4. 팩토리 관련 패턴

팩토리 메소드 패턴 & 추상 팩토리 패턴

"new"의 문제점???

- 구상 클래스를 바탕으로 코딩 → 코드 수정 가능성↑, 유연성↓
- 코드에 구상 클래스를 많이 사용하면 새로운 구상 클래스가 추가될 때마다 코드를 고쳐야 하기 때문에 문제 발생 가능성이 있음.
  → 변화에 대해 닫혀 있는 코드가 될 수 있다.
- 바뀔 수 있는 부분을 찾아내서 바뀌지 않는 부분하고 분리시켜야 함. (encapsulation)

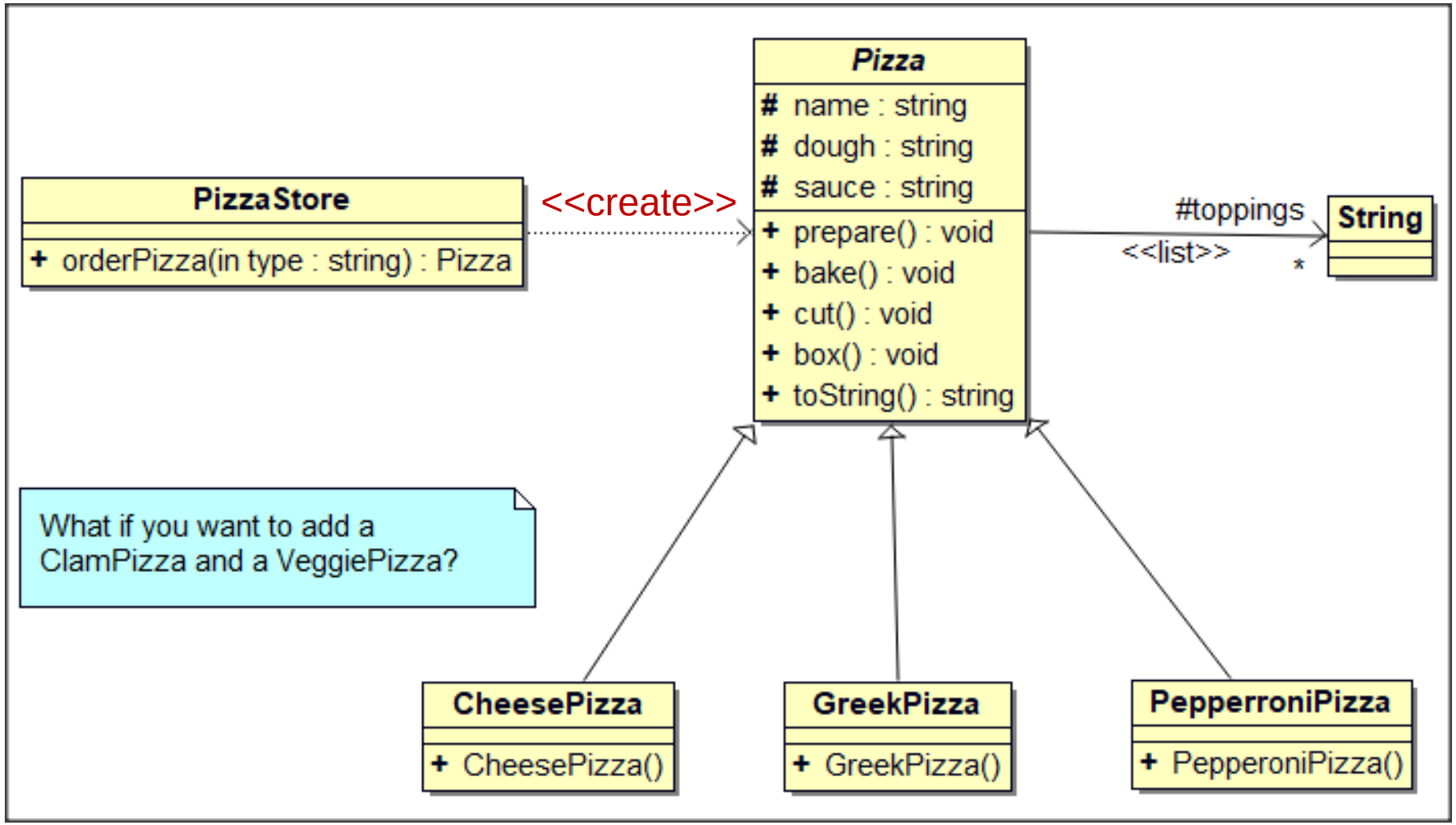
```
Duck duck;

if (picnic) {
    duck = new MallardDuck();
} else if (hunting) {
    duck = new DecoyDuck();
} else if (inBathTub) {
    duck = new RubberDuck();
}
```

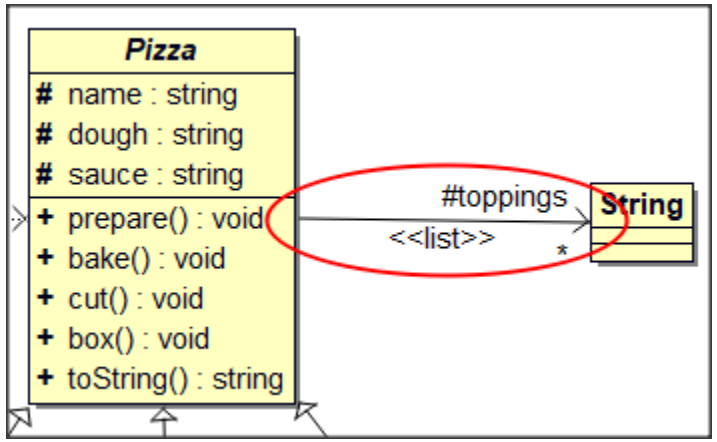
오리를 나타내는 클래스들이 여러 가지 있지만, 컴파일시에는 어떤 것의 인스턴스를 만들어야 할지 알 수 없습니다.

