Design Pattern

< 2조 >

조철호 박은진
이용준 황보은영

Design Pattern - Factory Method

Factory Method
의도

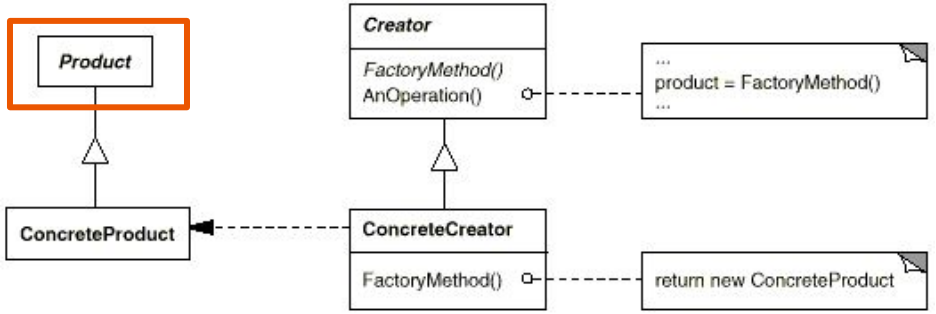
- 객체를 생성하기 위한 인터페이스를 정의하지만,

  어떤 클래스의 인스턴스를 생성할지에 대한 결정은 서브 클래스가 내리도록 하는 것

Factory Method
활용성

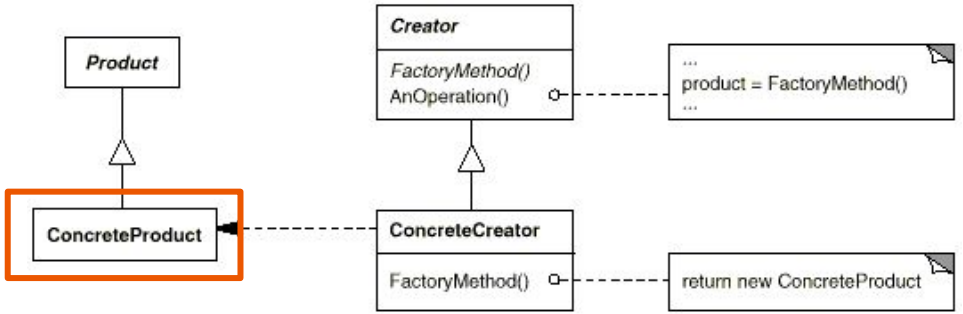
- 어떤 클래스가 자신이 생성해야하는 객체의 클래스를 예측할 수 없을 때
- 생성할 객체를 기술하는 책임을 자신의 서브 클래스가 지정했으면 할 때

Factory Method 구조



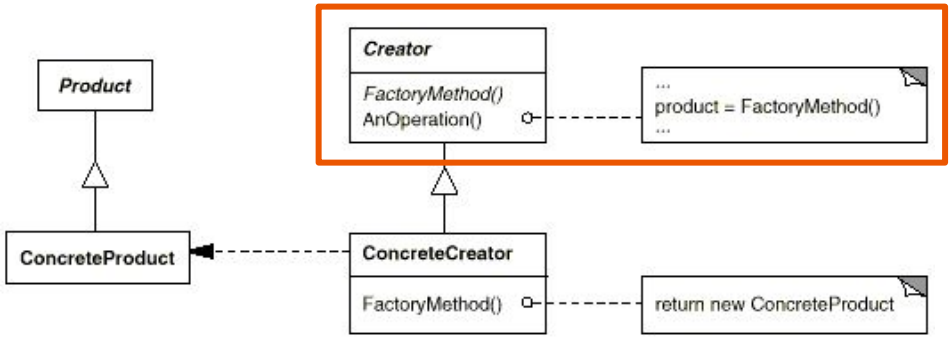
- **Product : Factory Method가 생성하는 객체의 인터페이스를 정의한다.**
- ConcreteProduct
- Creator
- ConcreteCreator

