Pola Desain Perangkat Lunak

[Week] 5 – Creational Pattern, Factory Method & Abstract Factory
Prepared by: Tifanny Nabarian

Design Patterns Category

Creational Patterns	Creational patterns prescribe the way that objects are created.
Structural Patterns	• Structural patterns are concerned with how classes and objects are composed to form larger structures
Behavioral Patterns	• Behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects.
Concurrency Patterns	• Concurrency patterns prescribe the way access to shared resources is coordinated or sequenced

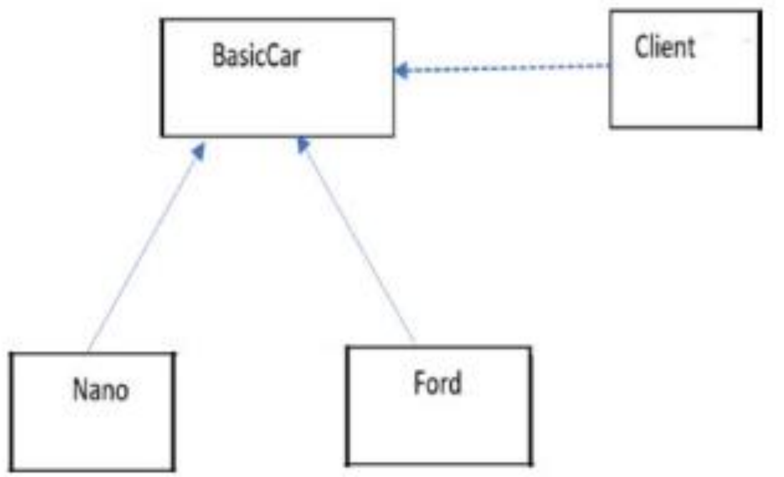
Design Patterns Scope

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	• Factory method	• Adapter	• Interpreter • Template method
	Object	• Abstract factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Fasad • Flyweight • Proxy	• Chain of responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

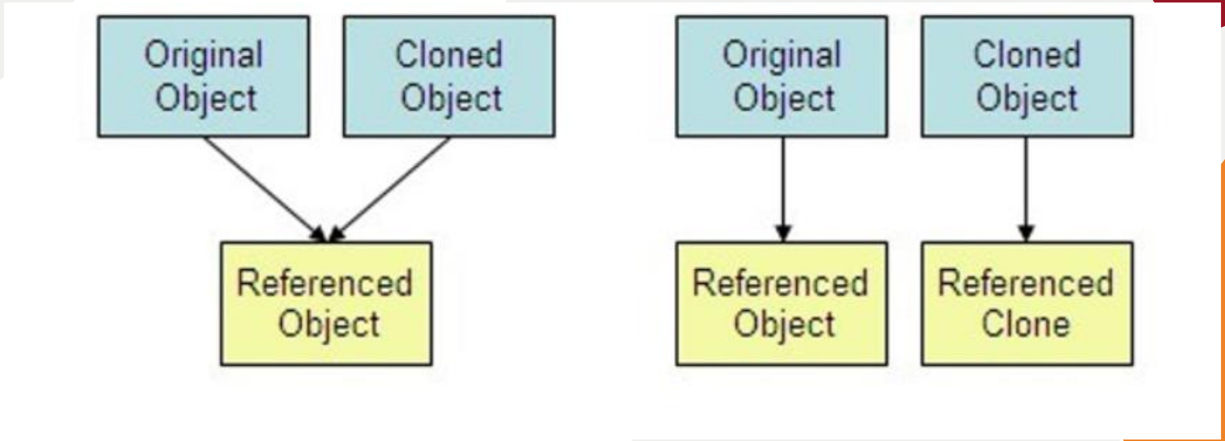
Creational Pattern

Review Prototype

Let's Play with the code!



