

# Factory Method

Lecture 17

# Factory Method

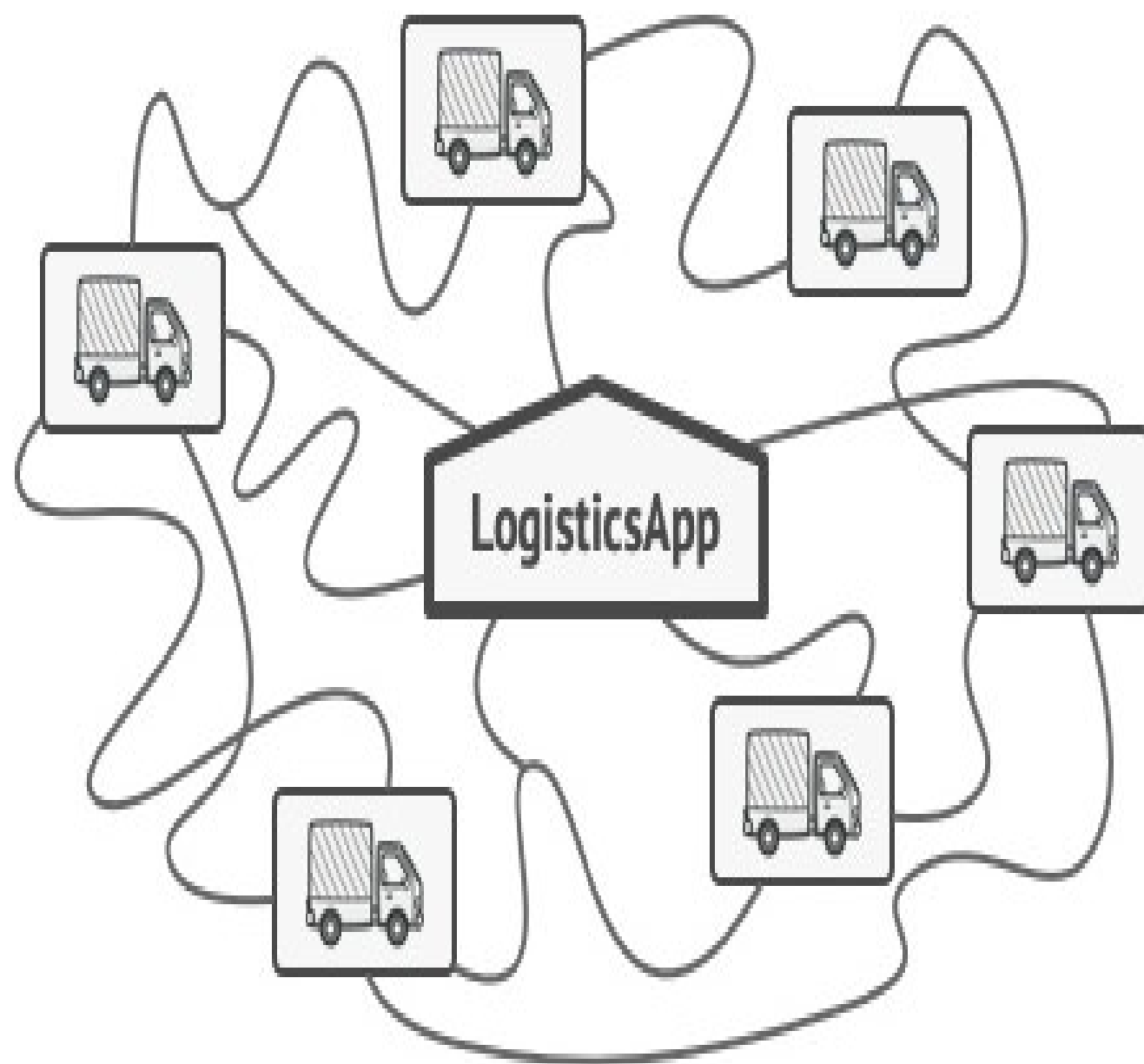
- Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.
- In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