Factory Method 구조



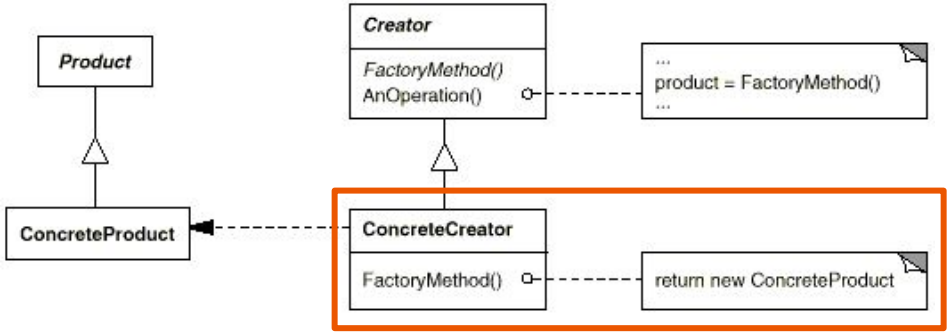
- Product
- **ConcreteProduct : Product 클래스에 정의된 인터페이스를 실제로 구현한다.**
- Creator
- ConcreteCreator

Factory Method 구조



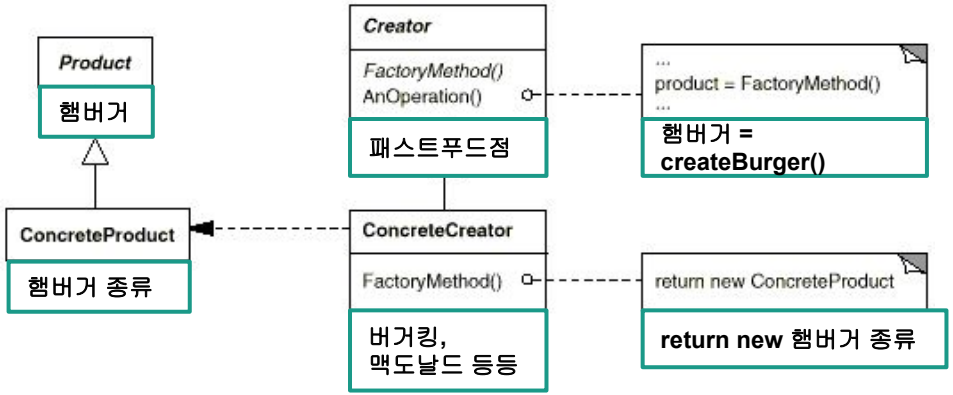
- Product
- ConcreteProduct
- **Creator : Product 타입의 객체를 반환하는 Factory Method를 선언한다.**
- ConcreteCreator

Factory Method 구조



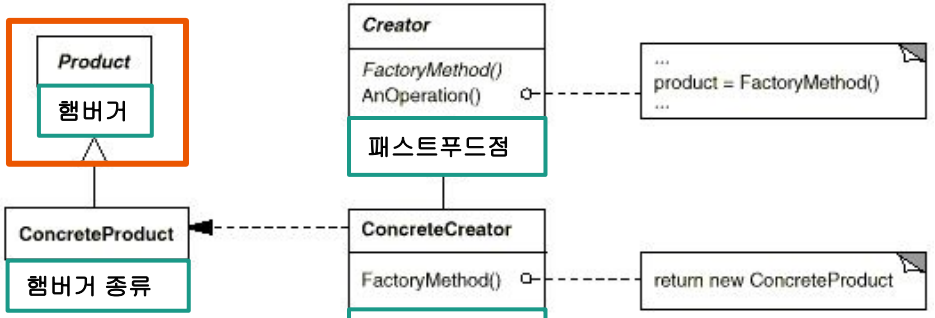
- Product
- ConcreteProduct
- Creator
- **ConcreteCreator : Factory Method를 재정의해서 ConcreteProduct의 인스턴스를 반환한다.**

Factory Method
예제 구조



- Product : 햄버거
- ConcreteProduct : 햄버거 종류(불고기 버거, 치즈 버거 등등..)
- Creator : 패스트푸드점
- ConcreteCreator : 패스트푸드점 종류(버거킹, 맥도날드 등등...)

