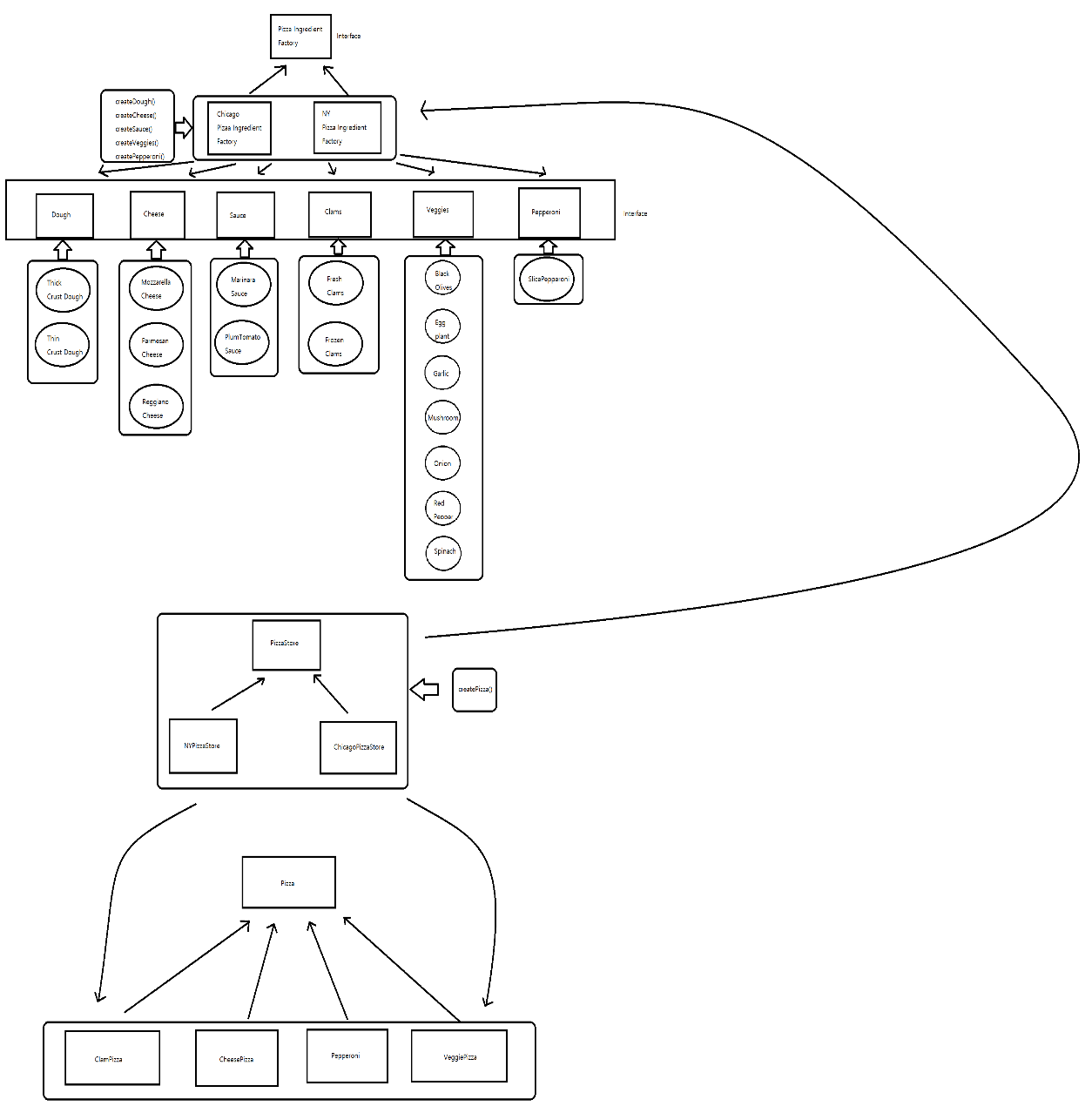
|  |  |
| --- | --- |
| 과목명 | 디자인패턴 |
| 담당교수 | 박제호 교수님 |
| 학과 | 소프트웨어학과 |
| 학번 | 32155068 |
| 이름 | 홍승기 |
| 제출일자 | 2020.5.03 |