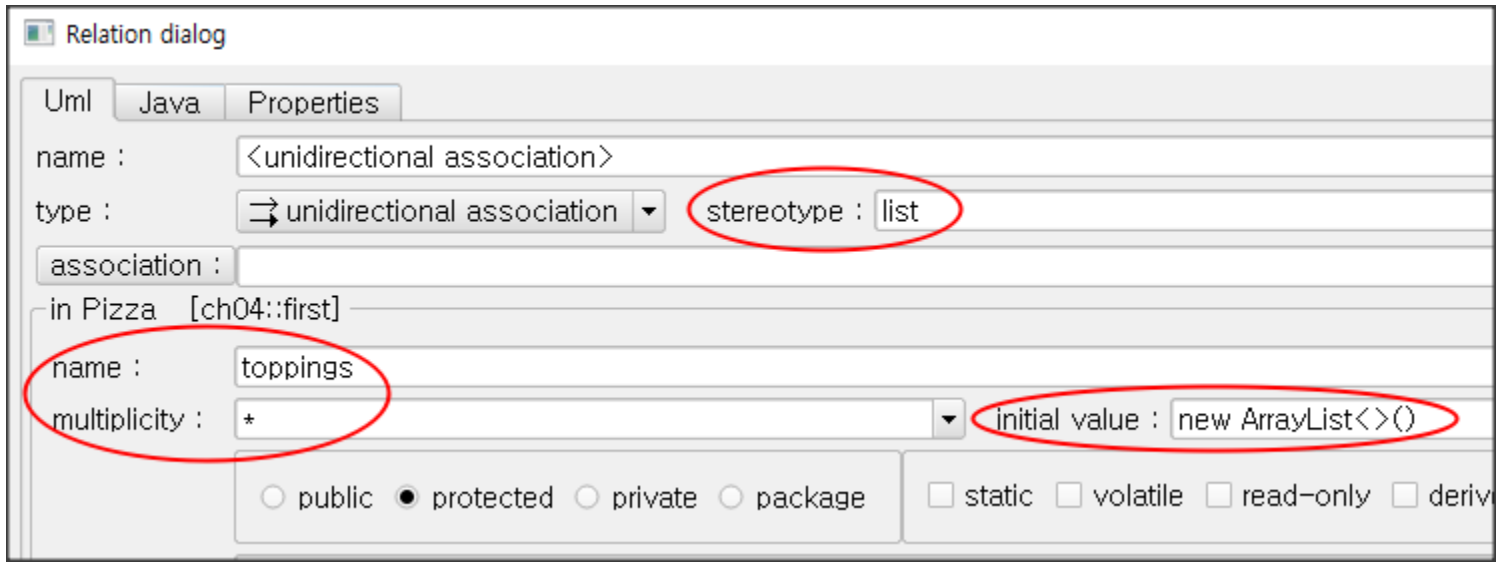
피자 가게

- 다음 조건을 만족시키는 클래스 설계를 하시오.
  - 피자 종류: 치즈 피자, 야채 피자, 페퍼로니 피자 등
  - 동일한 피자이더라도 피자 가게는 지역별로 조금씩 다른 피자를 만듦:
    뉴욕 스타일, 시카고 스타일, 캘리포니아 스타일 등

피자 가게: 첫 번째 설계 (first)




Pizza
name : string
dough : string
sauce : string
+ prepare() : void
+ bake() : void
+ cut() : void
+ box() : void
+ toString() : string
#toppings
<<list>>
*
String


Relation dialog
Uml
Java
Properties
name :
<unidirectional association>
type :
unidirectional association
stereotype :
list
association :
in Pizza [ch04::first]
name :
toppings
multiplicity :
*
initial value :
new ArrayList<>()
public
protected
private
package
static
volatile
read-only

코드 변경시 문제점

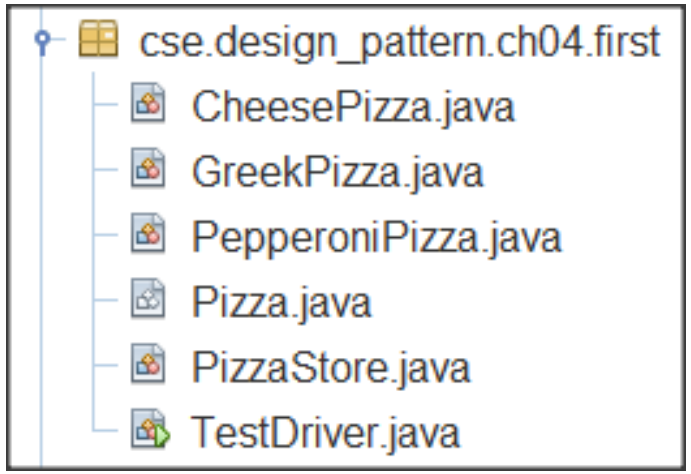
```
Pizza orderPizza(String type) {
    Pizza pizza;

    if (type.equals("cheese")) {
        pizza = new CheesePizza();
    } else if (type.equals("greek") {
        pizza = new GreekPizza();
    } else if (type.equals("pepperoni") {
        pizza = new PepperoniPizza();
    } else if (type.equals("clam") {
        pizza = new ClamPizza();
    } else if (type.equals("veggie") {
        pizza = new VeggiePizza();
    }

    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}
```

이 코드는 코드 변경에 대해 닫혀있지 않습니다. 피자 가게에서 메뉴를 변경하려면 이 코드를 직접 고쳐야 합니다.

