생성패턴:

인스턴스를 만드는 절차를 추상화하는 패턴

생성패턴에 속하는 패턴은 객체를 생성 합성하는 방법이나 객체의 표현방법을 시스템과 분리한다

<https://wikidocs.net/897>

생성 패턴(creational pattern)은 인스턴스를 만드는 절차를 추상화하는 패턴입니다 이 범주에 해당하는 패턴은 객체를 생성 합성하는 방법이나 객체의 표현 방법과 (소프트웨어) 시스템을 분리해 줍니다.

클래스 생성 패턴 - 인스턴스로 만들 클래스를 다양하게 만들기 위한 용도로 상속을 사용함

객체 생성 패턴 - 인스턴스화 작업을 다른 객체에게 떠넘김

생성 패턴의 특징

생성 패턴은 시스템이 어떤 구체 클래스를 사용하는 지에 대한 정보를 캡슐화합니다.

생성 패턴은 이들 클래스의 인스턴스들이 어떻게 만들고 어떻게 서로 맞붙는지에 대한 부분을 완전히 숨깁니다

생성패턴 이슈

1. 생성패턴은 시스템이 어떤 Concrete class 사용하는지에 대한 정보를 캡슐화한다

2. 생성패턴은 이들 클래스의 인스턴스들이 어떻게 만들고 어떻게 결합하는지에대한 부분을 완전히 가려준다

* 생성패턴 사용시 무엇이 생성되고 누가 이것을 생성했으며 이것이 어떻게 생성되는지 언제 생성할것인지 결정하는데 유연성 확보가능

<https://www.devkuma.com/docs/design-pattern/abstract-factory-Methods/>

---> 각 패턴의 설명 ..?

prototype pattenr

<https://velog.io/@newtownboy/%EB%94%94%EC%9E%90%EC%9D%B8%ED%8C%A8%ED%84%B4-%ED%94%84%EB%A1%9C%ED%86%A0%ED%83%80%EC%9E%85%ED%8C%A8%ED%84%B4Prototype-Pattern>

https://readystory.tistory.com/122

객체를 생성하는데 비용이 많이들고 비슷한 객체가 이미있는경우에 사용되는 생성패턴

프로토타입 패턴은 원본 객체를 새로운 객체에 복사하여 필요에따라 수정하는 방식을 제공한다

DB 또는 데이터를 가져올때 시간이 많이들음으로써 그 데이터를 clone 을 사용하여 데이터들을 미리 복사해놓는다 ( 데이터 가져오는것보다 clone이 더효율적이다)

🍊 프로토타입, 장단점

프로토타입 패턴의 **장점**으로

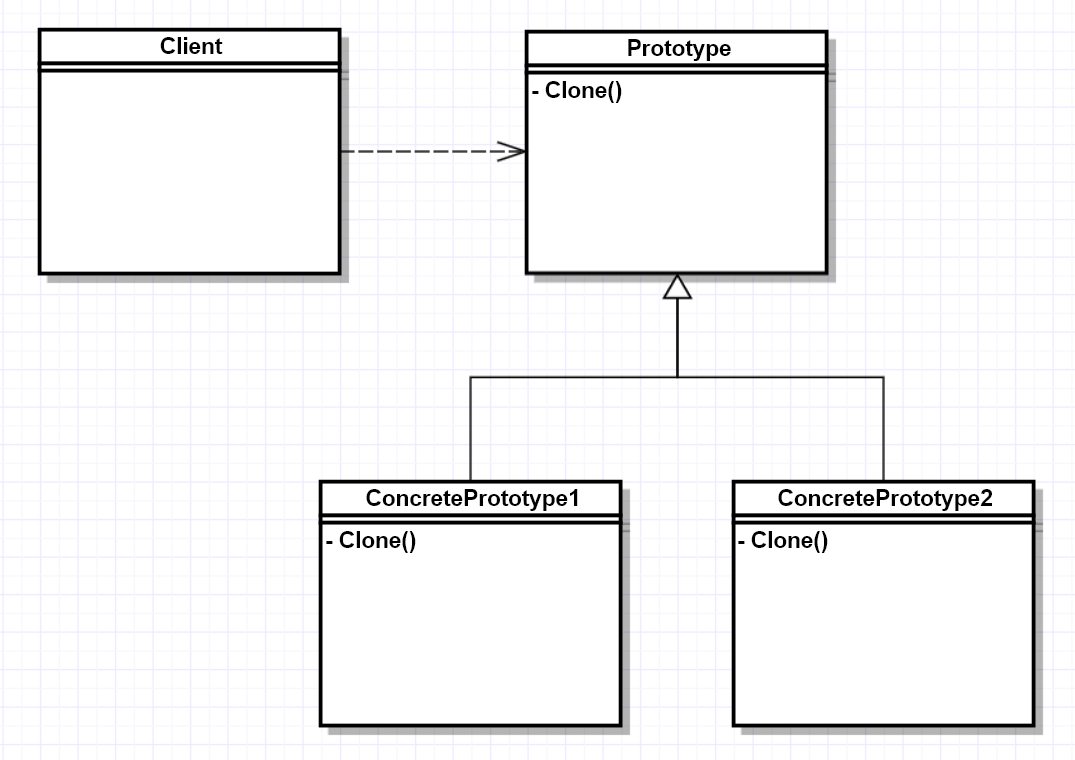
**첫**째로, 복잡한 객체를 만드는 과정을 숨길 수 있다.