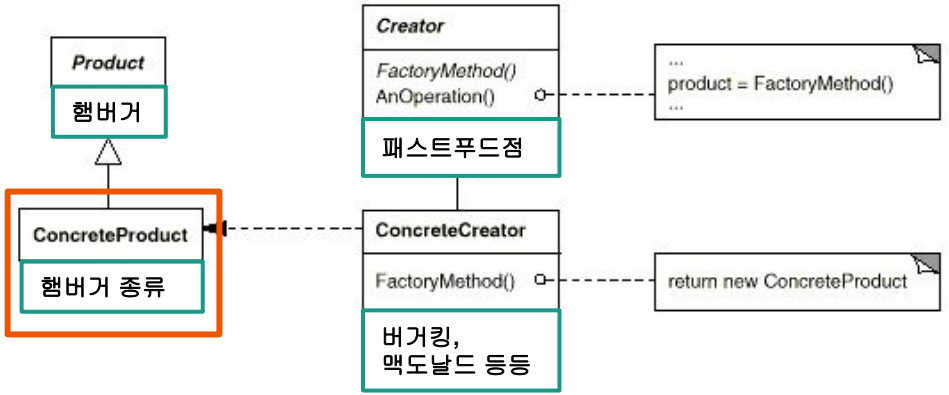
Factory Method
예제 코드



Hamburger.java

```java
package hamburger;

public abstract class Hamburger {
    public abstract String getName();
}
```

Factory Method
예제 코드



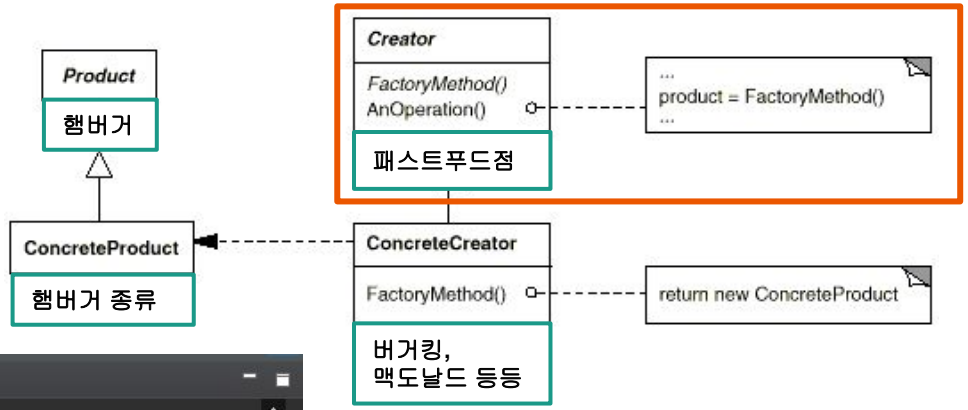
Bulgogi.java

```java
package hamburger;

public class Bulgogi extends Hamburger {

    @Override
    public String getName() {
        return "불고기 버거";
    }

}
```

Cheese.java

```java
package hamburger;

public class Cheese extends Hamburger {

    @Override
    public String getName() {
        return "치즈버거";
    }

}
```

