo {  
    public static void main(String[] args) {  
        AbstractFactory shapeFactory =  
            FactoryProducer.getFactory("SHAPE");  
        Shape shape1 = shapeFactory.getShape("CIRCLE");  
        shape1.draw();  
        Shape shape2 = shapeFactory.getShape("RECTANGLE");  
        shape2.draw();  
        Shape shape3 = shapeFactory.getShape("SQUARE");  
        shape3.draw();  
        AbstractFactory colorFactory =  
            FactoryProducer.getFactory("COLOR");  
        Color color1 = colorFactory.getColor("RED");  
        ....  
    }  
}
```

Inside Circle::draw() method.
Inside Rectangle::draw() method.
Inside Square::draw() method.
Inside Red::fill() method.
Inside Green::fill() method.
Inside Blue::fill() method.

AbstractFactoryShapeColor_2022.zip

```
public abstract class AbstractFactory {  
    abstract Color getColor(String color);  
    abstract Shape getShape(String shape) ;  
}
```

```
public interface Color {  
    void fill();  
}
```

```
public interface Shape {  
    void draw();  
}
```

```
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;  
    }  
}
```



```

public class ShapeFactory extends AbstractFactory {
    @Override
    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;
    }
    @Override
    Color getColor(String color) {
        return null;
    }
}

```

```

public abstract class AbstractFactory {
    abstract Color getColor(String color);
    abstract Shape getShape(String shape) ;
}

```

```
public class ColorFactory extends AbstractFactory {  
    @Override  
    public Shape getShape(String shapeType){  
        return null;  
    }  
    @Override  
    Color getColor(String color) {  
        if(color == null){  
            return null;  
        }  
        if(color.equalsIgnoreCase("RED")){  
            return new Red();  
        } else if(color.equalsIgnoreCase("GREEN")){  
            return new Green();  
        } else if(color.equalsIgnoreCase("BLUE")){  
            return new Blue();  
        }  
        return null;  
    }  
}
```