Shallow Copy VS Deep Copy



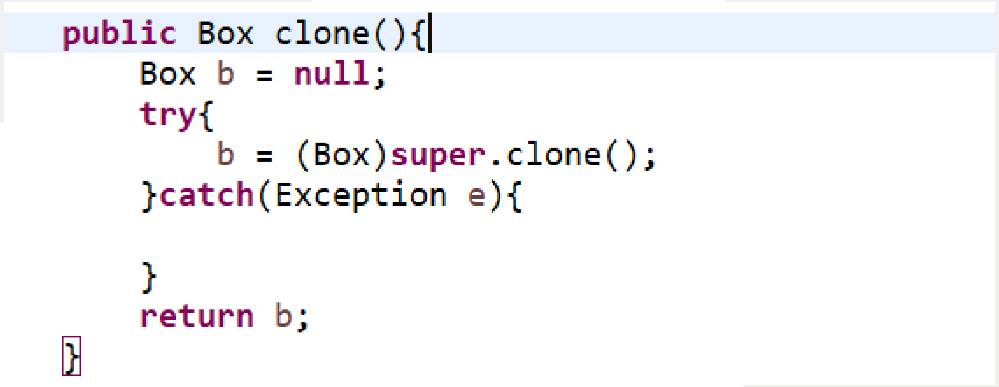
Shallow Copy

Deep Copy

Shallow Copy VS Deep Copy

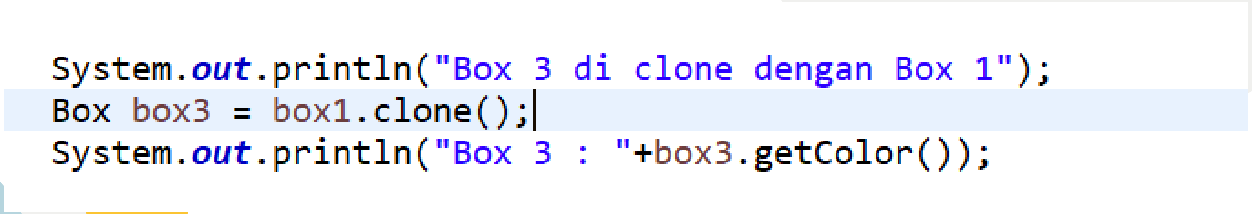
- A shallow copy creates a new object and then copies various field values from the original object to the new object.
- So, it is also known as a field-by-field copy.
- If the original object contains any references to other objects as fields, then the references of those objects are copied into the new object, (i.e., **you do not create the copies of those objects).**

Implementasi Shallow Copy



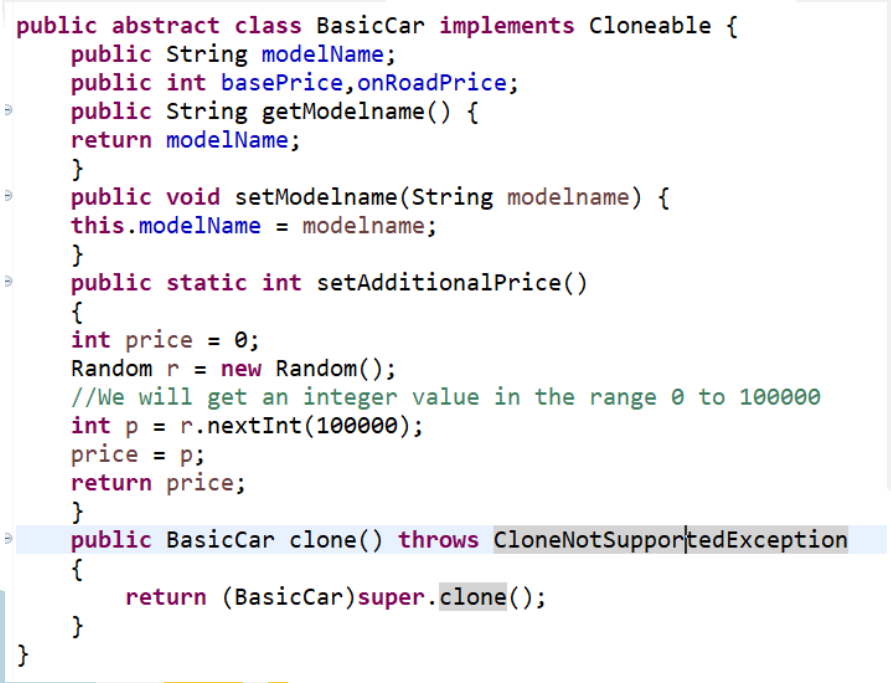
```
public Box clone(){
        Box b = null;
        try{
                b = (Box)super.clone();
        }catch(Exception e){

        }
        return b;
}
```

