Factory Method

By: Kirin Masood

Overview

Pattern Name	Factory Method
Other Names	Virtual Constructor
Classification	Class Creational
Intent	Defines a standard interface for object creation, Defers instantiation to subclass

Factory Method: Main Idea

Create objects without having to specify the exact class of the object that will be created.

This is done by calling a factory method—either specified in an interface or child class—rather than by calling a constructor.

Significance:
Use the factory pattern...

When you don't know what class object you need ahead of time

When all of the potential classes are in the same subclass hierarchy

To centralize class selection code

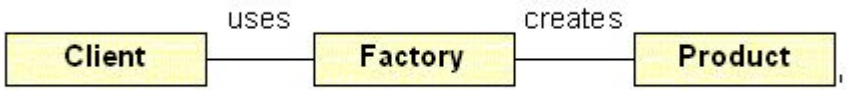
When you don't want the user to know every subclass

To encapsulate object creation

Significance: Solution

Solution:

- Factory Method!
- Classes can defer instantiation to subclasses at runtime



Main idea: Factory creates the new object

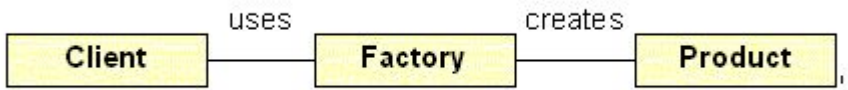
Logic

Factory
creates a new instance of the product

Client
Client requires an instance of Product

Does not create the product directly

Delegates this to Factory



Product
Created by factory

Passed back to client

Main idea: client uses the factory to create an instance of the product

Building the Diagram Definition of the Factory Method

Factory Method: Interface

Product: This is the interface of the objects the factory method creates

Ex: Dog Factory

Product: Dog

Method that must be implemented via the interface: speak();

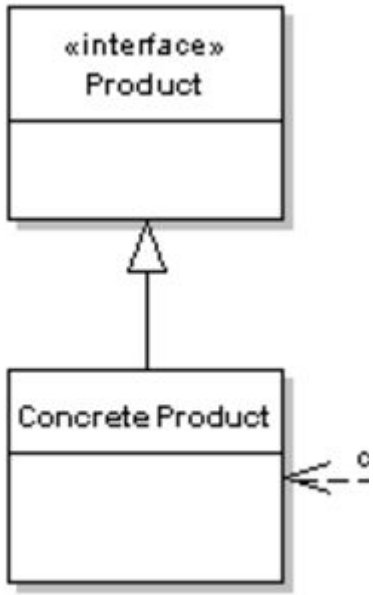
Factory Method: Concrete Product

Implements the product interface

Product: Dog

Concrete Products: Poodle, Dalmation, Beagle

Method that must be implemented:
speak();



Factory Method: Concrete Creator

Provides an interface for the creation of Products

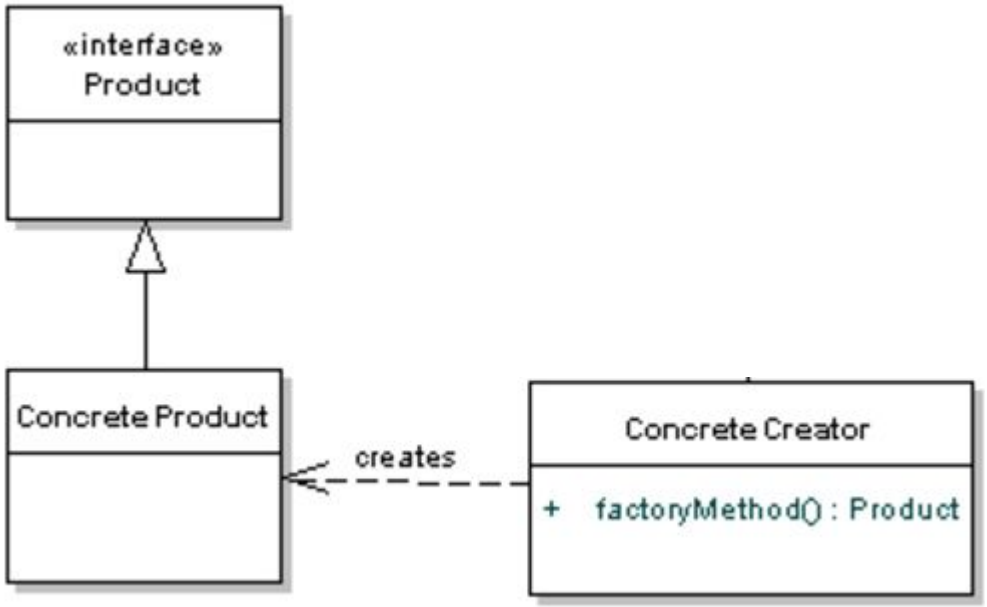
Ex: Dog Factory

Product: Dog

Concrete Product: Poodle, Dalmation, Beagle

Method that must be implemented: speak();

Concrete Creator: DogFactory that returns the Dog when the function is called i.e.
Return new Poodle();



Factory Method: Creator

Declares the factory method which returns an object of type Product.