# Intent

- **Factory Method** is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

# Problem

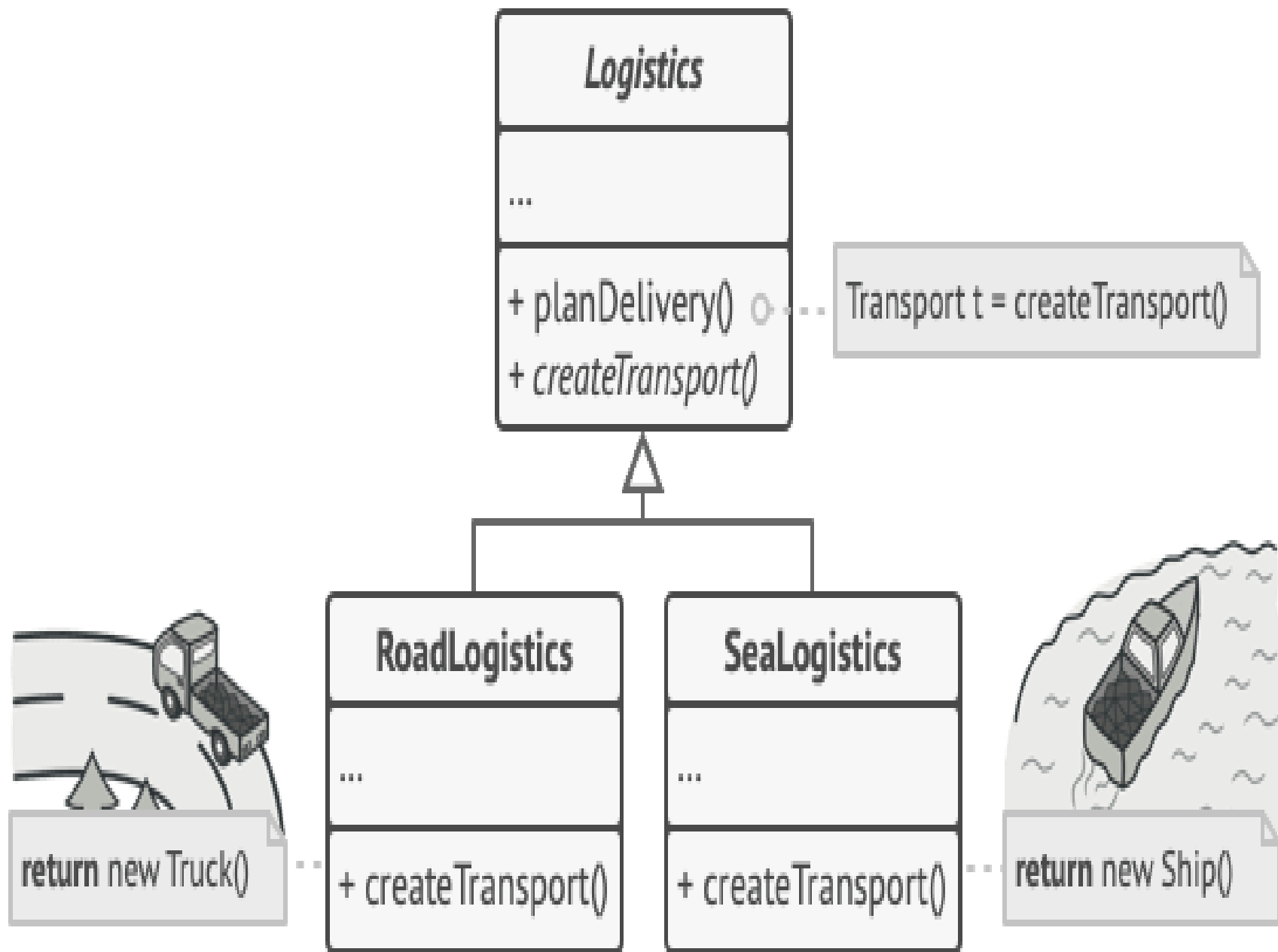
- Imagine that you're creating a logistics management application. The first version of your app can only handle transportation by trucks, so the bulk of your code lives inside the Truck class.
- After a while, your app becomes pretty popular. Each day you receive dozens of requests from sea transportation companies to incorporate sea logistics into the app.
-



- At present, most of your code is coupled to the Truck class.
- Adding Ships into the app would require making changes to the entire codebase. Moreover, if later you decide to add another type of transportation to the app, you will probably need to make all of these changes again.