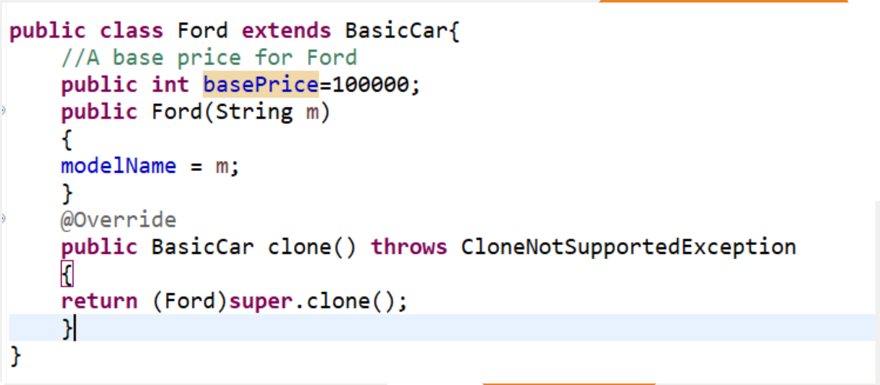


```
System.out.println("Box 3 di clone dengan Box 1");
Box box3 = box1.clone();
System.out.println("Box 3 : "+box3.getColor());
```

Implementasi Deep Copy



```
public abstract class BasicCar implements Cloneable {
    public String modelName;
    public int basePrice,onRoadPrice;
    public String getModelname() {
    return modelName;
    }
    public void setModelname(String modelname) {
    this.modelName = modelname;
    }
    public static int setAdditionalPrice()
    {
    int price = 0;
    Random r = new Random();
    //We will get an integer value in the range 0 to 100000
    int p = r.nextInt(100000);
    price = p;
    return price;
    }
    public BasicCar clone() throws CloneNotSupportedException
    {
        return (BasicCar)super.clone();
    }
}
```



```
public class Ford extends BasicCar{
    //A base price for Ford
    public int basePrice=100000;
    public Ford(String m)
    {
    modelName = m;
    }
    @Override
    public BasicCar clone() throws CloneNotSupportedException
    {
    return (Ford)super.clone();
    }
}
```

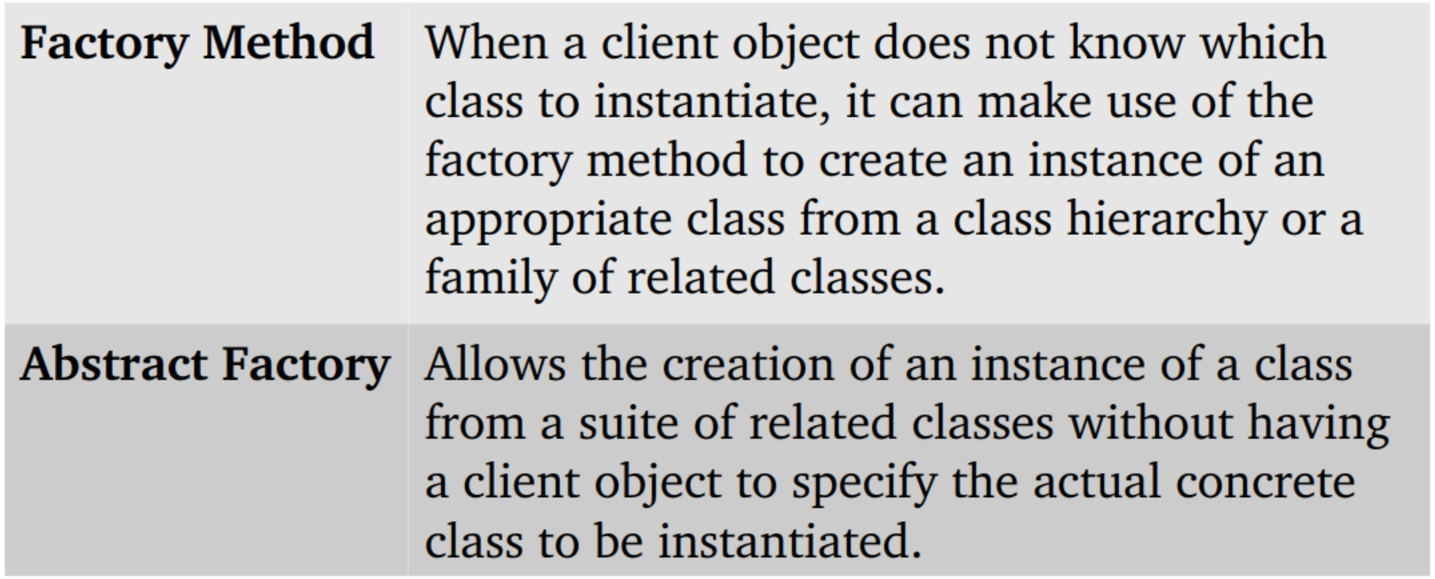
When do you choose a shallow copy over a deep copy?