Factory Method
예제 코드



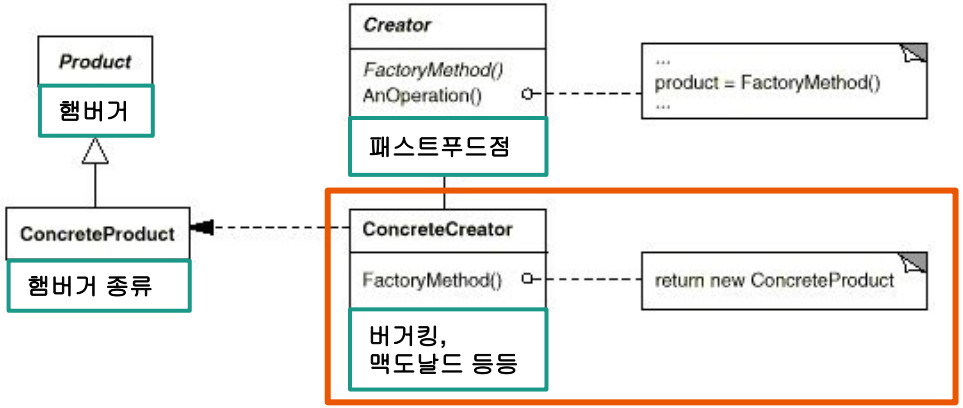
BugerStore.java

```java
package bugerStore;

import hamburger.Hamburger;

public abstract class BugerStore {
    public abstract Hamburger createBurger(String name);

    public abstract void order(String name);

}
```

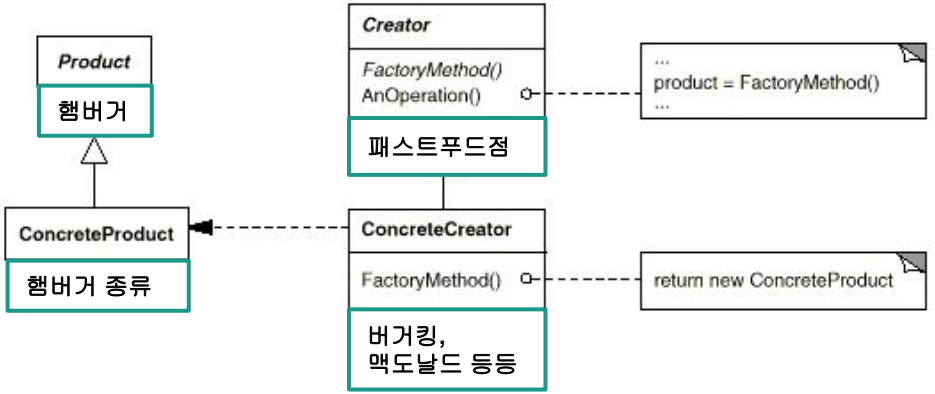
Factory Method
예제 코드

BurgerKing.java

```java
package bugerStore;

import hamburger.Bulgogi;

public class BurgerKing extends BugerStore {

    @Override
    public Hamburger createBurger(String name) {
        switch(name) {
            case "bulgogi" : return new Bulgogi();
            case "cheese" : return new Cheese();
        }
        return null;
    }

    @Override
    public void order(String name) {
        System.out.println("버거킹=> " + createBurger(name).getNam
    }

}
```



Factory Method
예제 코드



MainEntry.java

```java
package Main;

import bugerStore.BugerStore;
import bugerStore.BurgerKing;
import bugerStore.MomsTouch;

public class MainEntry {
    public static void main(String[] args) {
        BugerStore bk = new BurgerKing();
        BugerStore mt = new MomsTouch();

        bk.order("bulgogi");
        bk.order("cheese");

        mt.order("bulgogi");
        mt.order("cheese");
    }
}
```

Console × Problems

<terminated> MainEntry (1) [Java Application] C:₩myJavaDev₩jdk₩jdk1.8.0₩jdk1.8.0_301₩bin₩javaw.

```
버거킹=> 불고기 버거
버거킹=> 치즈버거
맘스터치=> 불고기 버거
맘스터치=> 치즈버거
```

Factory Method
결과

- 응용 프로그램에 국한된 클래스(Document)가 코드에 종속되지 않도록 해준다.

- 응용 프로그램은 Product 클래스에 정의된 인터페이스와만 동작하도록 코드가 만들어지기 때문에 사용자가 정의한 어떤 ConcreteProduct 클래스가 와도 동작할 수 있게 된다.

감사합니다.