**둘**째로. 기존 객체를 복제하는 과정이 새 인스턴스를 만드는 것보다 비용면에서 효율적일 수 있다.

**셋**째로,  clone에서 반환하는 타입이 clone을 정의한 클래스와 반드시 동일할 필요는 없다. 클래스의 계층 구조가 있을 때 추상화된 타입을 반환할 수 있는 유연성이 있다.

<https://kangworld.tistory.com/251>

<https://kangworld.tistory.com/249>



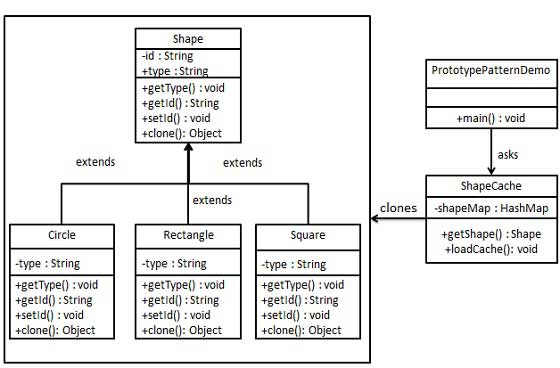
<<interface>> prototype

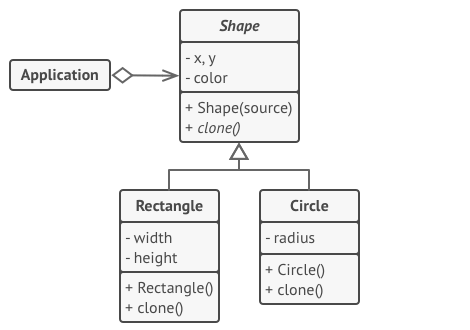
Cient 실체화관계 오버라이딩하여 실제기능으로 구현한다

일반화 관계 상속을받아 구현한다

프로토타입 패턴을 사용한다면 1회의 DB접근을 통해 가져온데이터를통해 다른 객체에 복사하여 사용하면 시간 비용적인 부분에서 절감가능

https://www.tutorialspoint.com/design\_pattern/prototype\_pattern.htm



\* 

Prototype 사용안할시 문제:

https://refactoring.guru/design-patterns/prototype

 Not all objects can be copied that way because some of the object’s fields may be private and not visible from outside of the object itself.

*Copying an object “from the outside”*[***isn’t***](https://refactoring.guru/cargo-cult)*always possible*

*.*builder 패턴

<https://refactoring.guru/design-patterns/builder> 원문

https://dev-youngjun.tistory.com/197

사용이유: produce different types and representations of an object using the same construction code.

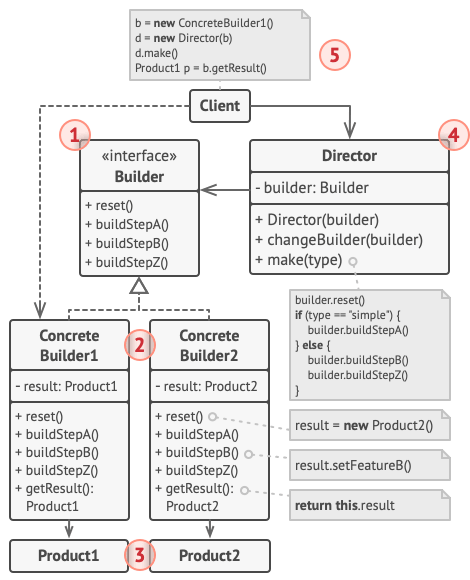
잘못된 해결책 ..? The simplest solution is to extend the base House