- ✓ A **shallow copy** is faster and less expensive. It is always better if your target object has the **primitive fields** only.
- ✓ A **deep copy** is expensive and slow. But it is useful if your target object contains many fields that have **references to other objects**.

Creational Pattern

Factory Method vs Abstract Factory

Topic



Factory Method	When a client object does not know which class to instantiate, it can make use of the factory method to create an instance of an appropriate class from a class hierarchy or a family of related classes.
Abstract Factory	Allows the creation of an instance of a class from a suite of related classes without having a client object to specify the actual concrete class to be instantiated.

Factory Method

Factory Method

- In general, all subclasses in a class hierarchy inherit the methods implemented by the parent class.
- A subclass may override the parent class implementation to offer a different type of functionality for the same method.
- When an application object is aware of the exact functionality it needs, it can directly instantiate the class from the class hierarchy that offers the required functionality.
- At times, an application object may only know that it needs to access a class from within the class hierarchy, but does not know exactly which class from among the set of subclasses of the parent class is to be selected.

Factory Method

- Intent:
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
  - Defining a "virtual" constructor.
  - The new operator considered harmful.
- Problem:

A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.

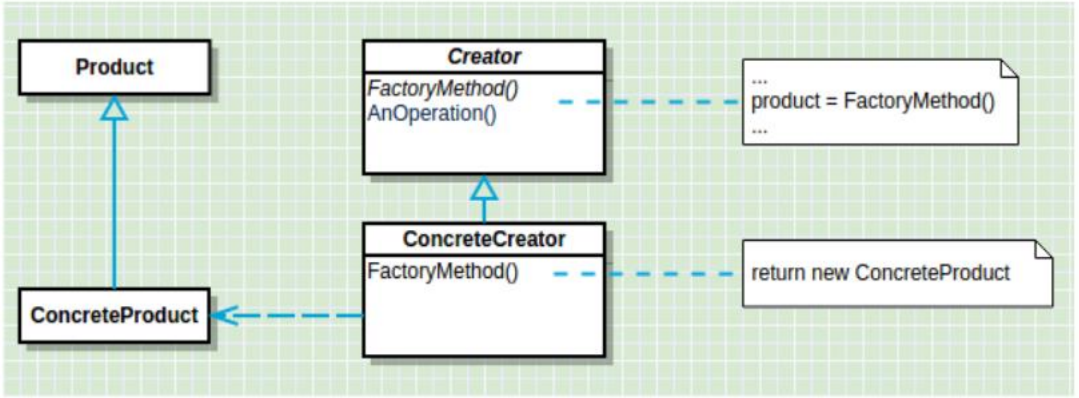
Factory Method

- An increasingly popular definition of factory method is: a static method of a class that returns an object of that class' type.
- But unlike a constructor, the actual object it returns might be an instance of a subclass.
- Unlike a constructor, an existing object might be reused, instead of a new object created.
- Unlike a constructor, factory methods can have different and more descriptive names e.g.

