# SD Factory Method Report.

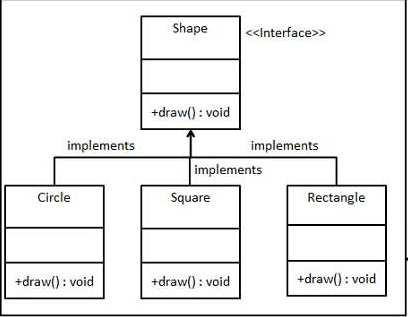
## 1. Intent of Factory Method.

Factory method intents to solve the problem of creating objects without having to specify the exact class of the object that will be created, with this we have separation of concerns, that is, a class will have the responsibility of the creation of objects.

* Pros: helps to make your application easy to understand, helps improving low coupling and high cohesion.
* Cons: can cause troubles when it is incorrectly used, for example try to wrap up all the application in a container instead of using dependency injection.

## 2. Explanation.

Let us consider the following example: we have a shape interface that defines different kinds of figures, for example circles, squares and rectangles. This classes will have a simple method draw which will just print the figure that will be drawn.



\*Note: we are using java programing language for the following example.

public class Rectangle implements Shape {

@Override

public void draw() {

System.out.println("Inside Rectangle::draw() method.");

}

}

Problem: How can we create this objects without exposing the creation logic?

Solution: Delegate the behaviour of creating the objects to a class ShapeFactory, that is, apply the factory method design pattern.

This class will generate the different shapes based on users input:

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;

}

}

Now the way of creating the object for example in a main.java class will be the following:

//get an object of Rectangle and call its draw method.

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

//call draw method of Rectangle

shape2.draw();

Follows the following SOLID principles:

Single Responsibility: the class which will handle the behaviour of creation is the factory one.

Liskov substitution principle: classes created with the factory as shown in the main example, if we call the draw method the application know that is a rectangle.

Interface segregation principle: ShapeFactory hides the implementation details of our class rectangle.

## 3. Related Patterns.

Factory method is related with the Builder design pattern, both belong to the creational patterns family and the differences are the following:

* Factory method pattern requires the entire object to be built in a single method call, with all the parameters pass in on a single line and then the final object will be created using this parameters and returned.
* Builder allow you to create the object using setters so you can build your class and then the *build()* method returns the created object.

Examples:

**Factory**

// Factory

static class FruitFactory {

static Fruit create(name, color, firmness) {

// Additional logic

return new Fruit(name, color, firmness);

}

}

// Usage

Fruit fruit = FruitFactory.create("apple", "red", "crunchy");

**Builder**

// Builder

class FruitBuilder {

String name, color, firmness;

FruitBuilder setName(name) { this.name = name; return this; }

FruitBuilder setColor(color) { this.color = color; return this; }

FruitBuilder setFirmness(firmness) { this.firmness = firmness; return this; }

Fruit build() {

return new Fruit(this); // Pass in the builder

}

}

// Usage

Fruit fruit = new FruitBuilder()

.setName("apple")

.setColor("red")

.setFirmness("crunchy")

.build();

## 4. Common situations of use.

* From the .NET **Base Class Library** (BCL) :

<https://msdn.microsoft.com/en-us/library/system.windows.forms.control.createcontrolsinstance.aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-1>

Control.CreateControlsInstance can be override to create your own controls.

## 5. Can be mistaken with the Builder design pattern.

Differences explained in point 3.

## 6. References.

<https://sourcemaking.com/design_patterns/factory_method>

<https://www.quora.com/What-are-the-pros-and-cons-of-the-factory-design-pattern>

<https://en.wikipedia.org/wiki/Factory_method_pattern>