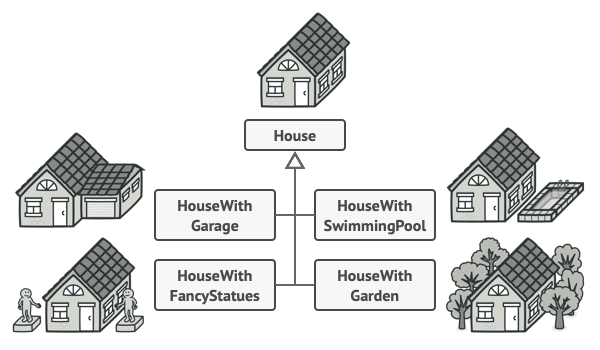
solutions:

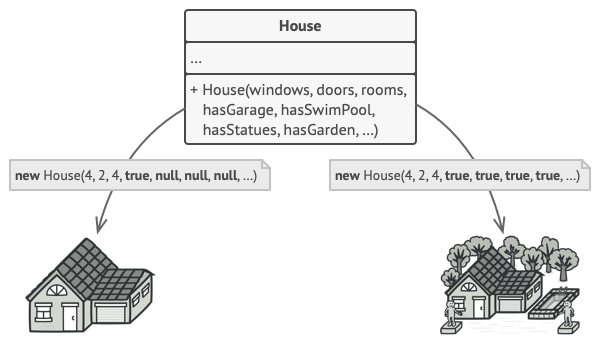
The important part is that you don’t need to call all of the steps.

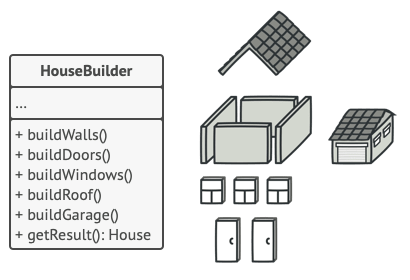
ou can call only those steps that are necessary for producing a particular configuration of an object.