1. Pizzaaf



위 표는 Pizzaaf 패키지를 구성하는 클래스와 인터페이스를 도식화 하여 표현한 것이다

그림으로 확인 할 수 있는 것은 메인 메소드에서 해당되는 PizzaStore 객체를 생성하여 프로그램이 돌아가면 orderPizza() 메소드에 재료를 전달하면 해당되는 PizzaStore 객체는 자신이 가지고 있는 PizzaIngredientFactory 객체를 생성하고, 전달된 재료의 이름에 해당되는 피자 객체를 생성하며 앞서 생성된 PizzaIngredientFactory 객체 같이 전달해 준다. 그 과정에서 지정된 dough, clams, Sauce, Cheese, Veggies, Pepperoni 객체를 생성하게 된다. 정리해 보면

1. ex) Main에서 NYPizzaStore 객체 생성

2. Pizza pizza = NYPizzaStore.orderPizza(‘cheese’) 라고 선언하게 되면 PizzaStore 클래스의 orderPizza 메소드에 cheese 가 전달되며 createPizza(‘cheese’) 메소드 호출

3. NYPizzaStore 클래스에서 재정의한 createPizza 메소드가 호출되며 NYPizzaIngredientFactory 객체를 생성

4. 전달된 cheese에 해당 되는 CheesePizza 객체를 생성하면서 앞서 생성된 PizzaIngredientFactory 객체를 전달해 주고, Pizza의 이름 setName메소드를 이용하여 설정해준다.

5. PizzaIngredientFactory 객체를 전달 받은 CheesePizza의 생성자가 실행되며 자신이 가지고 있던 PizzaIngredientFactory 객체에 전달받는 것을 담아 둔다.

6. 위 과정이 종료되면 NYPizzaStore 클래스에서 생성된 Pizza(CheesePizza)를 리턴 해준다.

7. 6번 과정에서 리턴된 Pizza를 받아 객체에 저장하고 orderPizza메소드의 남은 코드

System.***out***.println("--- Making a " + pizza.getName() + " ---");

pizza.prepare();

pizza.bake();

pizza.cut();

pizza.box();

를 순서대로 실행한다. 여기서 pizza는 6번에서 리턴 받은 CheesePizza임을 유의해야 한다. 그러므로 bake, cut, box 메소드는 CheesePizza가 상속 받는 Pizza 추상 클래스 안에 정의된 bake, cut, box 가 수행 되지만, 추상 메소드로 정의된 prepare메소드는 CheesePizza 클래스에서 재정의한 prepare 메소드가 수행된다.

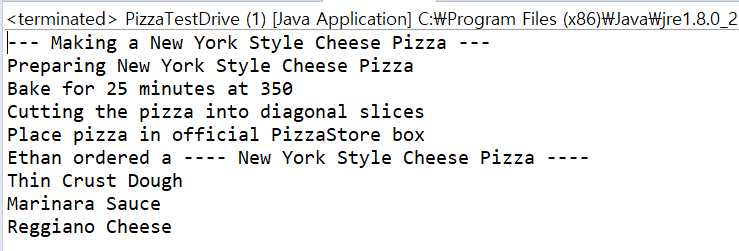
8. CheesePizza 클래스에 정의된 prepare 메소드가 수행되면서 5번 과정에서 담아둔 PizzaIngredientFactory 객체를 이용하여 dough, sauce, cheese 객체를 생성한다. 이 객체들은 부모 클래스인 Pizza에 정의된 dough, sauce, cheese로 전달된다.

9. 모든 과정이 완료되면 orderPizza 메소드는 종료되며 Main에 생성된 Pizza객체에 만들어진 Pizza 객체를 리턴해준다.