이 부분이 바뀌는 부분입니다. 피자 종류가 바뀔 때마다 코드를 고쳐야 되겠죠...

이 부분은 바뀌지 않습니다. 피자를 굽고 베하고 자르고 포장하는 것은 피자를 판매하는 데 있어서 당연히 해야 하는 것이기 때문에, 이 코드는 거의 고칠 일이 없습니다. 어떤 피자 클래스의 메소드가 호출되는지만 달라질 뿐이죠.

4장: first


cse.design_pattern.ch04.first
CheesePizza.java
GreekPizza.java
PepperoniPizza.java
Pizza.java
PizzaStore.java
TestDriver.java

Pizza.java

```java
package cse.design_pattern.ch04.first;

import java.util.ArrayList;
import java.util.List;

public abstract class Pizza {

    protected List<String> toppings = new ArrayList<>();
    protected String name;
    protected String dough;
    protected String sauce;


    public void prepare() {
        System.out.println("Preparing " + name);
    }

    public void bake() {
        System.out.println("Baking " + name);
    }
```

```
21      public void cut() {
22          System.out.println("Cutting " + name);
23      }
24
25      public void box() {
26          System.out.println("Boxing " + name);
27      }
28
29      @Override
        public String toString() {
31          StringBuilder display = new StringBuilder();
32
            display.append("---- " + name + " ----₩n");
            display.append(dough + "₩n");
            display.append(sauce + "₩n");
36          for (int i = 0; i < toppings.size(); i++) {
                display.append(toppings.get(i) + "₩n");
38          }
39          return display.toString();
40      }
41  }
```

CheesePizza.java

```
1   package cse.design_pattern.ch04.first;
2
3   public class CheesePizza extends Pizza {
4
5       public CheesePizza() {
6           name = "Cheese Pizza";
7           dough = "Regular Crust";
8           sauce = "Marinara Pizza Sauce";
9           toppings.add("Fresh Mozzarella");
10          toppings.add("Parmesan");
11      }
12  }
```

GreekPizza.java

```
1   package cse.design_pattern.ch04.first;
2
3   public class GreekPizza extends Pizza {
4
5       public GreekPizza() {
6           name = "Greek Pizza";
7           dough = "Oily Crust";
8           sauce = "Tomato Sauce";
9           toppings.add("Feta Cheese");
10          toppings.add("Onion");
11          toppings.add("Olive");
12      }
13  }
```

PepperoniPizza.java

```java
package cse.design_pattern.ch04.first;

public class PepperoniPizza extends Pizza {

    public PepperoniPizza() {
        name = "Pepperoni Pizza";
        dough = "Crust";
        sauce = "Marinara sauce";
        toppings.add("Sliced Pepperoni");
        toppings.add("Sliced Onion");
        toppings.add("Grated parmesan cheese");
    }
}
```

PizzaStore.java

```
1   package cse.design_pattern.ch04.first;
2
3   public class PizzaStore {
4
5       public Pizza orderPizza(String type) {
6           Pizza pizza = null;
7
            if (type.equals("cheese")) {
9               pizza = new CheesePizza();
10          } else if (type.equals("greek")) {
11              // 더이상 그릭피자를 생산하지 않는다면?
12              pizza = new GreekPizza();
13          } else if (type.equals("pepperoni")) {
14              pizza = new PepperoniPizza();
15          }
```

```
16          // clam pizza와 veggie pizza를 생산해야 한다면?
17          // 구현에 대해 프로그래밍하여 코드 변경에
18          // 대해 닫혀 있지 않음.
19
20          // pizza가 제대로 생성되지 않으면?
21          // --> Null pointer dereference (널 포인터 역참조)
22          if (pizza != null) {
23              pizza.prepare();
24              pizza.bake();
25              pizza.cut();
26              pizza.box();
27          }
28
29          return pizza;
30      }
31
32  }
```