빌더패턴 사진









1.점층적 생성자 패턴

생성자 오버로딩을 이용한 구현

단점: 1.인자가 추가될때마다 생성자 방식을 추가해야한다

2. 인자의 순서를 알고있어야한다

2.자바 빈 패턴

setter을 이용한 구현

인자순서 인지는 안해도되고 set메서드를 통하여 변수할당

그러나 인자수만큼 setter를 호출해야하고

객체생성을 한번에 할수없다

또한 setter로 인한 값변경이 가능하므로 constant 변수를 만들수없다

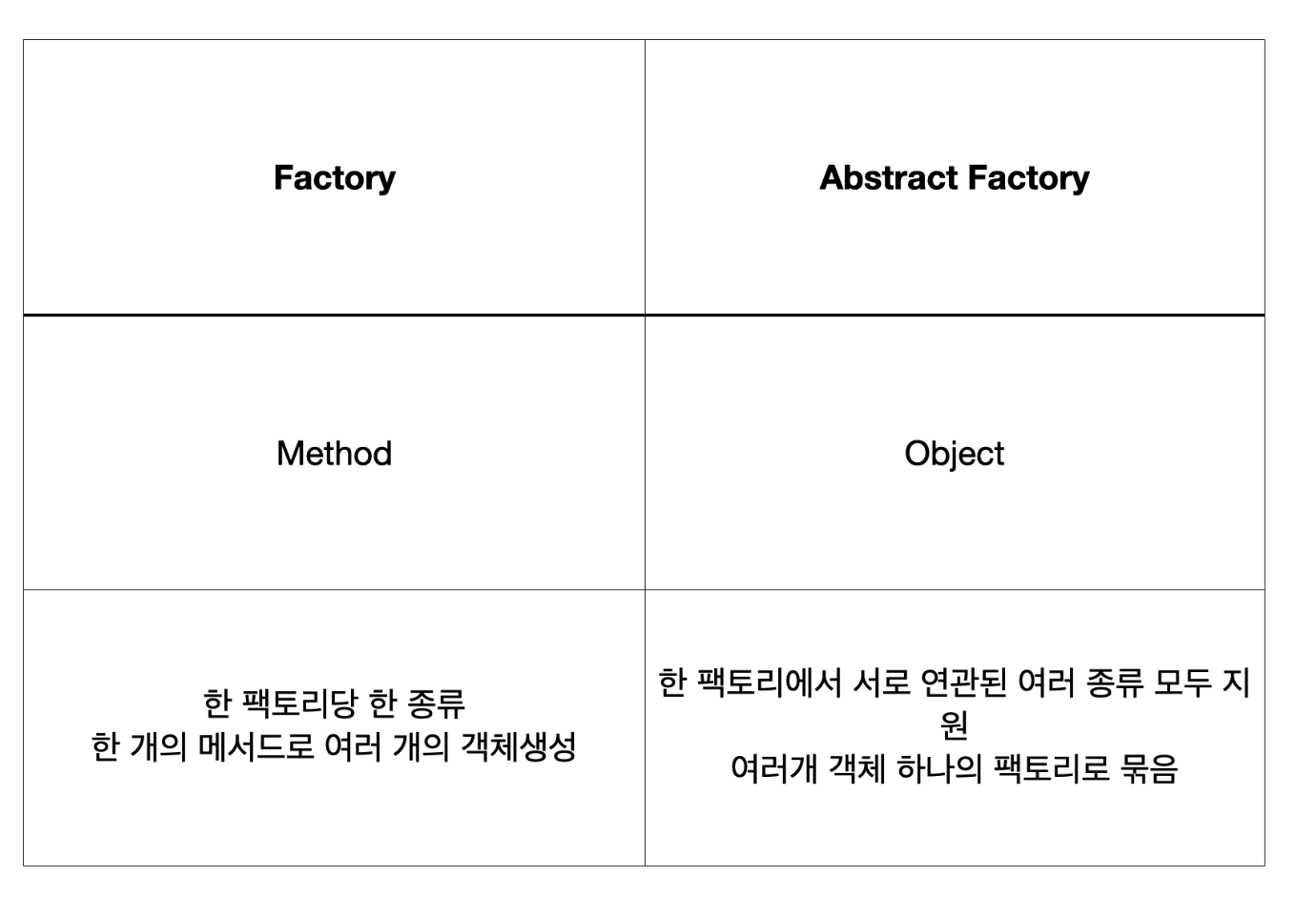
빌더패턴 사용안할시 단점:

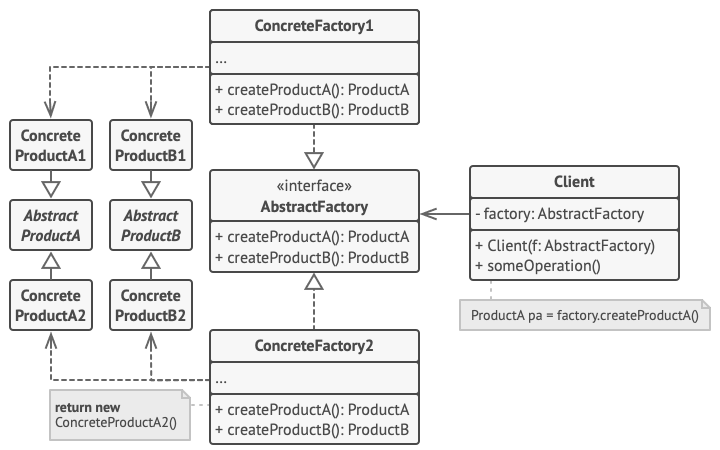
*1. 클라이언트 프로그램에서 팩토리 클래스를 호출할 때 Optional한 인자가 많아지면, 타입과 순서에 대한 관리가 어려워져 에러 발생 확률이 높아진다.  
2. 경우에 따라 필요 없는 파라미터들에 대해서 팩토리 클래스에 일일이 NULL 값을 넘겨줘야한다.*

abstract Factory

<https://velog.io/@bae_mung/%EB%94%94%EC%9E%90%EC%9D%B8-%ED%8C%A8%ED%84%B4-%EC%83%9D%EC%84%B1-Abstract-Factory-Pattern>