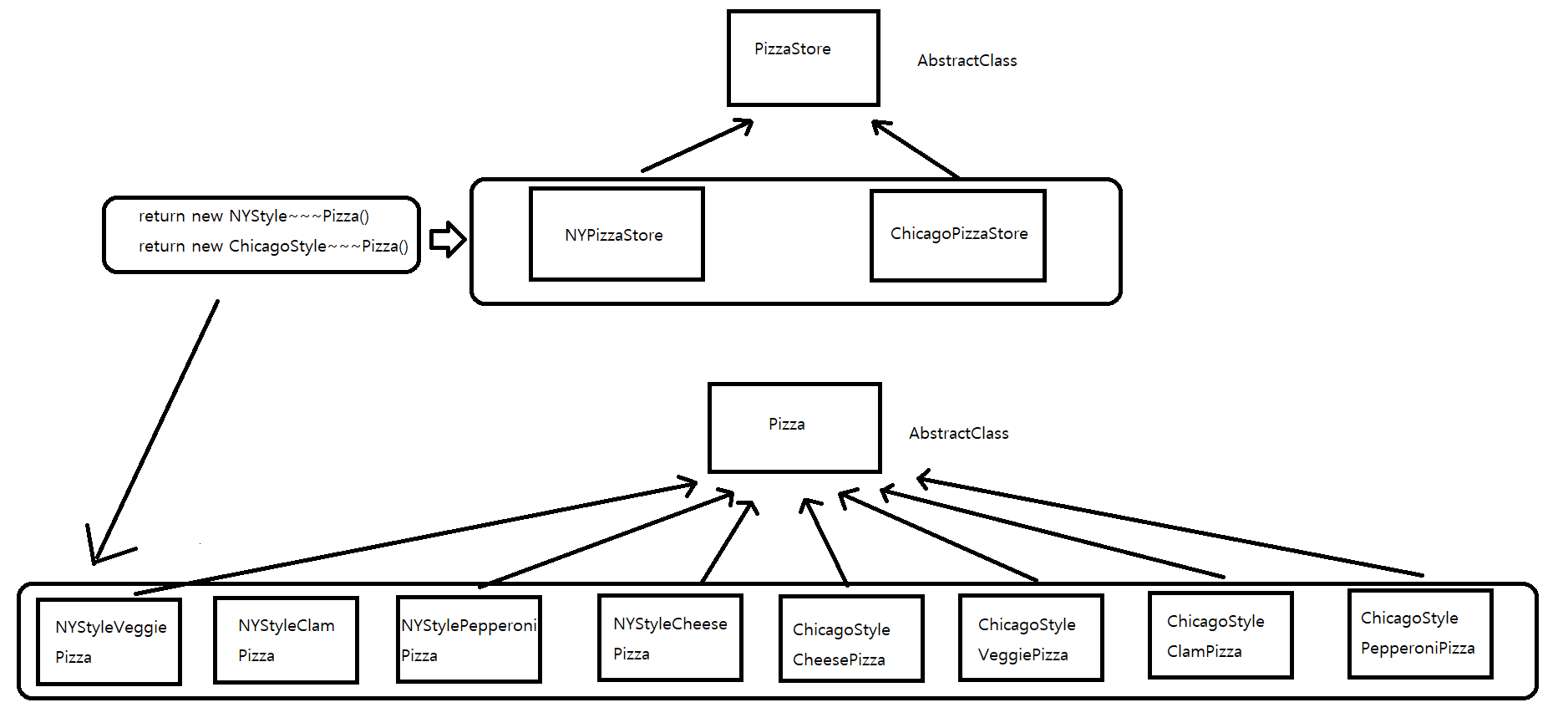
10. Main에서 Pizza를 System.out.print를 통해 출력해보면 Pizza 추상 클래스에 정의된 toString 메소드가 수행되며 생성된 Pizza의 이름(4번 과정에서 설정), dough, sauce, cheese, pepperoni 정보를 출력해준다.

아래 사진은 수행결과를 보여준다. 위 순서대로 수행됨을 알 수 있다.

실제 출력되는 부분은 위 과정에서 7번 과정부터 이루어 진다.



1. Pizzafm



위 그림은 Pizzafm 패키지를 도식화하여 표현한 것이다. 그림을 통해 코드의 구성을 살펴 보면 PizzaStore, Pizza 라는 추상 클래스가 존재하며 PizzaStore은 NYPizzaStore, ChicagoPizzaStore라는 두가지의 자식 클래스 가지고 있으며 Pizza 클래스는 NYStyle 4개, ChicagoStyle 4개의 클래스를 자식 클래스로 가지고 있다. Main 메소드에서 프로그램이 실행되면 해당되는 PizzaStore 객체가 생성되며 각 재료에 맞는 생성된 PizzaStore의 style에 맞는 Pizza 객체가 생성되면서 리턴된다. 수행 과정을 정리해 보면

1. ex) Main에서 NYPizzaStore 객체 생성
2. . Pizza pizza = NYPizzaStore.orderPizza(‘cheese’) 라고 선언하게 되면 PizzaStore 클래스의 orderPizza 메소드에 cheese 가 전달되며 createPizza(‘cheese’) 메소드 호출
3. NYPizzaStore 클래스에서 재정의한 createPizza 메소드가 호출되며 new NYStyleCheesePizza 객체를 리턴해준다.
4. NYStyleCheesePizza 객체가 생성되면서 생성자가 실행되며 부모클래스(Pizza)에 선언된 name, dough, sauce, toppings(Array) 변수에 지정된 요소들을 넣는다.
5. 다시 PizzaStore 클래스로 돌아오고, 3번 과정에서 리턴받은 해당되는 Pizza를 저장해두고 남은 아래 코드 들을 수행한다.