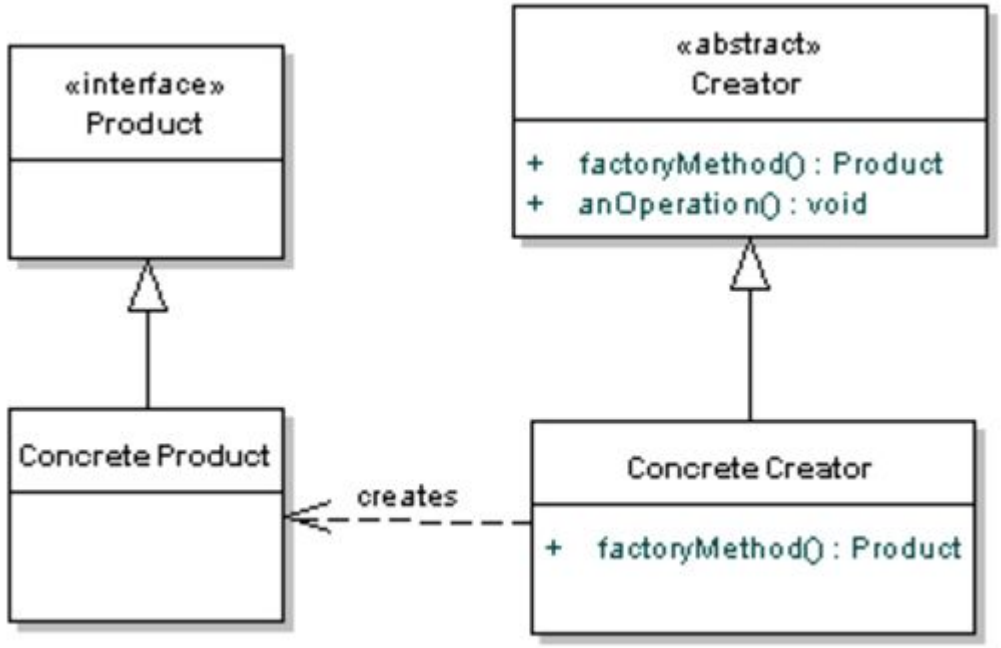
Product: Dog

Concrete Product: Poodle, Dalmation, Beagle

Method that must be implemented: speak();

Concrete Creator: DogFactory

Creator: Gets different dogs from the factory



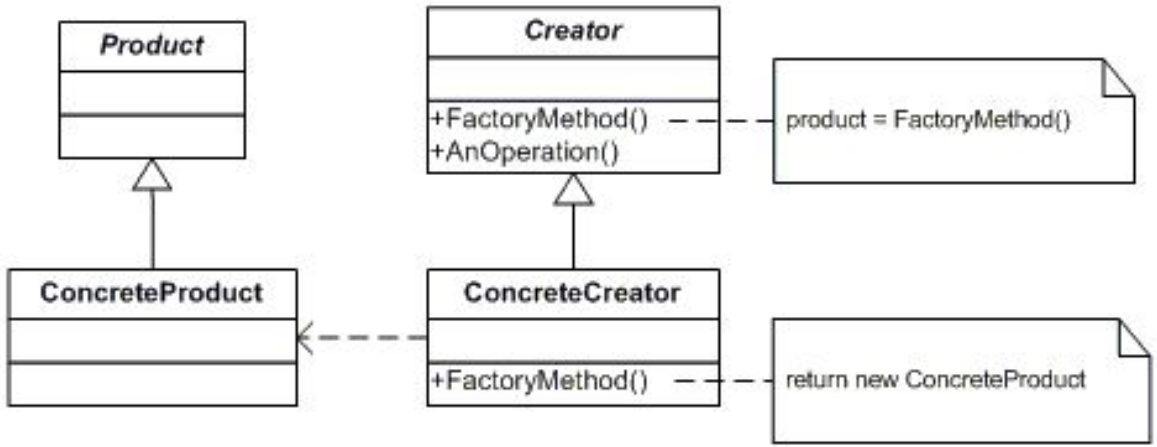
Overview

Product (Dog)

defines the interface of objects the factory method creates

Creator

declares the factory method, which returns an object of type Product.



Concrete Product (Poodle, Dalmation, Beagle)

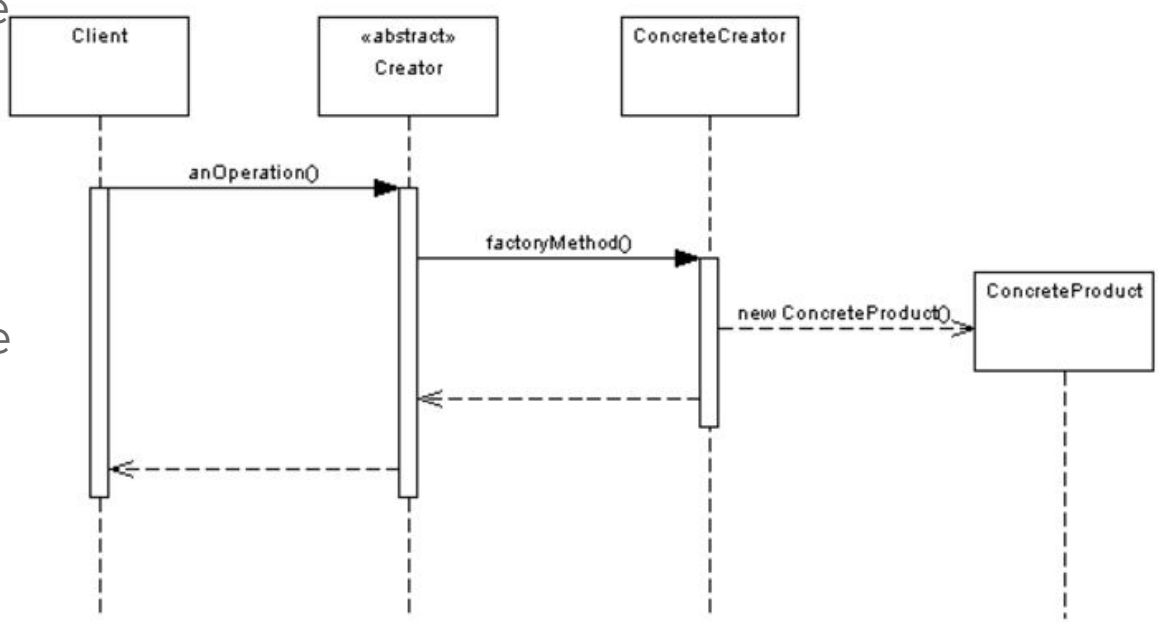
implements the Product interface

Concrete Creator

overrides the factory method to return an instance of a Concrete Product.