TestDriver.java

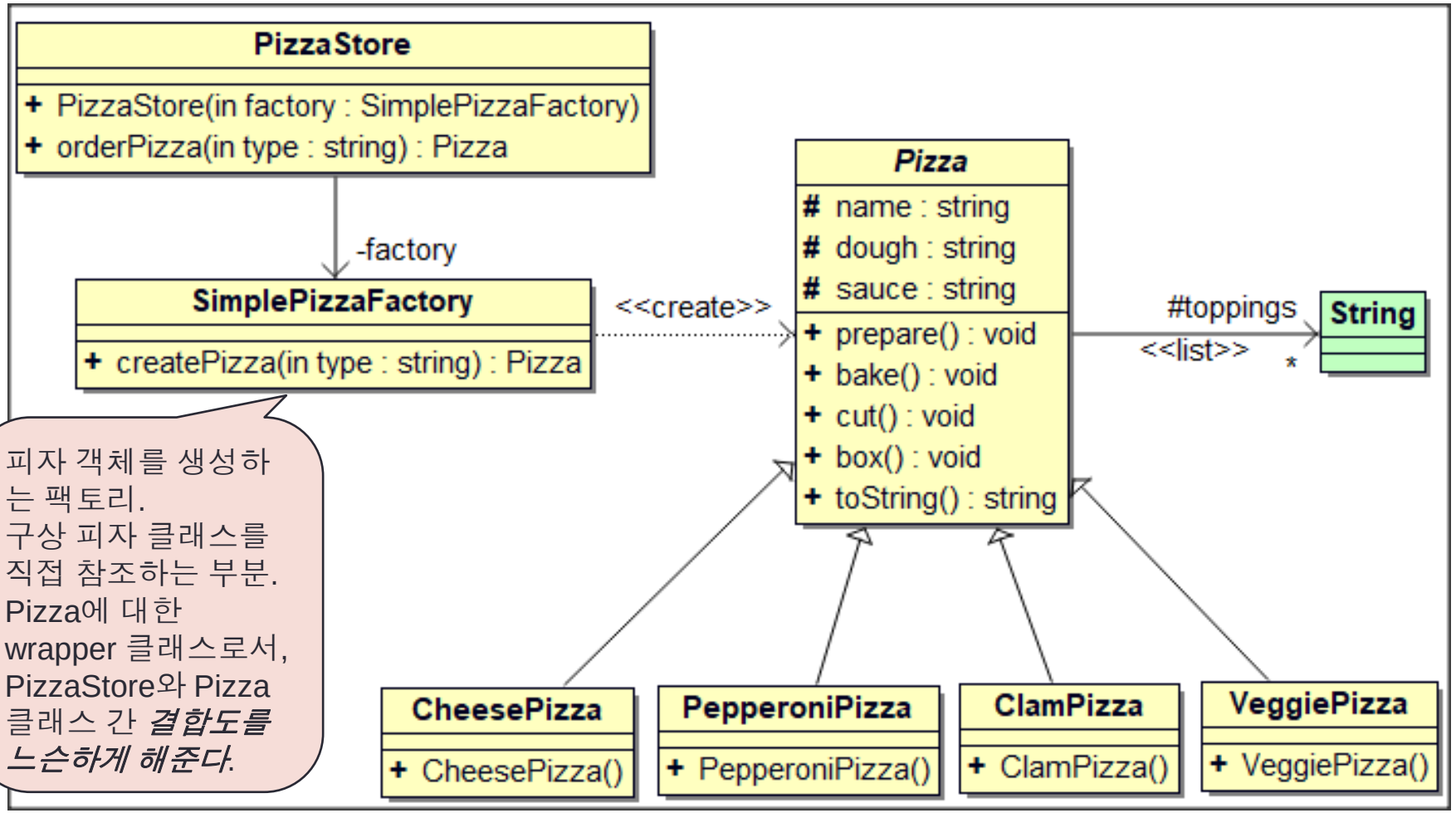
```
6   package cse.design_pattern.ch04.first;
7
8   /**...4 lines */
12  public class TestDriver {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          PizzaStore store = new PizzaStore();
19
20          Pizza pizza1 = store.orderPizza("cheese");
21          System.out.println(pizza1);
22
23          Pizza pizza2 = store.orderPizza("greek");
24          System.out.println(pizza2);
25
26          String pizzaName = "PEPPERONI";
27          Pizza pizza3 = store.orderPizza(pizzaName);
28          if (pizza3 != null) {  // 널 포인터 역참조 예방
29              System.out.println(pizza3);
30          } else {
31              System.out.printf("%s 피자는 없습니다.₩n", pizzaName);
32          }
33      }
34
35  }
```

실행 결과

```
--- exec-maven-plugin:1.5.0:e
Preparing Cheese Pizza
Baking Cheese Pizza
Cutting Cheese Pizza
Boxing Cheese Pizza
---- Cheese Pizza ----
Regular Crust
Marinara Pizza Sauce
Fresh Mozzarella
Parmesan

Preparing Greek Pizza
Baking Greek Pizza
Cutting Greek Pizza
Boxing Greek Pizza
---- Greek Pizza ----
Oily Crust
Tomato Sauce
Feta Cheese
Onion
Olive

PEPPERONI 피자는 없습니다.
```