System.***out***.println("--- Making a " + pizza.getName() + " ---");

pizza.prepare();

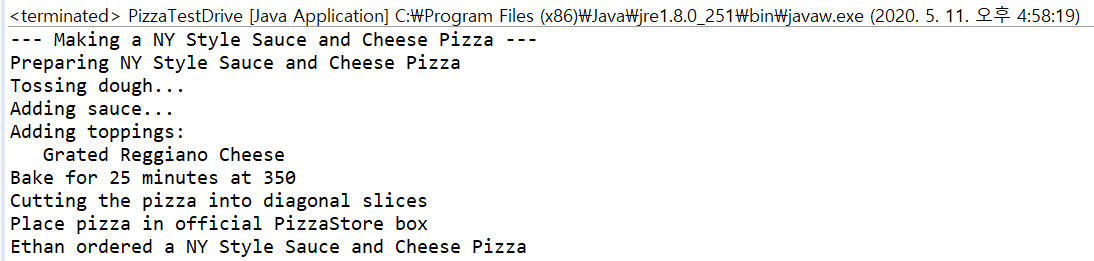
pizza.bake();

pizza.cut();

pizza.box();

**return** pizza;

1. 위 코드들이 수행되는 과정에서의 pizza라는 객체는 3번에서 전달된 pizza객체를 담은 변수임을 유의하고, 해당되는 prepare, bake, cut, box, 메소드를 실행하고 해당 pizza 객체를 리턴해준다.
2. Main에서 전달 받은 6번 과정에서 전달받은 pizza가 최종 결과물이 되며 해당 pizza를 출력해보면 아래와 같은 결과를 얻을 수 있다.



위 결과물은 5번 과정부터 순차적으로 출력 됨을 알수 있다.

1. 결과

두개의 패키지는 비슷한듯 하면서 다른 차이점이 있다. 1번 Pizzaaf는 추상 팩토리 패턴을 사용한 것이고, 2번은 팩토리 메소드 패턴을 사용했다는 것에 차이점이 있다. 두가지 방법의 공통점은

1. 객체 생성을 캡슐화 한다
2. 구상 클래스가 아닌, 추상 클래스와 인터페이스에 맞춰 코딩 가능

등 이 있으며

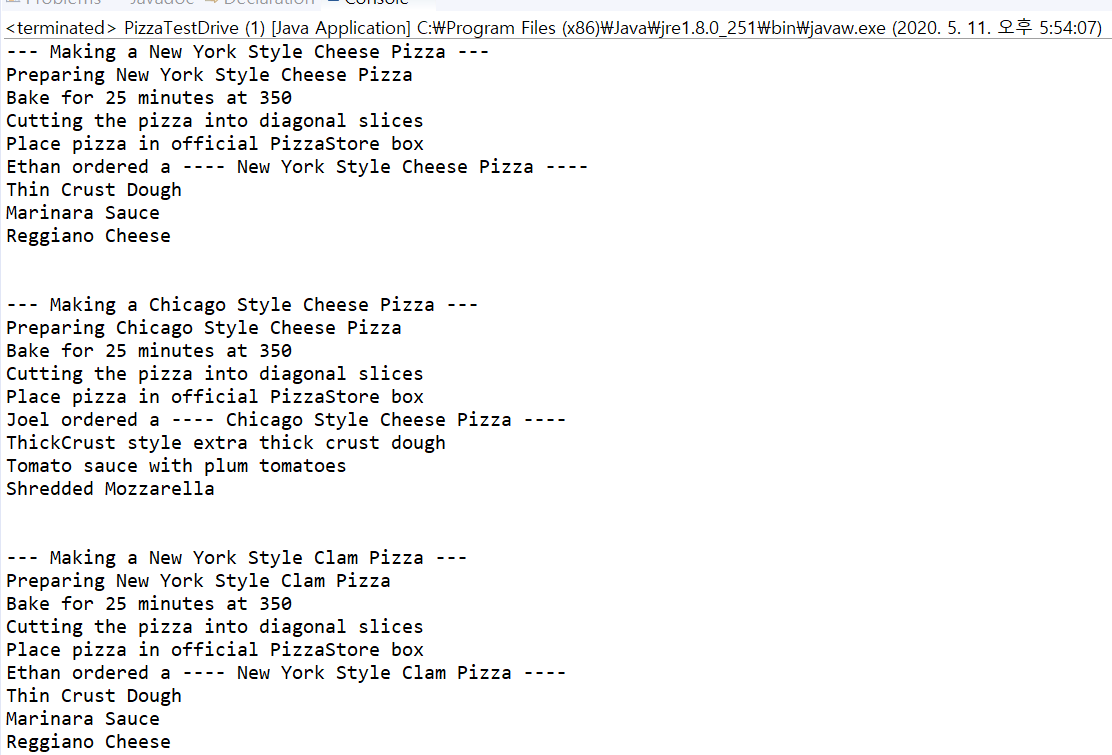
차이점으로는 팩토리 메소드 패턴의 경우 한 개의 메소드로 여러 개의 객체를 만들고 한 팩토리당 한 종류의 객체에 대한 생성을 지원하지만, 추상 팩토리 패턴의 경우 한 팩토리에서 서로 연관된 여러 종류를 모두 지원 하며 구상 클래스에 의존하지 않고 여러 개의 관련된 객체를 하나의 팩토리로 묶을 수 있다. 팩토리 메서드에서 만드는 객체의 종류 또한 팩토리 메소드 패턴의 경우 인자에 따라 객체의 종류가 결정 되지만 추상 팩토리 패턴의 경우 인자에 따라 관련된 객체들을 생성하는 팩토리의 종류가 결정된다.