<https://velog.io/@kkh3/Abstract-Factory-Pattern>





<https://velog.io/@ililil9482/Abstract-Factory-Pattern>

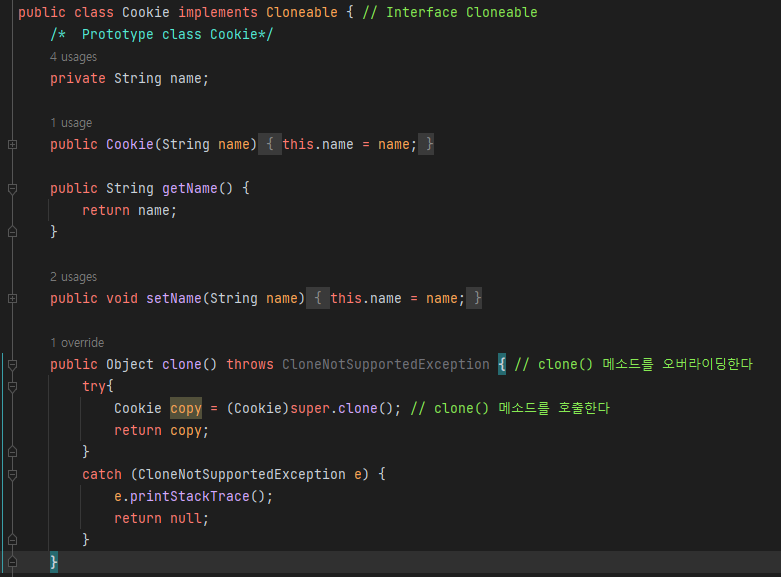
코드 삽입순서:

Prototype

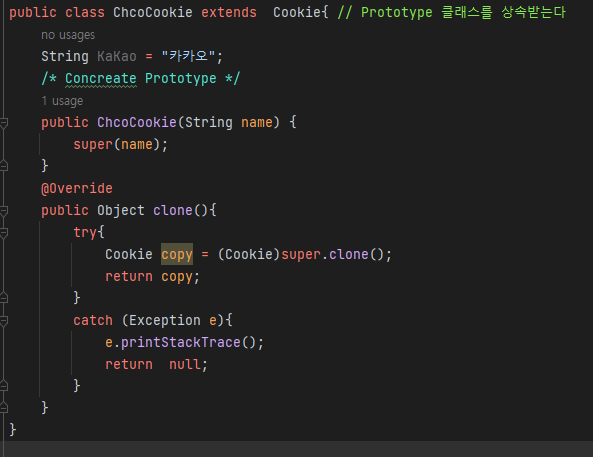
Cookie 클래스 먼저 보여주고 implemnets Cloneable 을 받는다는것과 clone메소드를 오버라이딩한다는것을 알려준다 원본객체를 copy한 복사 객체를 반환한다