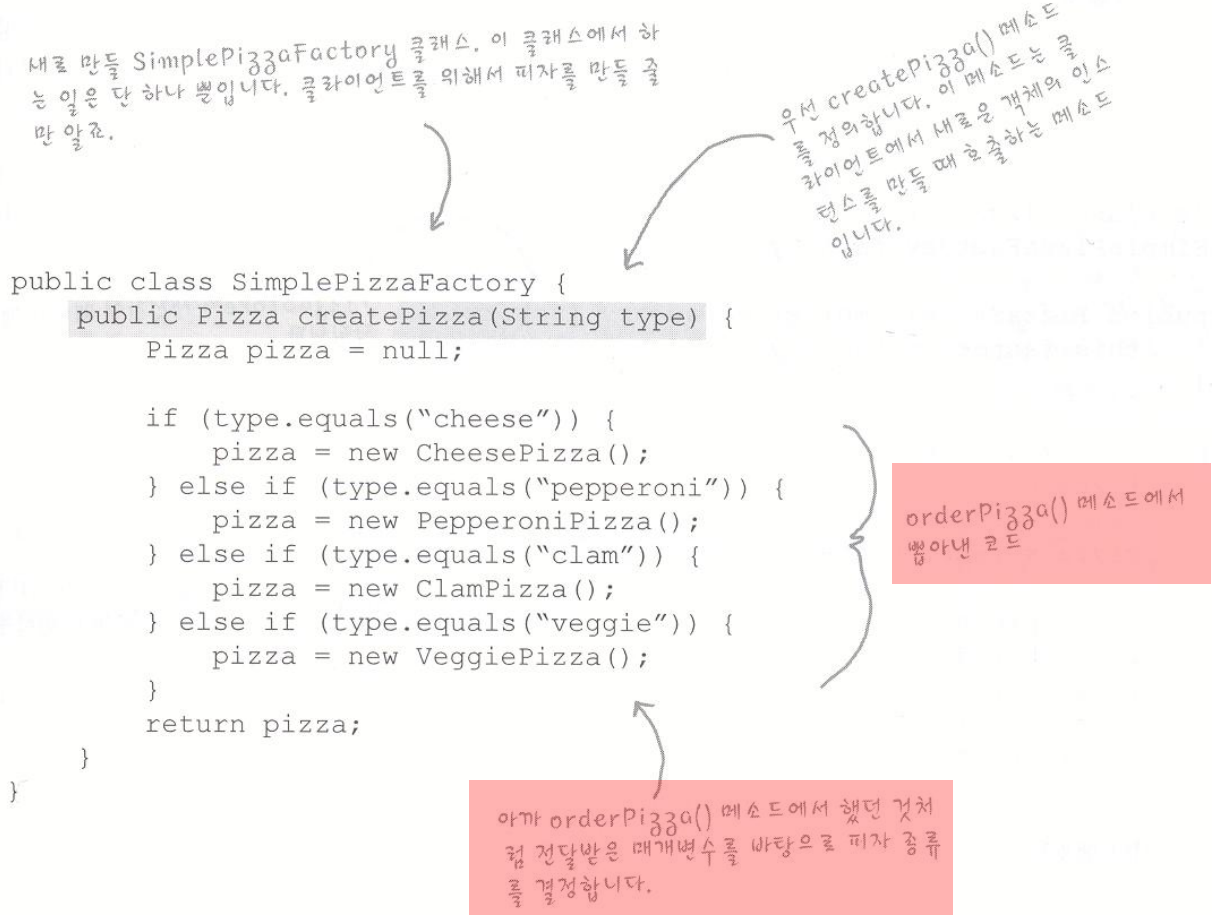
피자 가게: 두 번째 설계(simple_factory)