Factory Method: Behavior

- Client makes a call to the creator
- creator uses the factory method
- Factory method gets a new instance of concrete product
- It then completes the anOperation() on the concrete product
- Completes



Advantages of using the Factory Method

Advantages

- allows subclasses to choose what types of objects to create
- removes the instantiation of actual implementation classes from client code.
- makes our code more robust
- makes code less coupled
- Makes code easy to extend.
- provides abstraction between implementation and client classes through inheritance

Factory Method Example

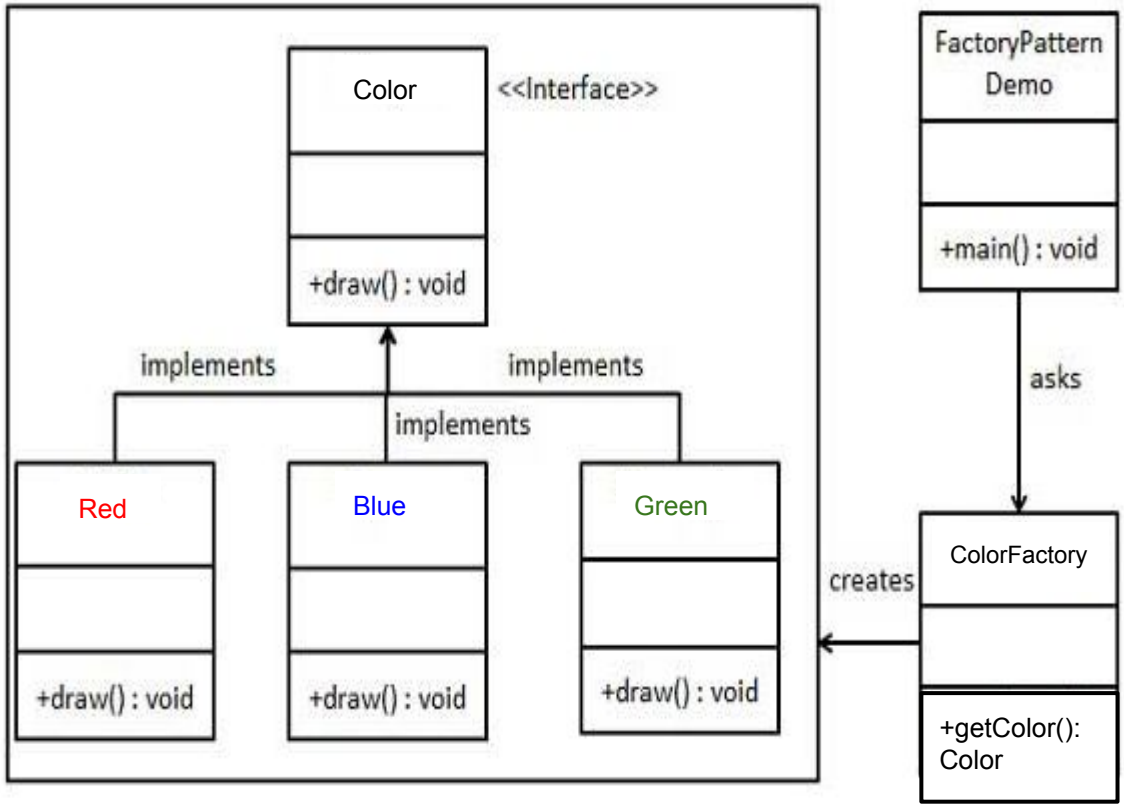
Example: UML

Interface: Color with method draw

Concrete Classes: Red, Blue, Green Implementing the same interface

Concrete Creator: ColorFactory

Creator: FactoryPatternDemo



Example: UML

Creator: FactoryPatternDemo

ASKS

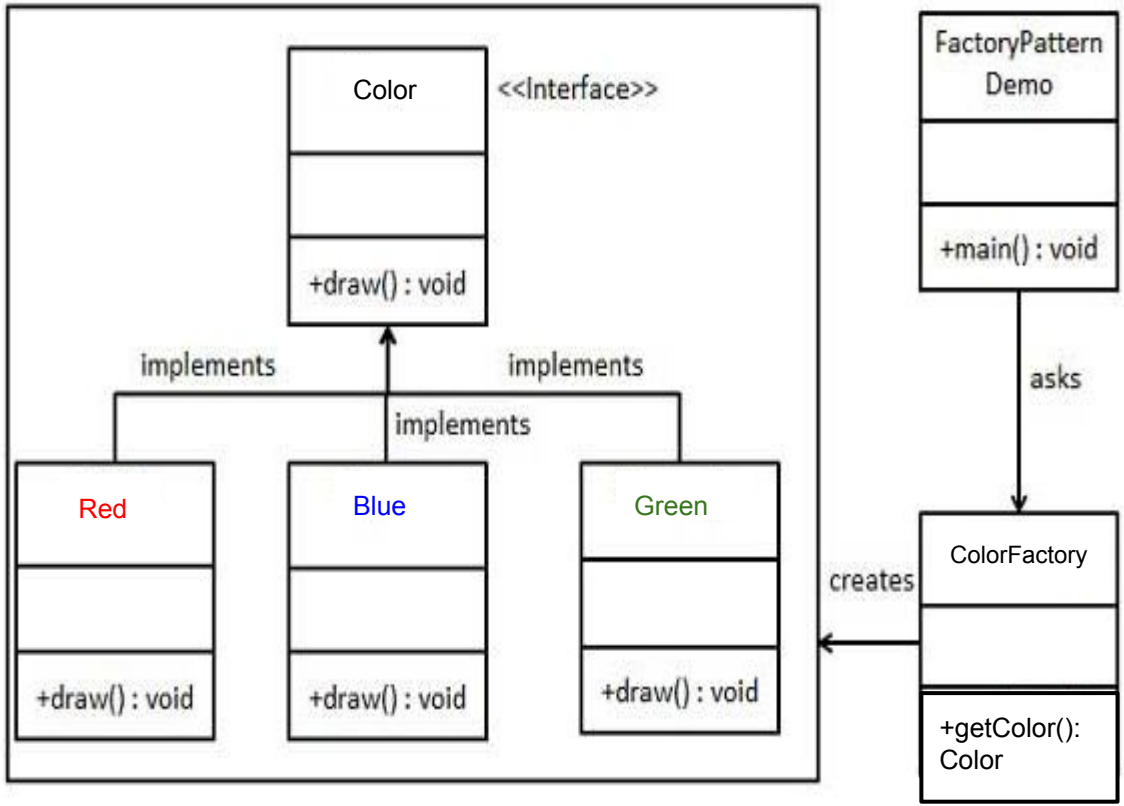
Concrete Creator: ColorFactory

CREATES

Concrete Classes: Red, Blue, Green

IMPLEMENTS

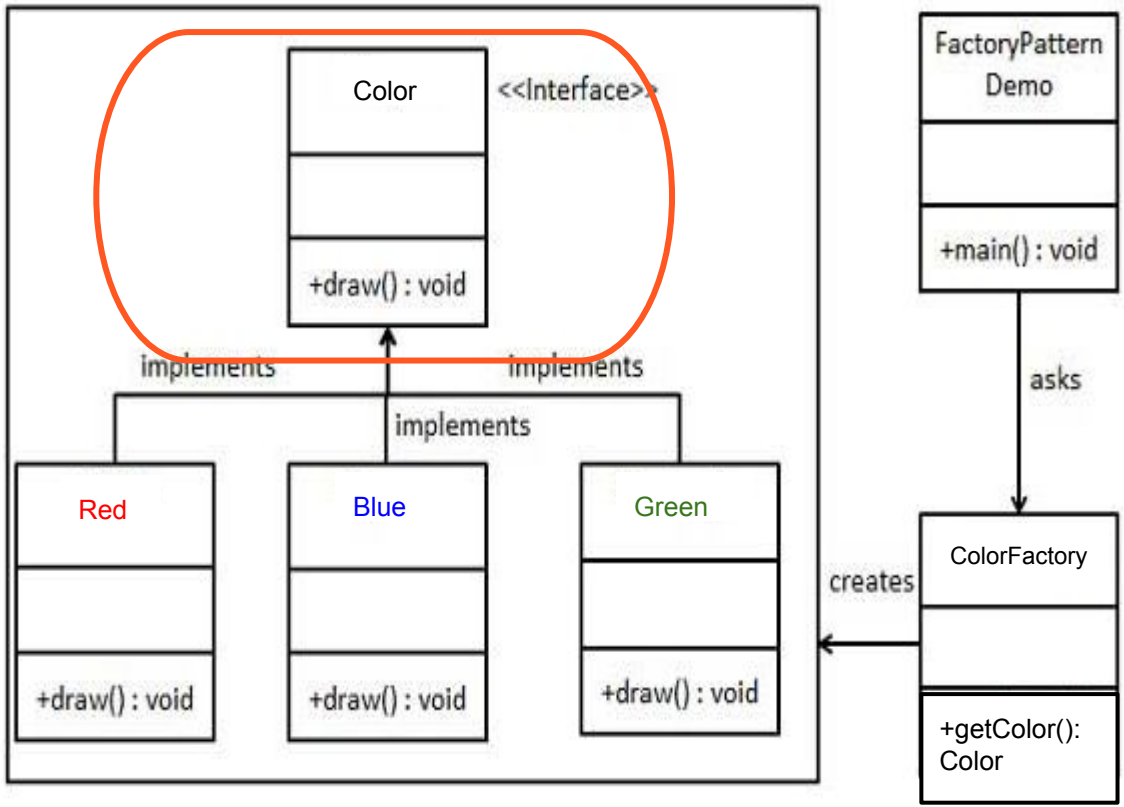
Product: Color Interface



Product: Interface

Example: UML

Interface: Color



Implementation: Interface

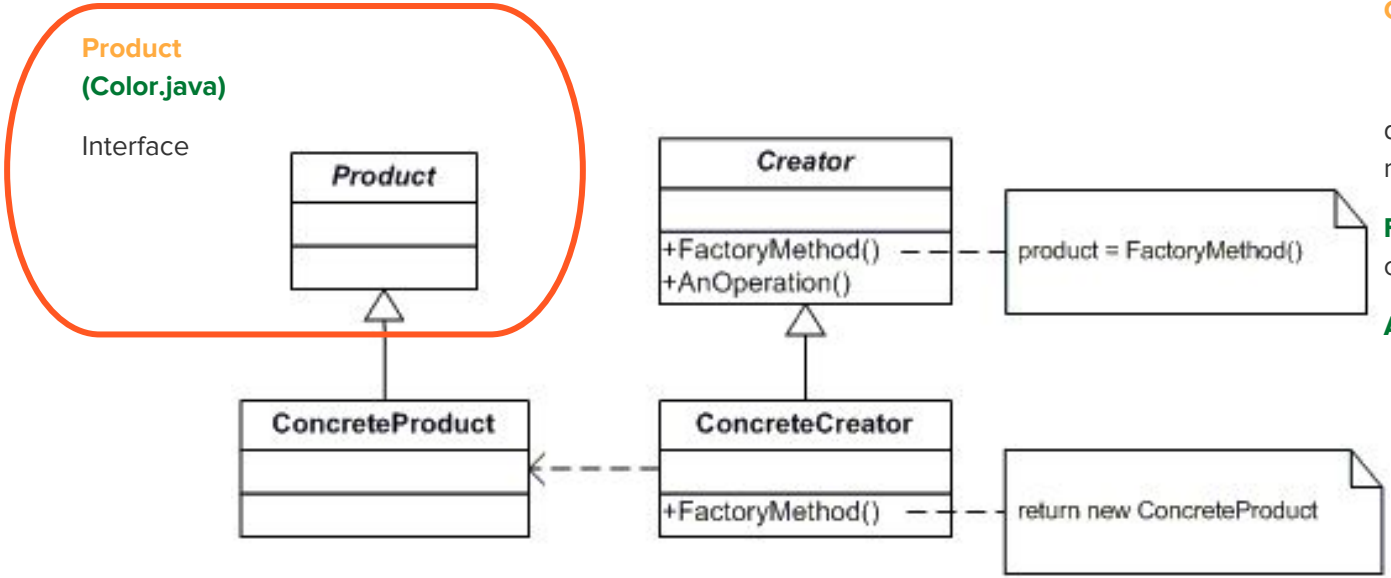
Step One: Create an Interface