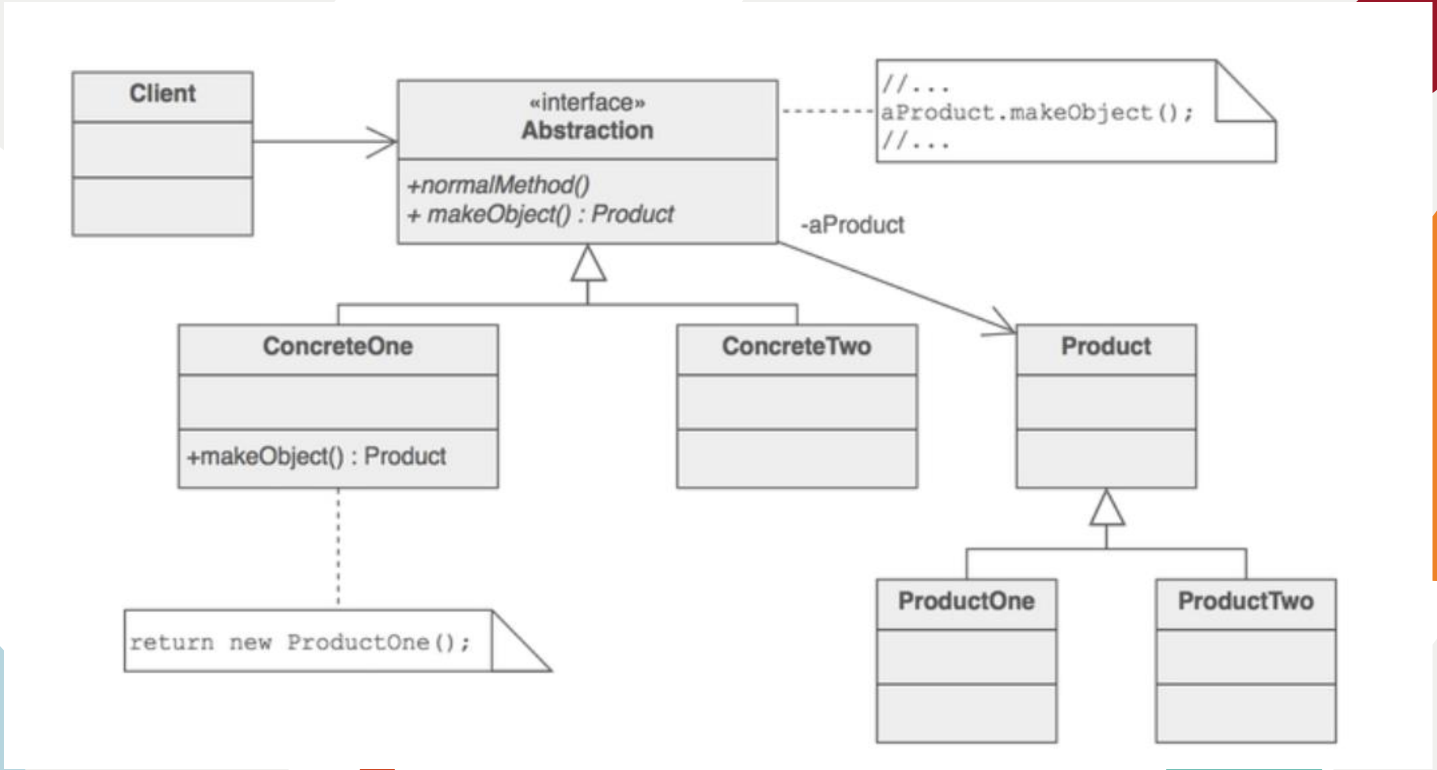
```
Color.make_RGB_color(float red, float green, float blue)
Color.make_HSB_color(float hue, float saturation, float brightness)
```

Factory Method

- Applicability - Use the Factory Method pattern in any of the following situations:
  - A class can't anticipate the class of objects it must create
  - A class wants its subclasses to specify the objects it creates