정리 해보면

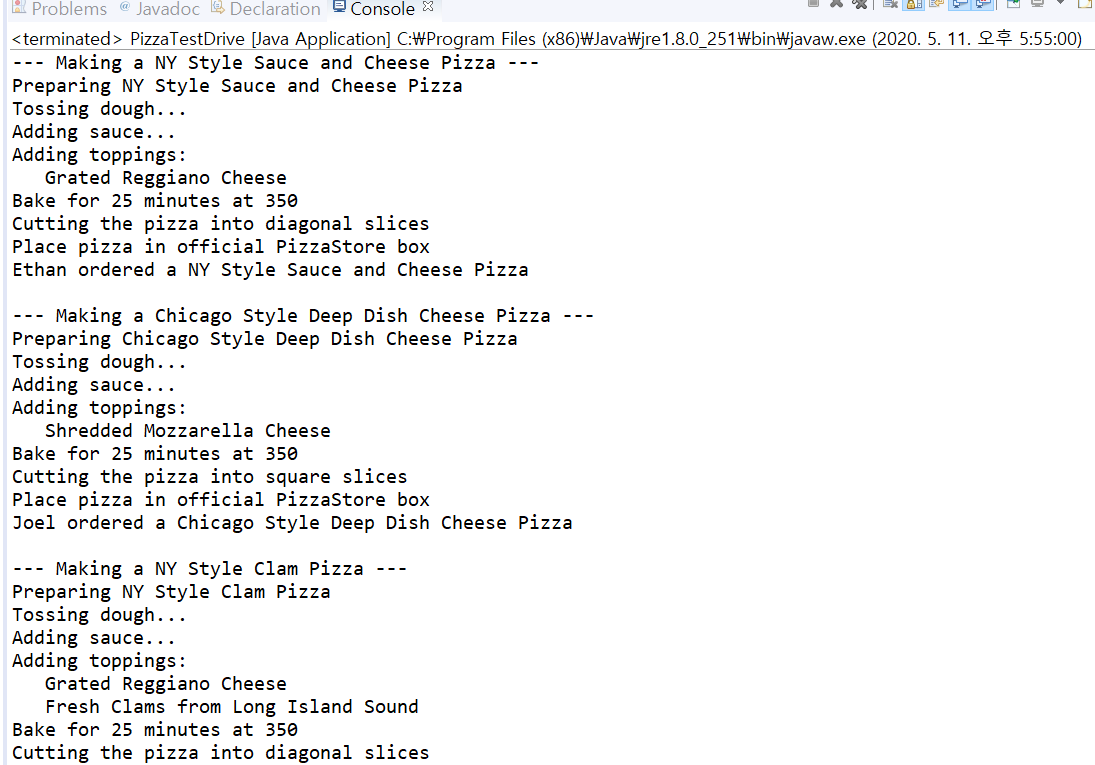
추상 팩토리 패턴은 제품군을 생성하기 위한 인터페이스를 생성하고 그 인터페이스를 구성하여 사용할수 있게 하는 것이고, 팩토리 메소드 패턴은 하나의 추상클래스에서 추상 메소드를 만들고 서브클래스들이 그 추상메소드를 구현하여 인스턴스를 생성하는 것이다.

1. 수행 결과

1 Pizzaaf



2 Pizzafm



정상적으로 프로그램이 수행되는 것을 확인 할 수 있다.