This is the common interface for the objects that will be created.

Connecting to Diagram Definition

Product
(Color.java)

Interface



Creator

(FactoryMethodDemo.java)

declares the factory method, which returns an object of type Product.

FactoryMethod: Color color1 = colorFactory.getColor("Red")

AnOperation :color1.draw();

Concrete Product

(Red.java ,Blue.java ,Green.java)

implements the Product interface

Concrete Creator

(ColorFactory.java)

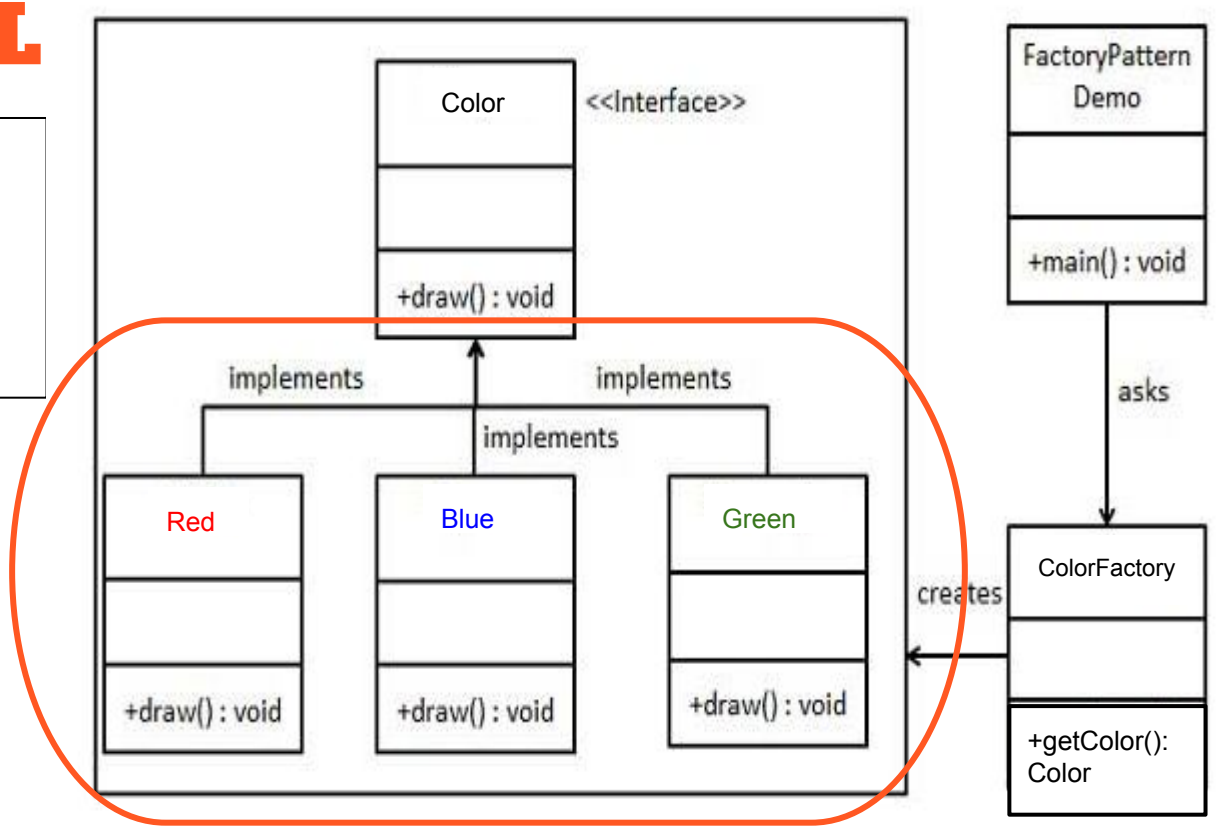
FactoryMethod: +Color();

overrides the factory method to return an instance of a Concrete Product.