- 해결 방법: 객체 생성 부분의 캡슐화



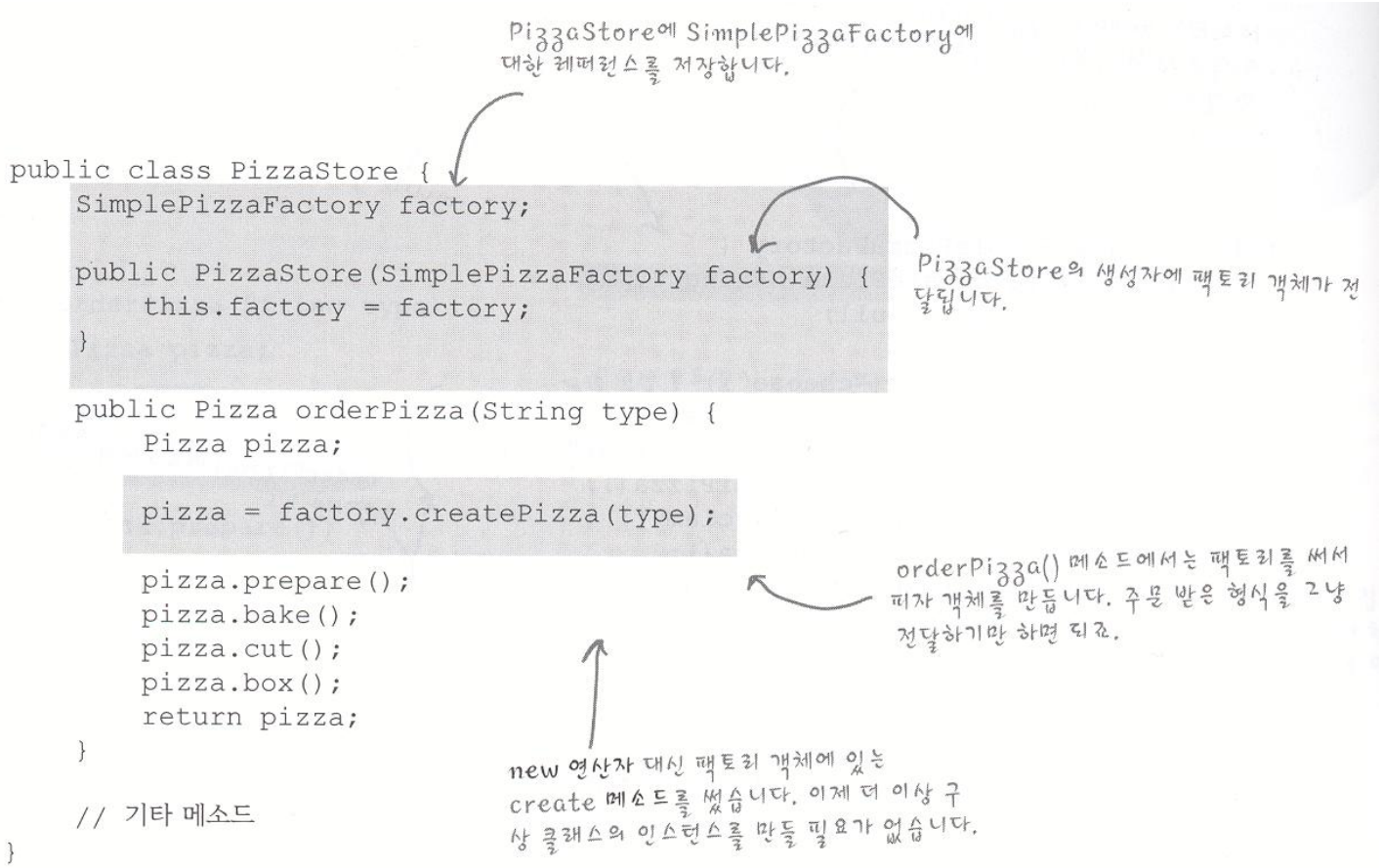
피자 생성 클래스

- 간단한 팩토리(Simple Factory)는 디자인 패턴이라고 할 수 없음.
- 프로그래밍을 하는 데 있어서 자주 사용되는 관용구라 할 수 있음.



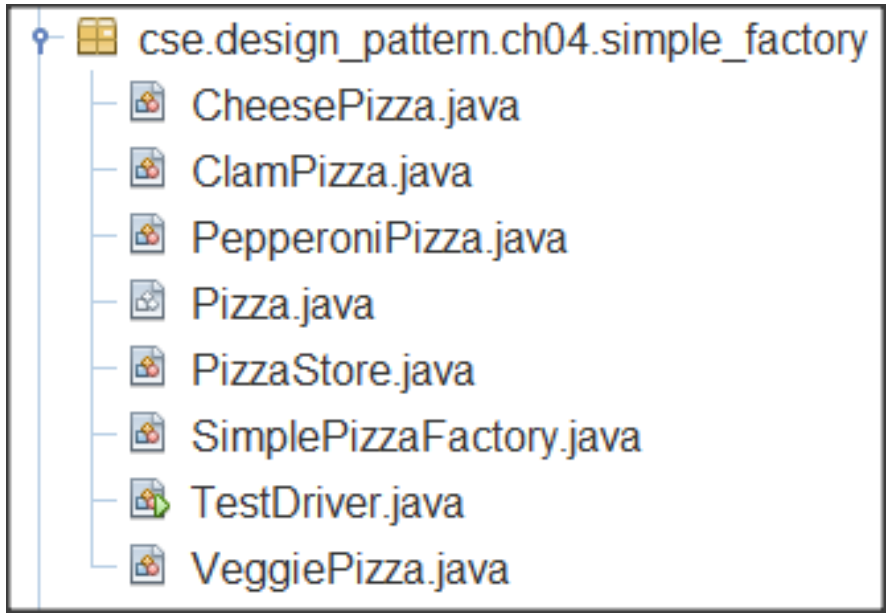
PizzaStore 클래스

- 문제점: 지역별로 다른 스타일을 반영할 수 없다.
  → SimplePizzaFactory를 제거하고 이를 대체할 수 있는 NYPizzaFactory, ChicagoPizzaFactory, CaliforniaPizzaFactory를 만들면 된다.




```
public class PizzaStore {
    SimplePizzaFactory factory;

    public PizzaStore(SimplePizzaFactory factory) {
        this.factory = factory;
    }

    public Pizza orderPizza(String type) {
        Pizza pizza;

        pizza = factory.createPizza(type);

        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();
        return pizza;
    }

    // 기타 메소드
}
```

4장: simple_factory



Pizza.java

```java
package cse.design_pattern.ch04.first;

import java.util.ArrayList;
import java.util.List;

public abstract class Pizza {

    protected List<String> toppings = new ArrayList<>();
    protected String name;
    protected String dough;
    protected String sauce;


    public void prepare() {
        System.out.println("Preparing " + name);
    }

    public void bake() {
        System.out.println("Baking " + name);
    }

    public void cut() {
        System.out.println("Cutting " + name);
    }

    public void box() {
        System.out.println("Boxing " + name);
    }
```

```
30      @Override
        public String toString() {
32          StringBuilder display = new StringBuilder();
33
            display.append("---- " + name + " ----₩n");
            display.append(dough + "₩n");
            display.append(sauce + "₩n");
37          for (int i = 0; i < toppings.size(); i++) {
                display.append(toppings.get(i) + "₩n");
39          }
40          return display.toString();
41      }
42  }
```

CheesePizza.java

```java
package cse.design_pattern.ch04.simple_factory;

public class CheesePizza extends Pizza {

    public CheesePizza() {
        name = "Cheese Pizza";
        dough = "Regular Crust";
        sauce = "Marinara Pizza Sauce";
        toppings.add("Fresh Mozzarella");
        toppings.add("Parmesan");
    }
}