# Solution

- The Factory Method pattern suggests that you replace direct object construction calls (using the new operator) with calls to a special *factory* method.
- The objects are still created via the new operator, but it's being called from within the factory method.
- Objects returned by a factory method are often referred to as *products*.



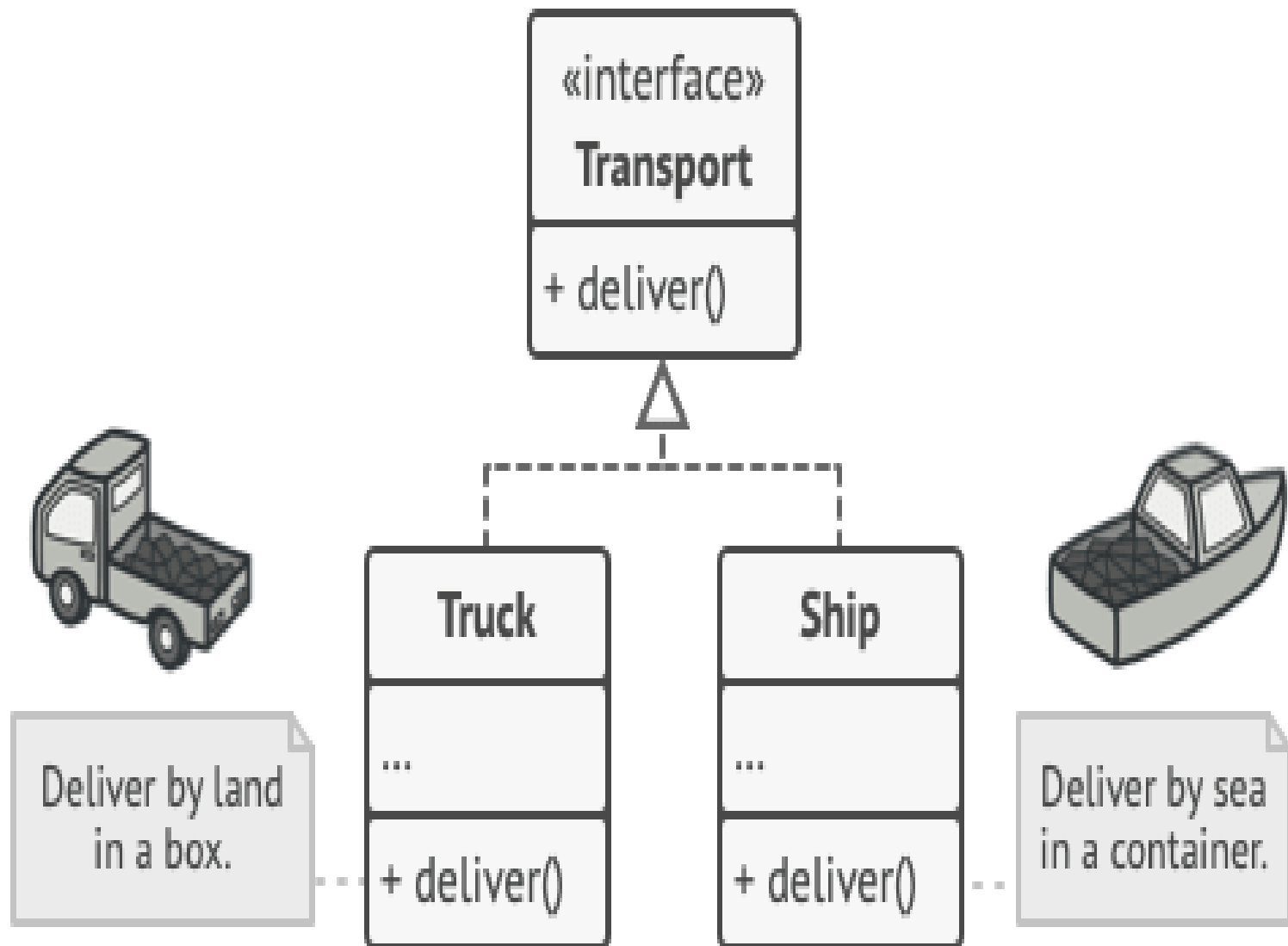


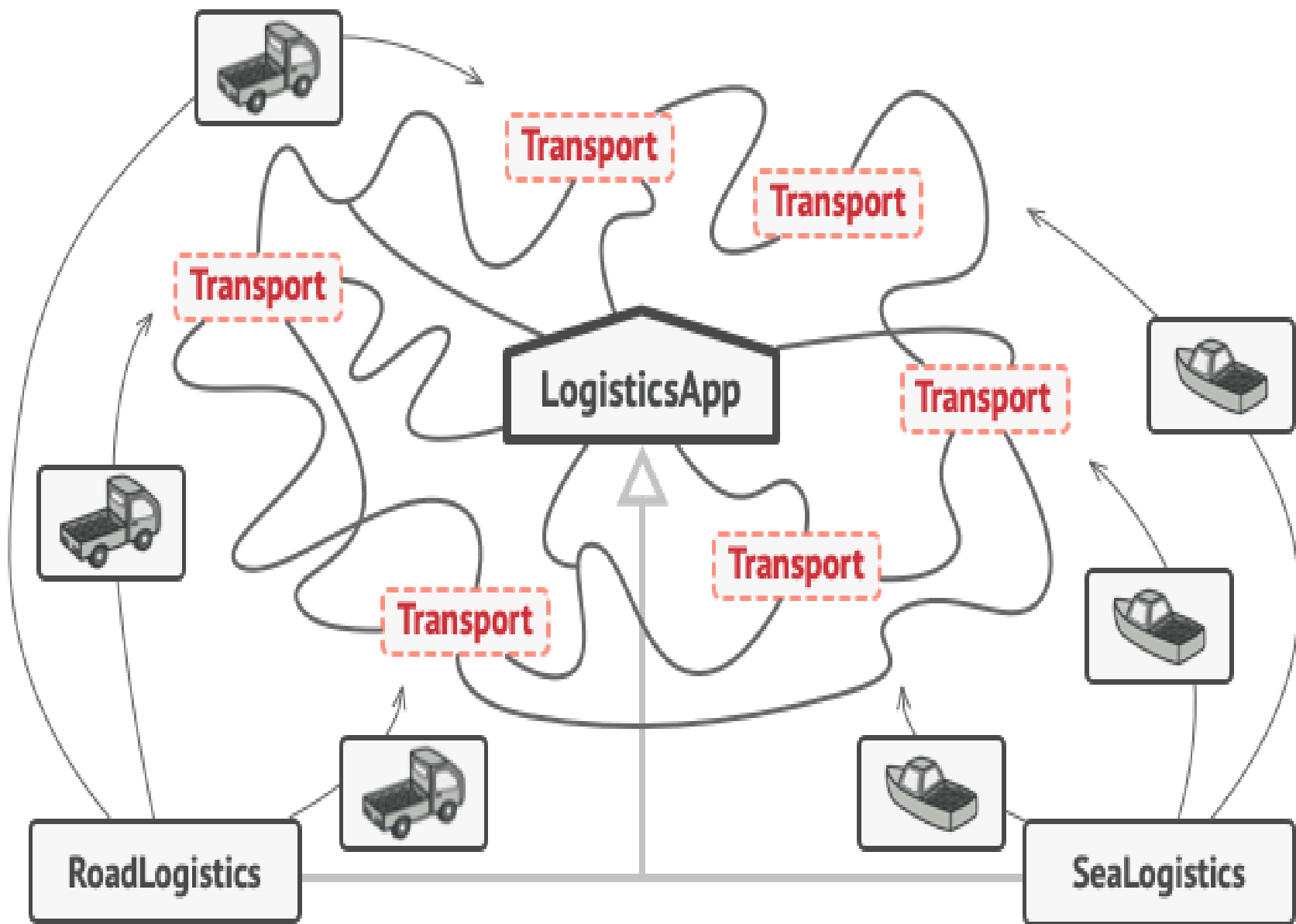
# Advantages

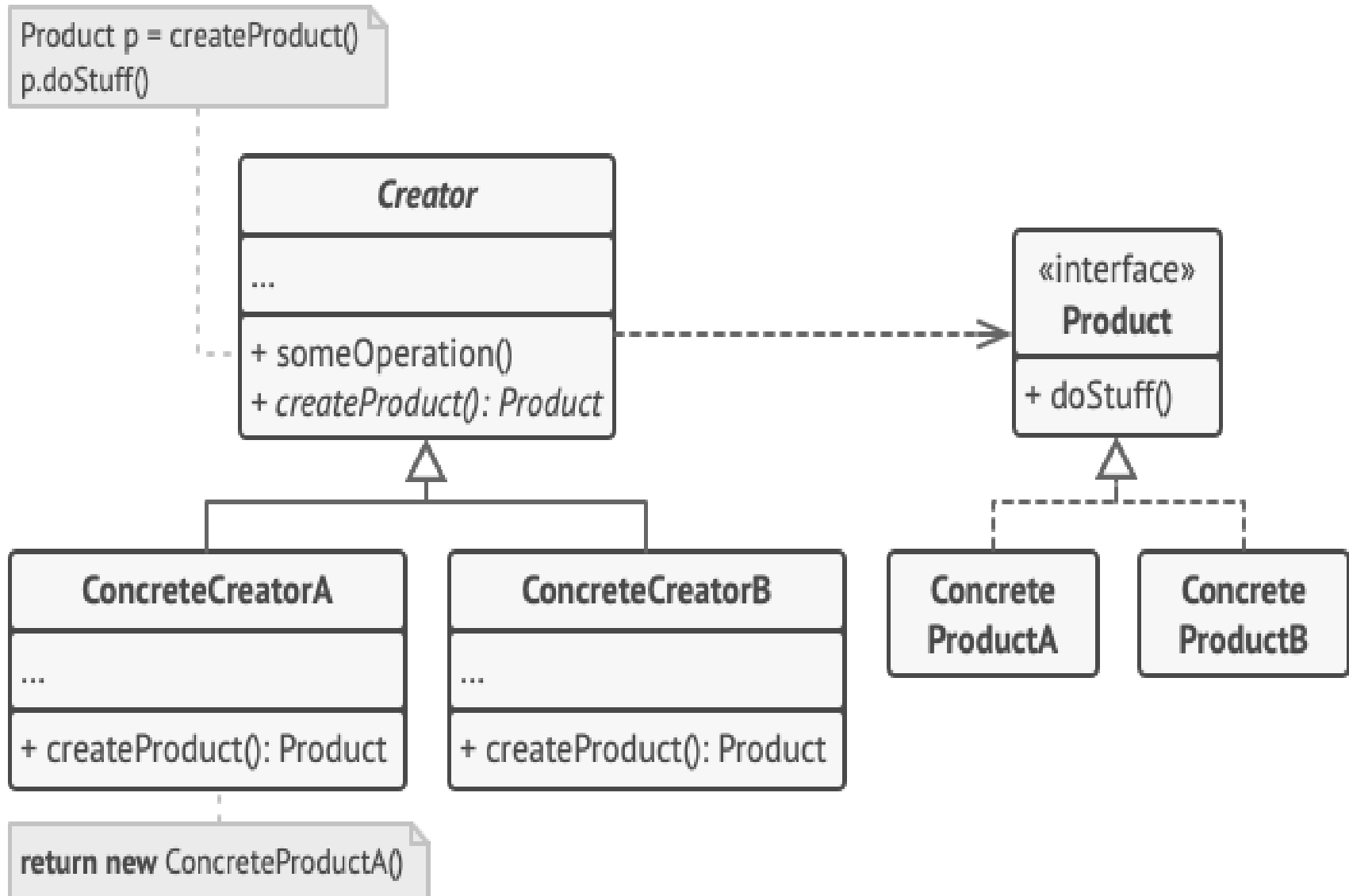
- At first glance, this change may look pointless: we just moved the constructor call from one part of the program to another. However, consider this: now you can override the factory method in a subclass and change the class of products being created by the method.

# Limitation

- There's a slight limitation though: subclasses may return different types of products only if these products have a common base class or interface. Also, the factory method in the base class should have its return type declared as this interface.