Concrete Classes

Example: UML

Concrete Classes:
Red, Blue, Green
Implementing the same interface



Implementation: Concrete Classes

Note: Each concrete class implements the same interface

Blue.java | Green.java

```java
/**
 * Created by kirinmasood on 5/4/17.
 */
public class Green implements Color{
    @Override
    public void draw() {
        System.out.println("Inside Green::draw() method.");
    }
}
```

Blue.java | Green.java | Red.java

```java
/**
 * Created by kirinmasood on 5/4/17.
 */
public class Red implements Color{
    @Override
    public void draw() {
        System.out.println("Inside Red::draw() method.");
    }
}
```

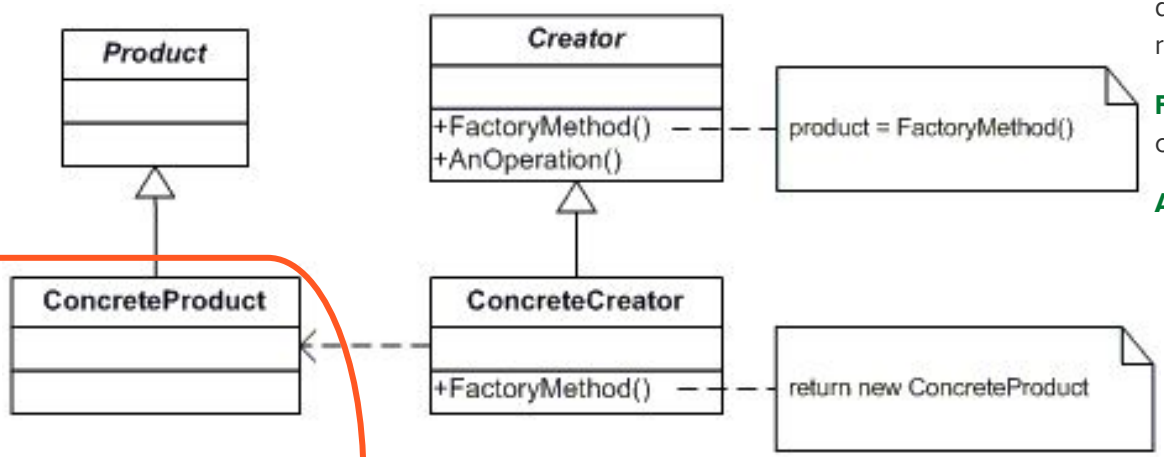
Blue.java

```java
/**
 * Created by kirinmasood on 5/4/17.
 */
public class Blue implements Color{
    @Override
    public void draw() {
        System.out.println("Inside Blue::draw() method.");
    }
}
```

Connecting to Diagram Definition

Product
(Color.java)

Interface



Creator

(FactoryMethodDemo.java)

declares the factory method, which returns an object of type Product.

FactoryMethod: Color color1 = colorFactory.getColor("Red")

AnOperation :color1.draw();

Concrete Product

(Red.java ,Blue.java ,Green.java)

implements the Product interface

Concrete Creator

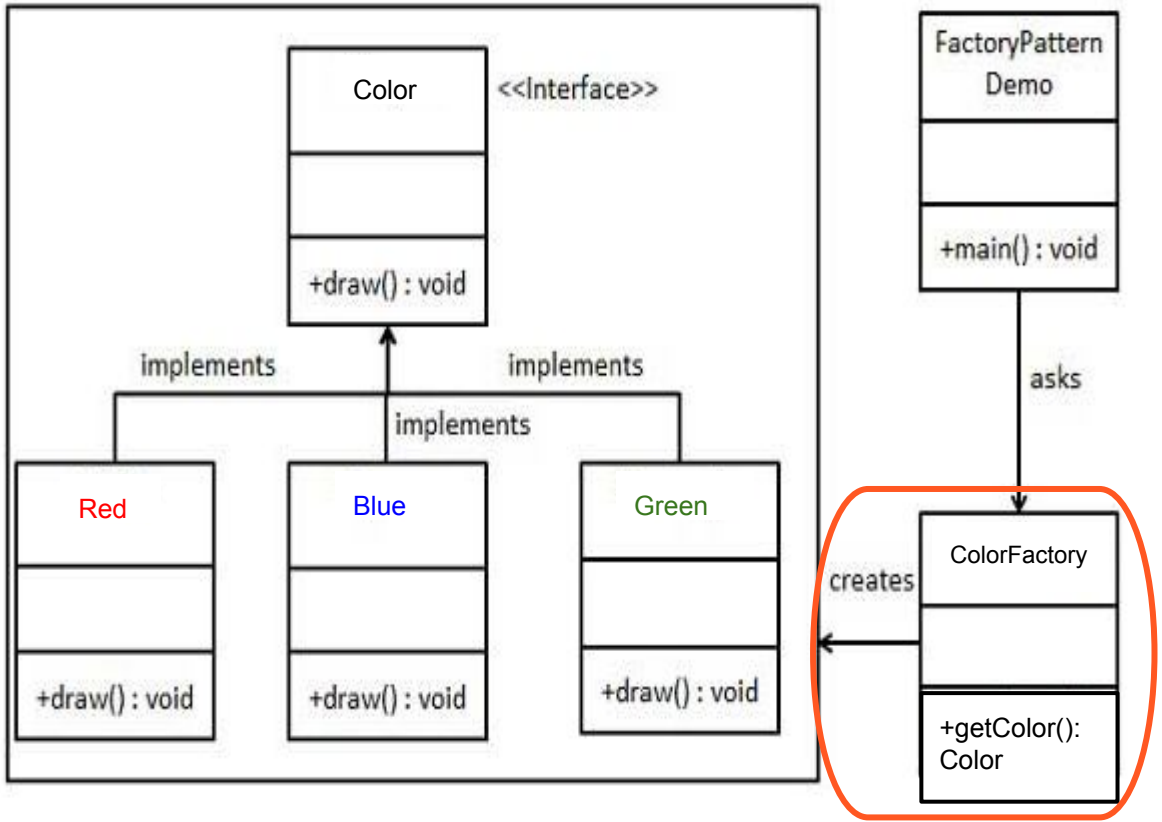
(ColorFactory.java)