```

VeggiePizza.java

```java
package cse.design_pattern.ch04.simple_factory;

public class VeggiePizza extends Pizza {

    public VeggiePizza() {
        name = "Veggie Pizza";
        dough = "Crust";
        sauce = "Marinara sauce";
        toppings.add("Shredded mozzarella");
        toppings.add("Grated parmesan");
        toppings.add("Diced onion");
        toppings.add("Sliced mushrooms");
        toppings.add("Sliced red pepper");
        toppings.add("Sliced black olives");
    }
}
```

ClamPizza.java

```java
package cse.design_pattern.ch04.simple_factory;

public class ClamPizza extends Pizza {

    public ClamPizza() {
        name = "Clam Pizza";
        dough = "Thin crust";
        sauce = "White garlic sauce";
        toppings.add("Clams");
        toppings.add("Grated parmesan cheese");
    }
}
```

PepperoniPizza.java

```java
package cse.design_pattern.ch04.simple_factory;

public class PepperoniPizza extends Pizza {

    public PepperoniPizza() {
        name = "Pepperoni Pizza";
        dough = "Crust";
        sauce = "Marinara sauce";
        toppings.add("Sliced Pepperoni");
        toppings.add("Sliced Onion");
        toppings.add("Grated parmesan cheese");
    }
}
```

SimplePizzaFactory.java

```
1  package cse.design_pattern.ch04.simple_factory;
2
3  public class SimplePizzaFactory {
4
5      public Pizza createPizza(String type) {
6          Pizza pizza = null;
7
9          if (type.equals("cheese")) {
10             pizza = new CheesePizza();
11         } else if (type.equals("pepperoni")) {
12             pizza = new PepperoniPizza();
13         } else if (type.equals("clam")) {
14             pizza = new ClamPizza();
15         } else if (type.equals("veggie")) {
16             pizza = new VeggiePizza();
17         }
18         return pizza;
19     }
   }
```

PizzaStore.java

```java
package cse.design_pattern.ch04.simple_factory;

public class PizzaStore {

    private SimplePizzaFactory factory;

    public PizzaStore(SimplePizzaFactory factory) {
        this.factory = factory;
    }

    public Pizza orderPizza(String type) {
        Pizza pizza;

        pizza = factory.createPizza(type);

        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();

        return pizza;
    }

}
```

TestDriver.java

```
6   package cse.design_pattern.ch04.simple_factory;
7
8   /**...4 lines */
12  public class TestDriver {
13
14      public static void main(String[] args) {
15          SimplePizzaFactory factory = new SimplePizzaFactory();
16          PizzaStore store = new PizzaStore(factory);
17
18          Pizza pizza = store.orderPizza("cheese");
19          System.out.println("Ethan ordered a " + pizza.name + "₩n");
20
21          pizza = store.orderPizza("veggie");
22          System.out.println("Joel  ordered a " + pizza.name + "₩n");
23      }
24  }
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec
Preparing Cheese Pizza
Baking Cheese Pizza
Cutting Cheese Pizza
Boxing Cheese Pizza
Ethan ordered a Cheese Pizza

Preparing Veggie Pizza
Baking Veggie Pizza
Cutting Veggie Pizza
Boxing Veggie Pizza
Joel  ordered a Veggie Pizza
```