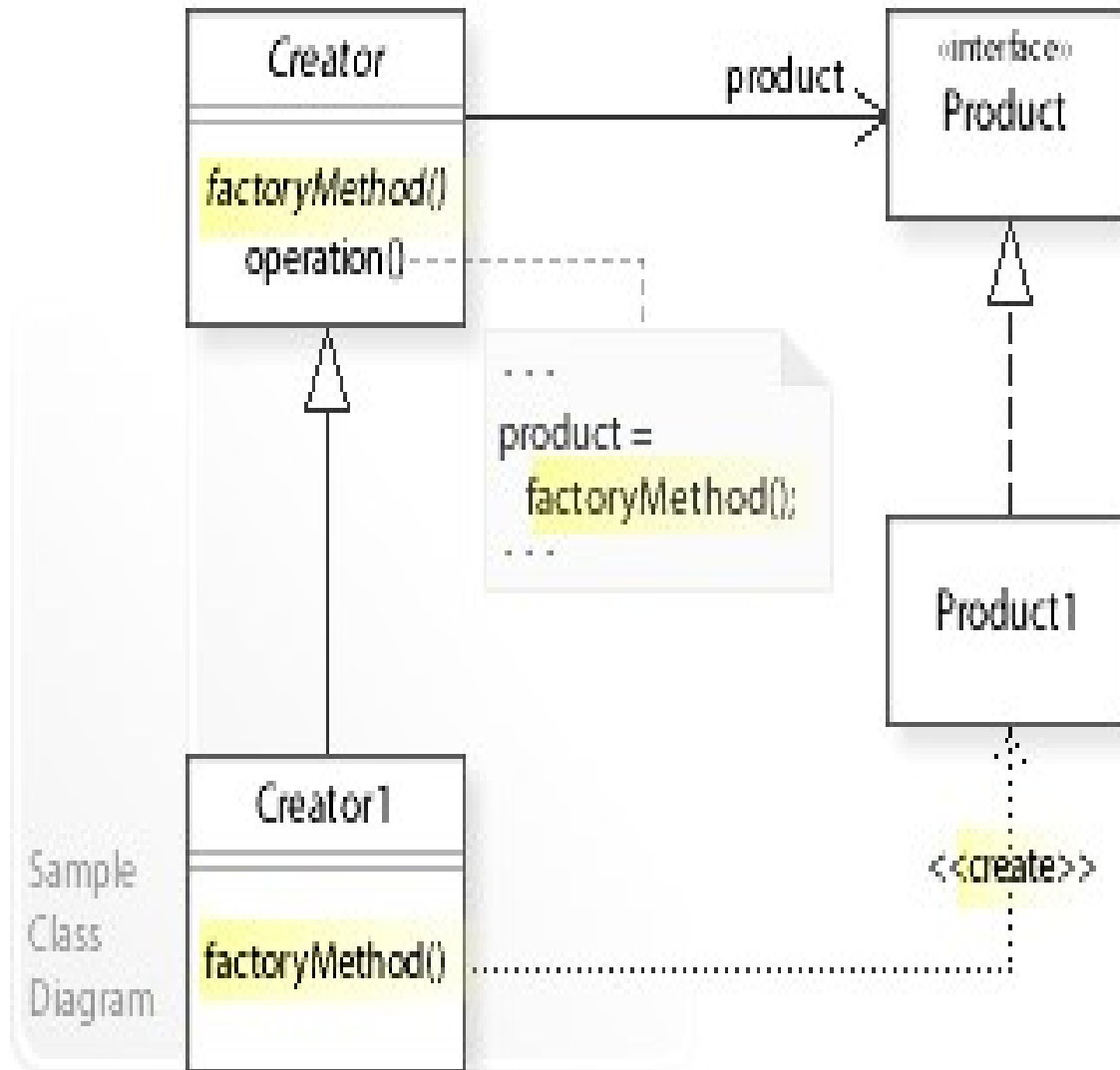


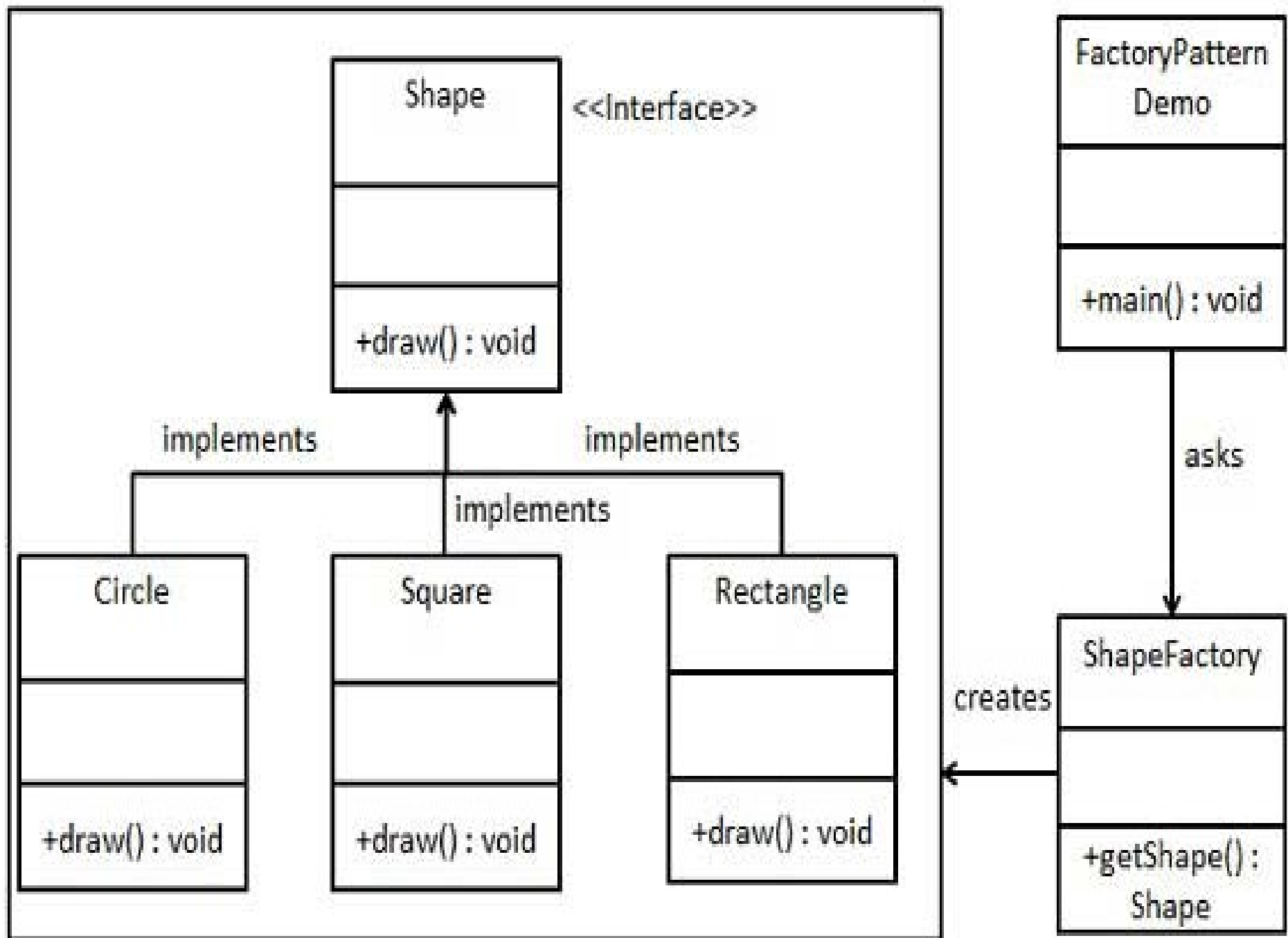




# Example

- *FactoryPatternDemo*, our demo class will use *ShapeFactory* to get a *Shape* object. It will pass information (*CIRCLE / RECTANGLE / SQUARE*) to *ShapeFactory* to get the type of object it needs.







## Step 1

Create an interface.

*Shape.java*

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

## Step 2

Create concrete classes implementing the same interface.

*Rectangle.java*

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

### *Square.java*

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

### *Circle.java*

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

## ShapeFactory.java

```
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 {  
  
    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();  
    }  
}
```

# References

- <https://refactoring.guru/design-patterns/factory-method>
- [https://en.wikipedia.org/wiki/Factory method pattern](https://en.wikipedia.org/wiki/Factory_method_pattern)
- [https://www.tutorialspoint.com/design pattern/factory pattern.htm](https://www.tutorialspoint.com/design_pattern/factory_pattern.htm)