FactoryMethod: +Color();

overrides the factory method to return an instance of a Concrete Product.

Example: UML

Concrete Creator:
ColorFactory
Generate object of concrete class based on given information



Concrete Creator

Implementation: Concrete Creator

The Factory generates object of concrete class based on information specified.

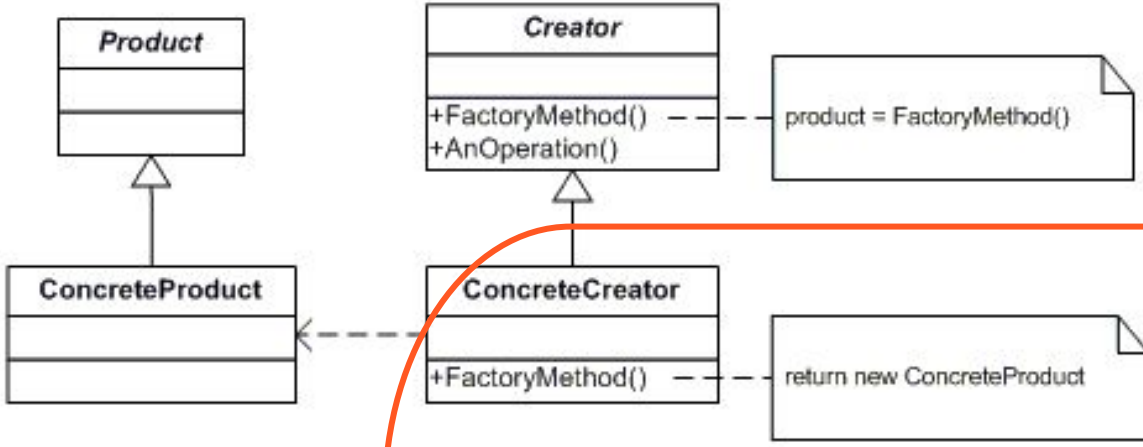
ColorFactory.java

```java
/**
 * Created by kirinmasood on 5/4/17.
 */
public class ColorFactory {
    public Color getColor(String colorShade){
    if(colorShade == null){
        return null;
    }
    if(colorShade.equalsIgnoreCase("BLUE")){
        return new Blue();

    } else if(colorShade.equalsIgnoreCase("GREEN")){
        return new Green();

    } else if(colorShade.equalsIgnoreCase("RED")){
        return new Red();
    }
    return null;
    }
}
```

Connecting to Diagram Definition

Product
(Color.java)

Interface



Creator

(FactoryMethodDemo.java)

declares the factory method, which returns an object of type Product.

FactoryMethod: Color color1 = colorFactory.getColor("Red")

AnOperation :color1.draw();

Concrete Product

(Red.java ,Blue.java ,Green.java)

implements the Product interface

Concrete Creator

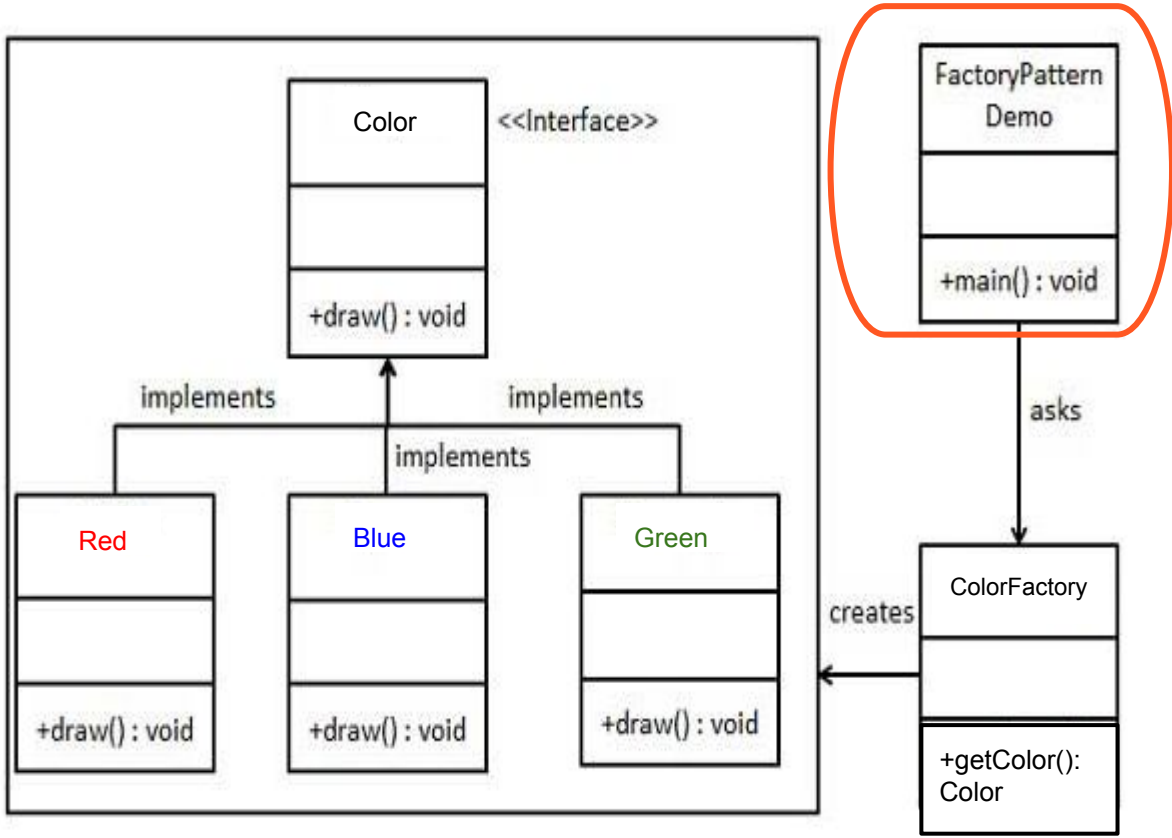
(ColorFactory.java)