CookieMachine을 통해서 쿠키를 복사한다 Main클래스에서 CokkieMachin에 쿠키를 넣으면 주소값이 다른 쿠키가 복사가된다

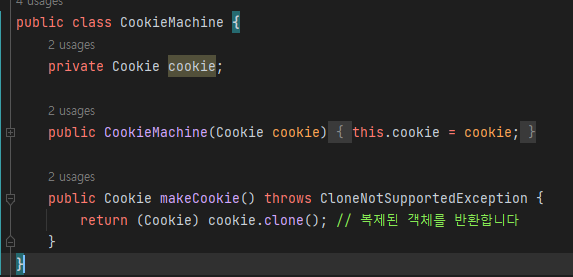
Cookie.java



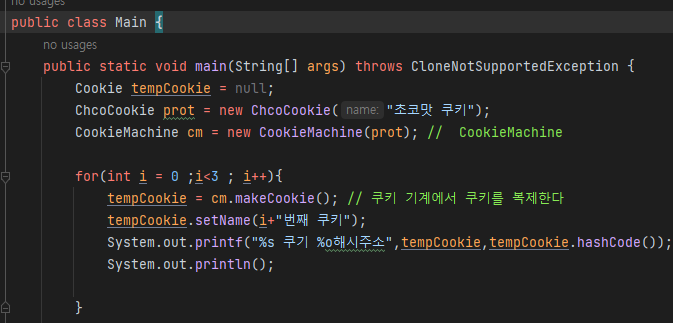
ChcoCookie.java



CookieMachine.java

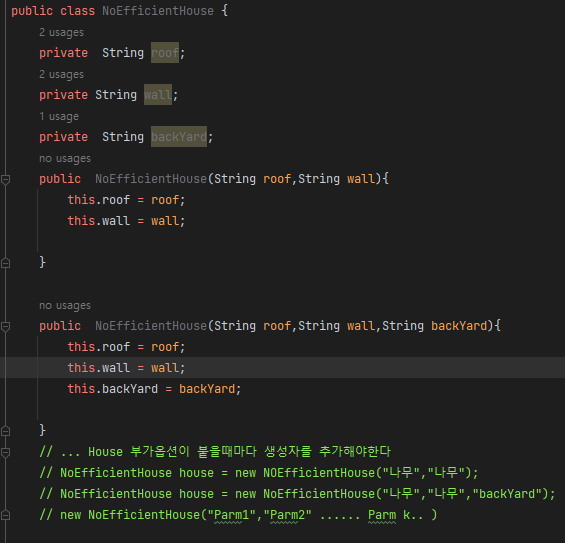


Main.java

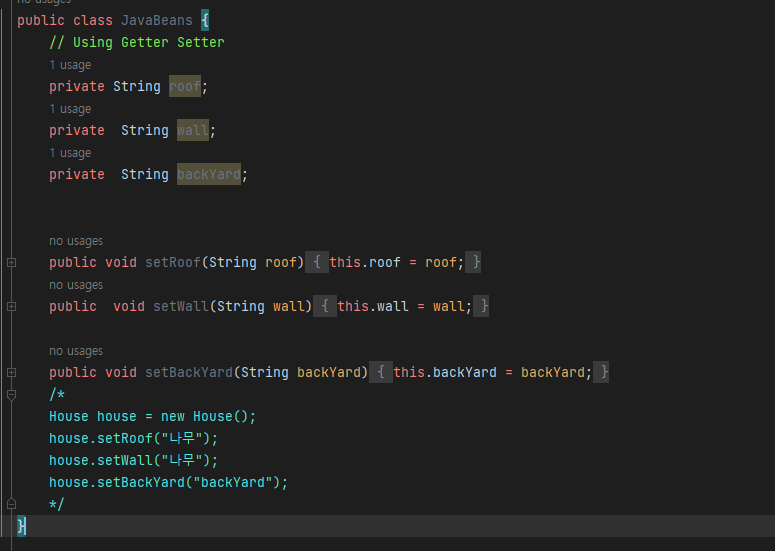


Builder Pattern

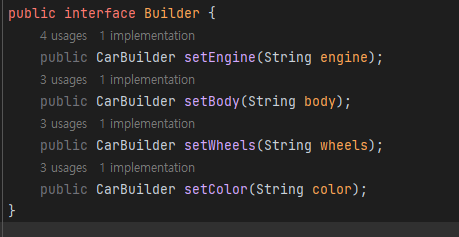
No efficientHouse.java



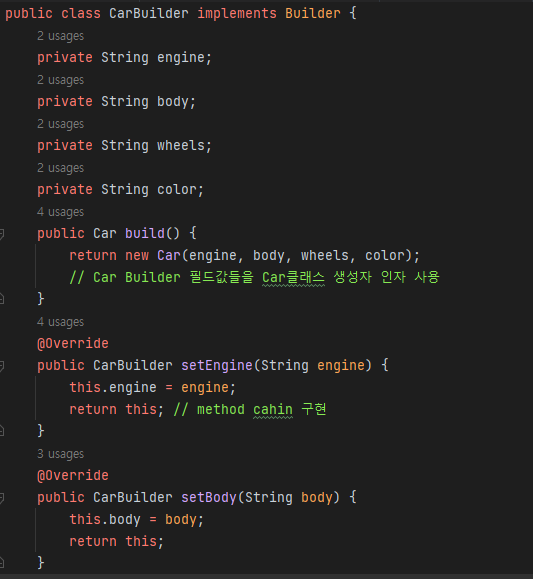
2. javaBeans



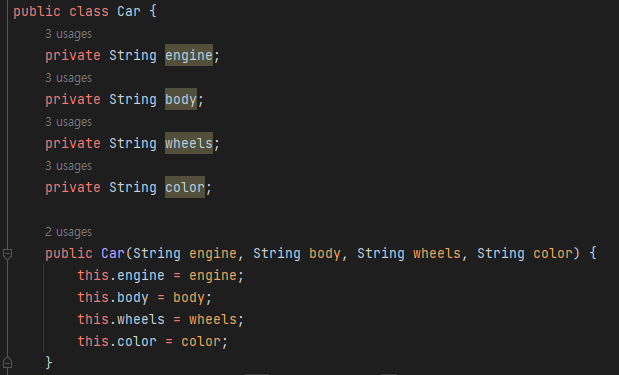
Builder Interface



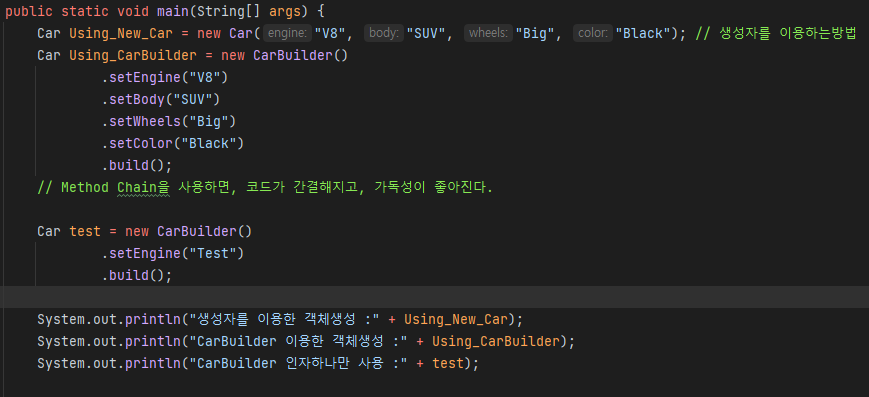