Factory Method

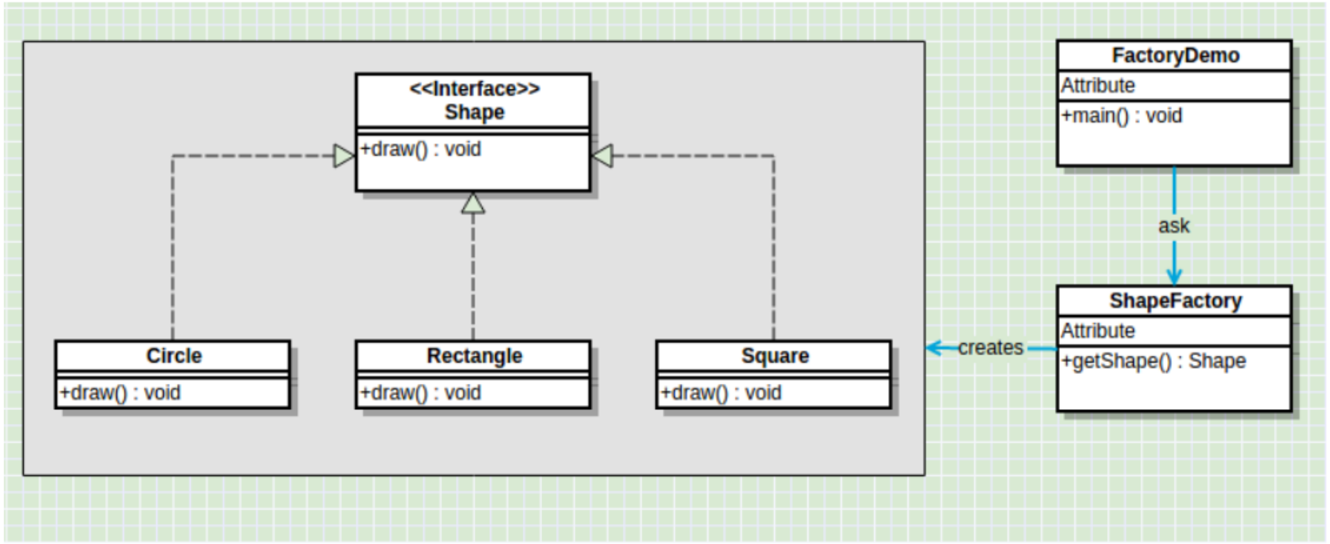


```
//...
aProduct.makeObject();
//...
```

```
return new ProductOne();
```

Factory Method

Example



Abstract Factory

Abstract Factory

- 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.

Abstract Factory

Intent:

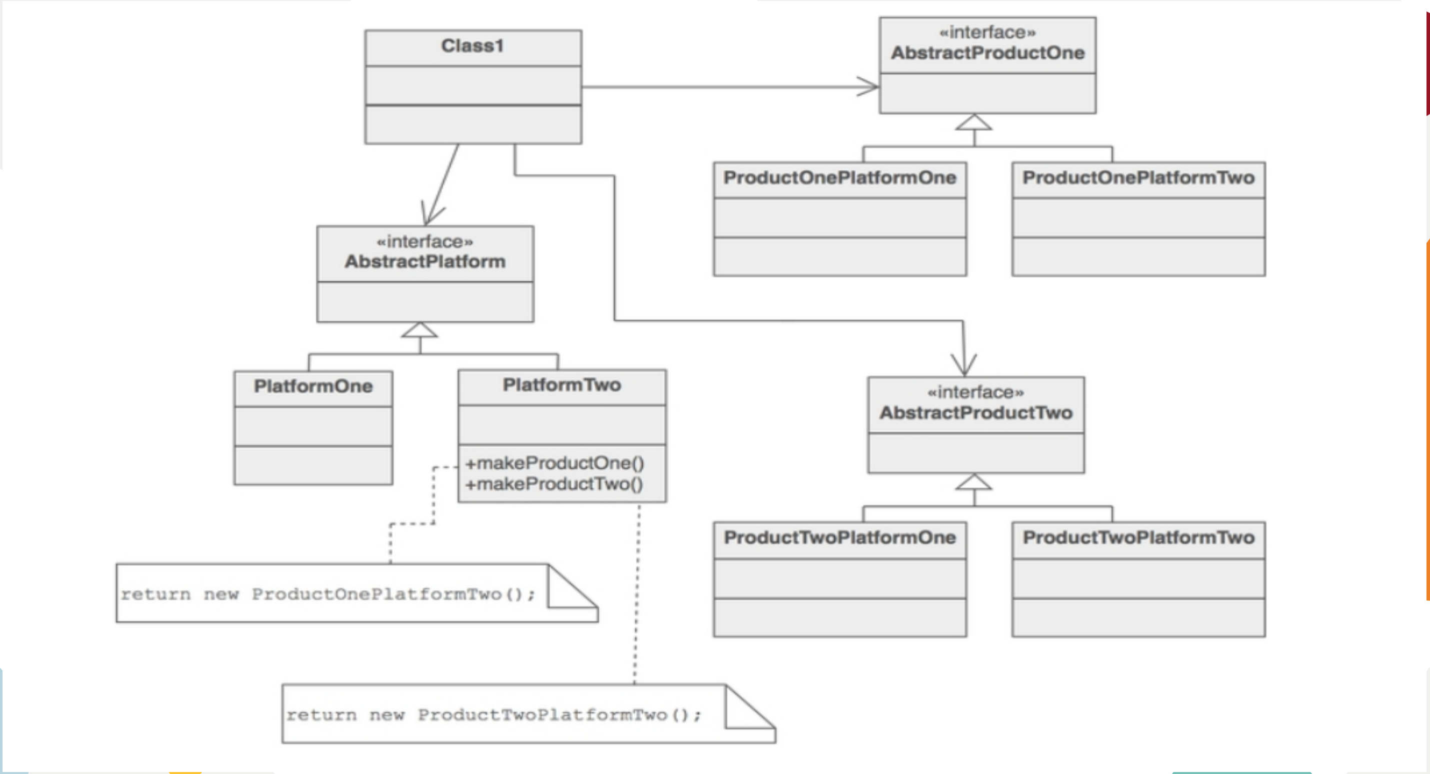
- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- A hierarchy that encapsulates: many possible "platforms", and the construction of a suite of "products".
- The new operator considered harmful.