FactoryMethod: +getColor();

overrides the factory method to return an instance of a Concrete Product.

Example: UML

Creator:
FactoryPatternDemo.java



Creator

Implementation: Creator

The Factory is used to get objects of the concrete class by passing on relevant information. In this case it is color name/shade.

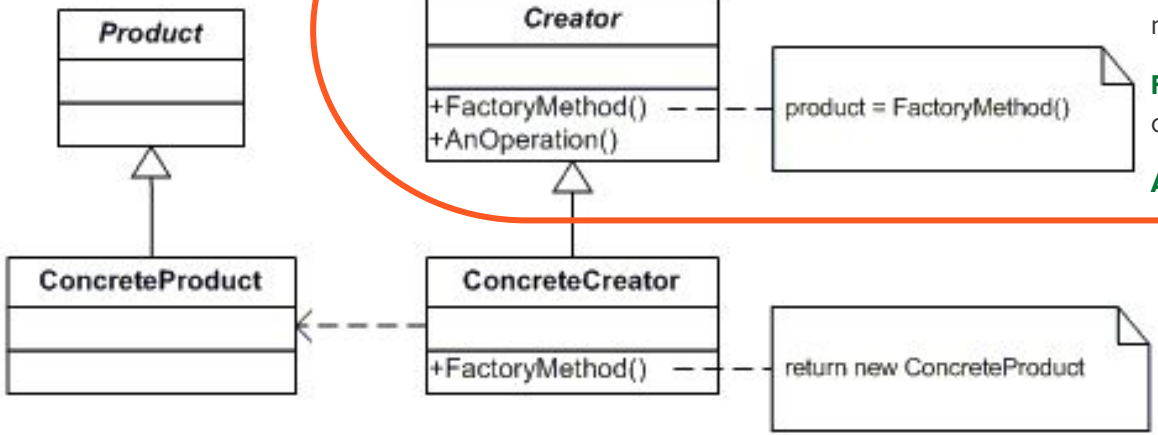
FactoryPatternDemo.java

```java
/**
 * Created by kirinmasood on 5/4/17.
 */
public class FactoryPatternDemo {

    public static void main(String[] args) {
        ColorFactory colorFactory = new ColorFactory();

        //get an object of Red and call its draw method.
        Color color1 = colorFactory.getColor("RED");

        //call draw method of Red
        color1.draw();

        //get an object of Green and call its draw method.
        Color color2 = colorFactory.getColor("GREEN");

        //call draw method of Green
        color2.draw();

        //get an object of Blue and call its draw method.
        Color color3 = colorFactory.getColor("BLUE");

        //call draw method of Blue
        color3.draw();
    }
}
```

Connecting to Diagram Definition

Product
(Color.java)

Interface



Creator

(FactoryMethodDemo.java)

declares the factory method, which returns an object of type Product.

FactoryMethod: Color color1 = colorFactory.getColor("Red")

AnOperation :color1.draw();

Concrete Product

(Red.java ,Blue.java ,Green.java)

implements the Product interface

Concrete Creator

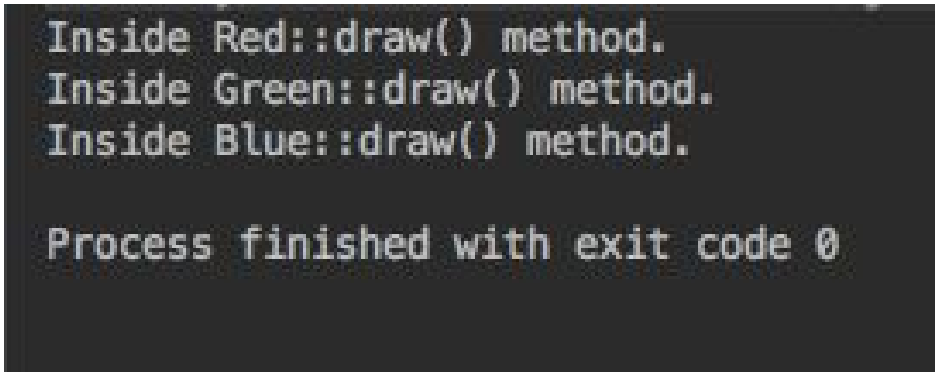
(ColorFactory.java)

FactoryMethod: +Color();

overrides the factory method to return an instance of a Concrete Product.

Output

Output verifies the draw method was called for each separate concrete class.



Thanks!