CarBuilder.java

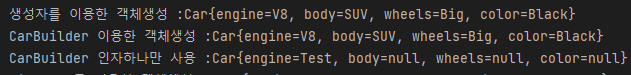


Car.java

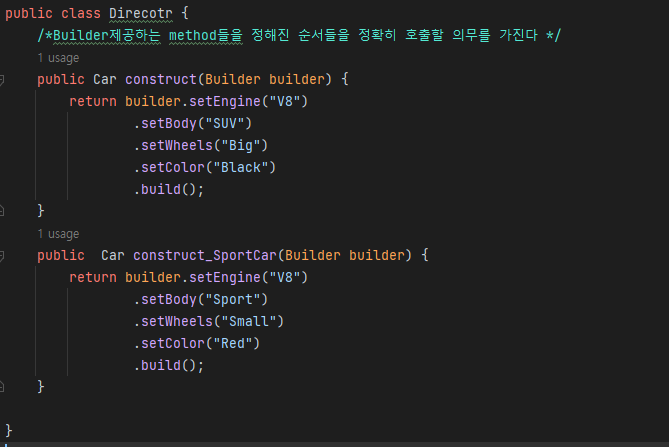


Main.java

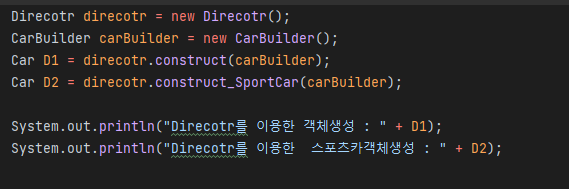


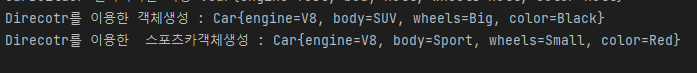


Director.java



Main.java



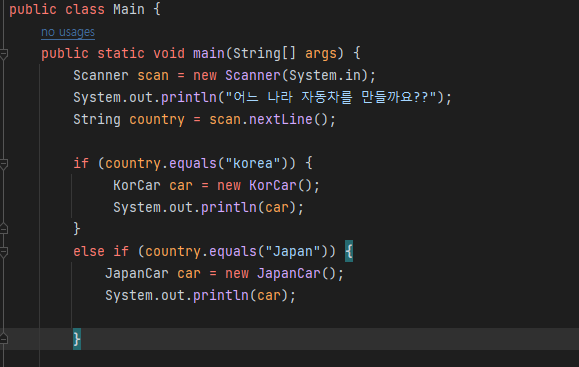


Abstract Pattern 안좋은예시

KOR.CAR

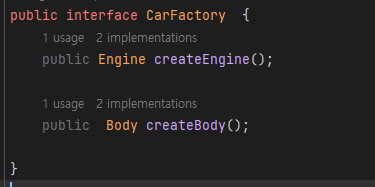
package My\_absFactory\_Pattern.Old;  
  
import java.awt.font.TextHitInfo;  
  
public class KorCar {  
 private String engine;  
 private String body;  
 private String wheels;  
 private String color;  
  