Abstract Factory

Problem:

- If an application is to be portable, it needs to encapsulate platform dependencies.
- These "platforms" might include: windowing system, operating system, database, etc.
- Too often, this encapsulatation is not engineered in advance, and lots of #ifdef case statements with options for all currently supported platforms begin to procreate like rabbits throughout the code.

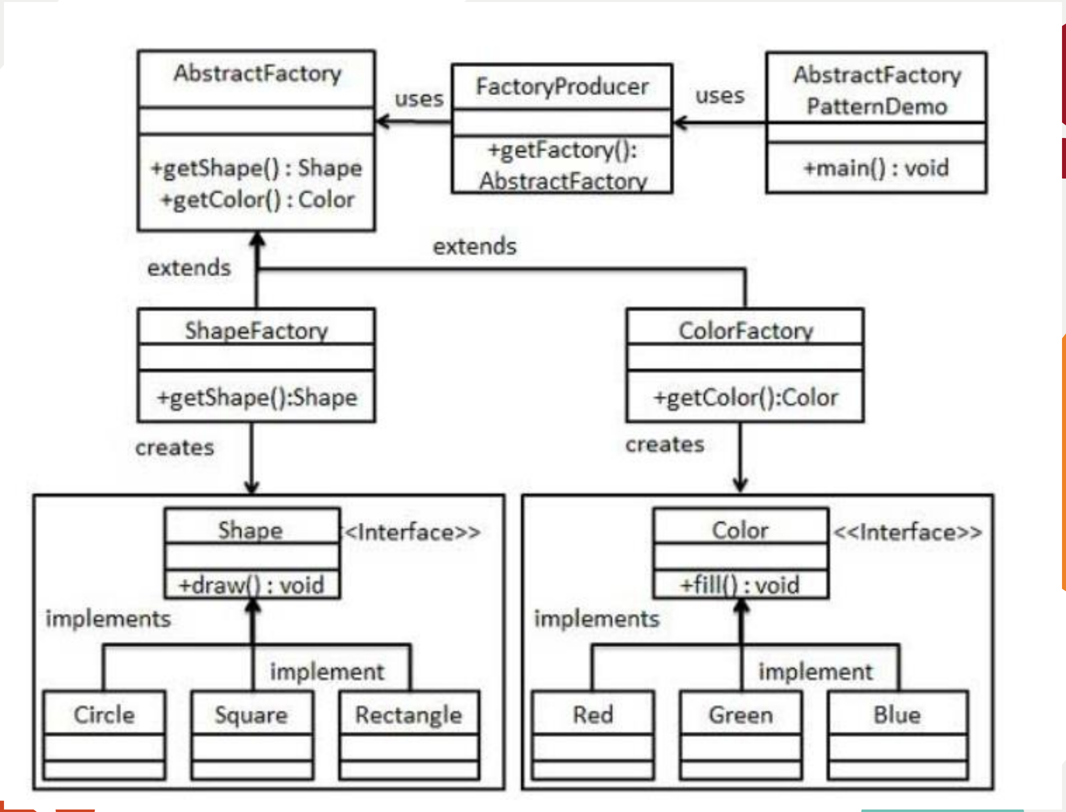
Abstract Factory



```
return new ProductOnePlatformTwo();
```

```
return new ProductTwoPlatformTwo();
```

Abstract Factory



Let's Practice!