  
 KorCar(){  
 this.engine = "Kor car engine";  
 this.body = "Kor car body";  
 this.color = "RED";  
 this.wheels = "Kor car wheels";  
 }  
  
 @Override  
 public String toString() {  
 return "KORCar{" + "engine=" + engine + ", body=" + body + ", wheels=" + wheels + ", color=" + color + '}';  
 }  
}

MAIN.java

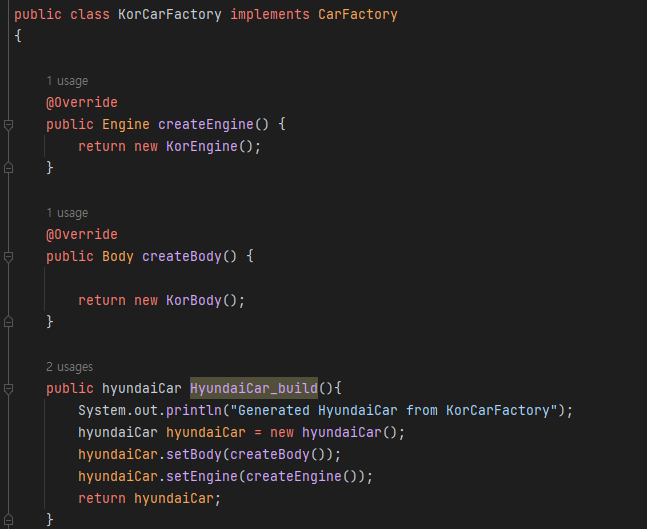


Abstract Patteren 좋은예시

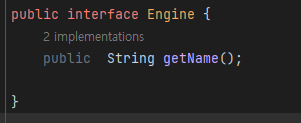
CarFactory.java



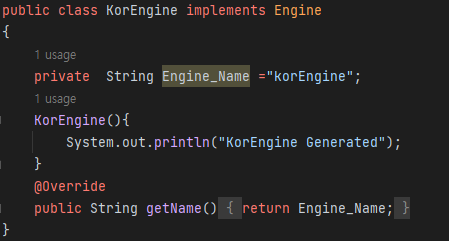
KorCarFactory.java



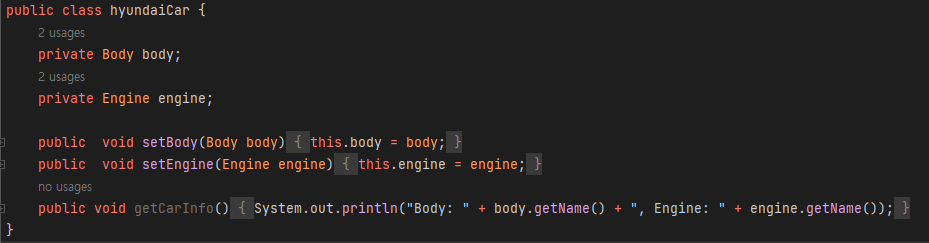
Engine.Interface



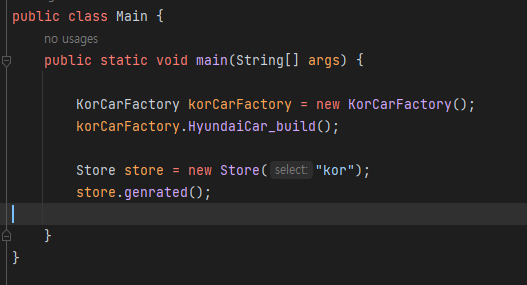
KorEngine.java



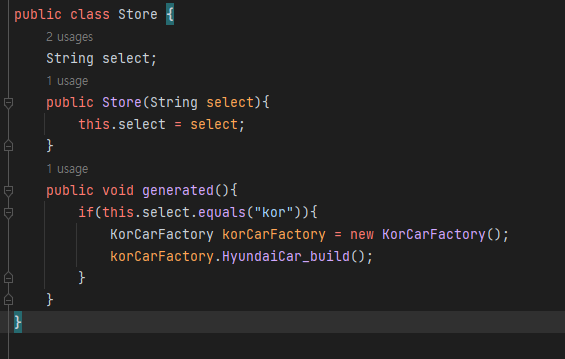
hyundaiCar.java



Main.java



Store.java



Main.java