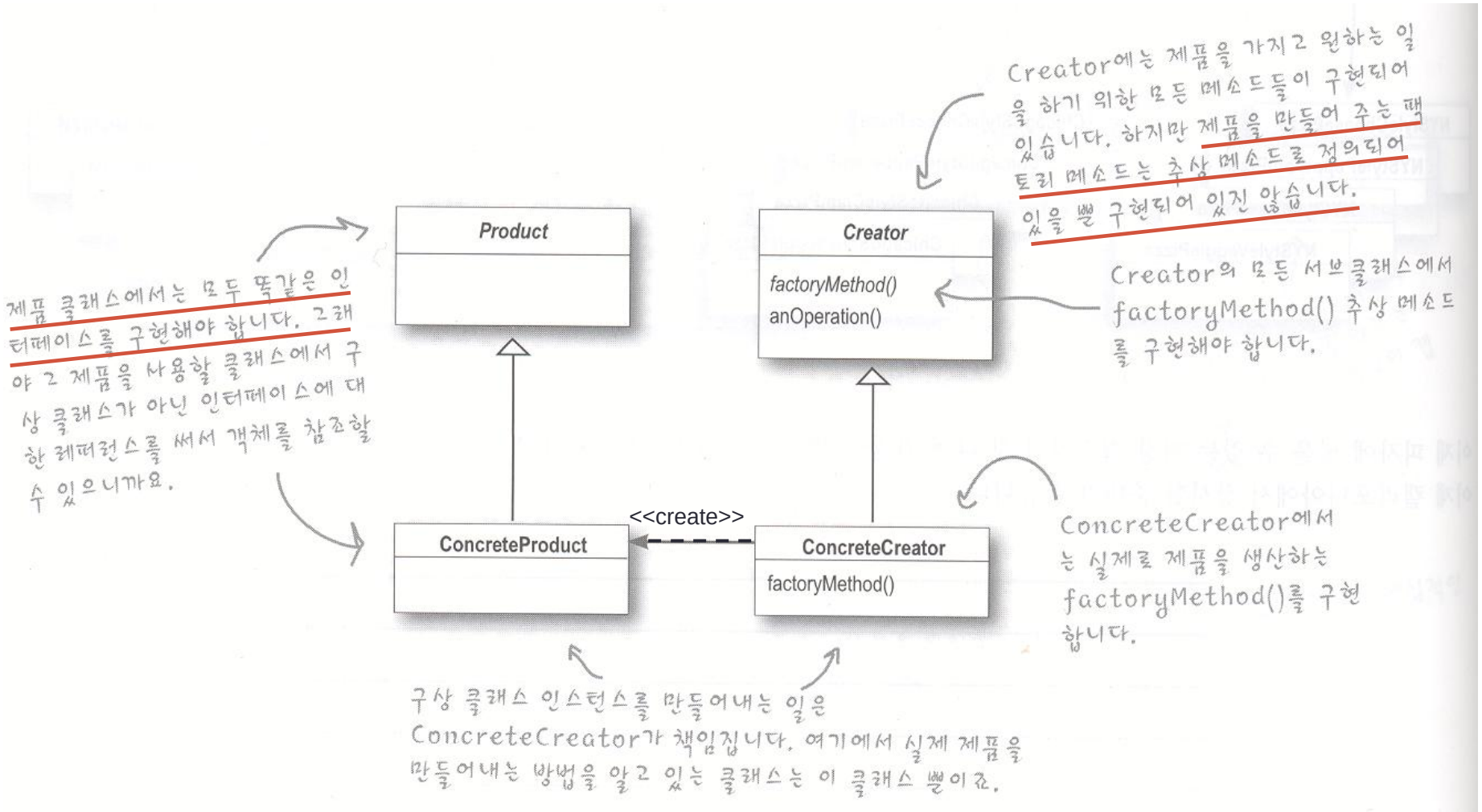
팩토리 메소드 패턴 적용

Factory Method Pattern

- It deals with the problem of creating objects (products) without specifying the exact class of object that will be created.
- (참고1) https://refactoring.guru/ko/design-patterns/factory-method
- (참고2) https://www.phind.com/search?cache=q0cgf88lkg0v5fdl9xf3gycm (define the factory method pattern in terms of a problem and a solution.)

팩토리 메서드 패턴은 객체 생성 프로세스를 중앙 집중화하여 서브 클래스가 인스턴스화할 클래스를 결정할 수 있도록 합니다. 이것은 객체를 만들기 위한 인터페이스(또는 추상 클래스)를 정의하고 서브 클래스가 인스턴스화할 클래스를 결정하도록 함으로써 달성됩니다. 패턴에는 다음이 포함됩니다.

- Creator 클래스: 객체를 만들기 위한 인터페이스를 정의하지만, 서브 클래스가 생성될 객체의 유형을 변경할 수 있도록 합니다.
- Concrete Creators: 팩토리 메서드를 구현하여 특정 유형의 개체를 만듭니다.
- Products: 팩토리 메서드로 만든 개체입니다.
- Client: Creator 클래스를 사용하여 생성될 제품의 구체적인 클래스를 모른 채 개체를 가져옵니다.


Product
Creator
factoryMethod()
anOperation()
ConcreteProduct
<<create>>
ConcreteCreator
factoryMethod()
제품 클래스에서는 모두 똑같은 인터페이스를 구현해야 합니다. 그래야 그 제품을 사용할 클래스에서 구상 클래스가 아닌 인터페이스에 대한 레퍼런스를 써서 객체를 참조할 수 있으니까요.
Creator에는 제품을 가지고 원하는 일을 하기 위한 모든 메소드들이 구현되어 있습니다. 하지만 제품을 만들어 주는 팩토리 메소드는 추상 메소드로 정의되어 있을 뿐 구현되어 있진 않습니다.
Creator의 모든 서브클래스에서 factoryMethod() 추상 메소드를 구현해야 합니다.
ConcreteCreator에서는 실제로 제품을 생산하는 factoryMethod()를 구현합니다.
구상 클래스 인스턴스를 만들어내는 일은 ConcreteCreator가 책임집니다. 여기에서 실제 제품을 만들어내는 방법을 알고 있는 클래스는 이 클래스 뿐이죠.

p.168

chosun.com 프린트 닫기

미셸, 시카고와 뉴욕 '피자 싸움' 불붙여

연합뉴스
입력 : 2010.03.28 08:38

자세히보기 GO ▸

김현 통신원 = 시카고 출신의 퍼스트 레이디 미셸 오바마가 정말 "뉴욕 피자가 최고"라고 말했는지 여부를 두고 시카고 사람들이 촉각을 곤두세웠다.

시카고 트리뷴에 따르면 미셸 오바마는 지난 24일, 봄방학을 맞은 두 딸 사샤와 말리아, 친정 어머니 마리안 로빈슨 여사 등과 함께 뉴욕에서 뮤지컬 공연을 관람한 뒤 점심을 먹기 위해 브루클린 다리 인근의 전통적인 뉴욕 피자집 '그리말디스'를 찾았다.

오바마 일행은 치즈피자, 페퍼로니피자, 소시지피자와 버섯, 페퍼로니, 양파를 올린 피자 등을 주문했고, '퍼스트 패밀리'를 맞은 그리말디스 직원들은 감격에 겨워 서빙을 했다.

오바마 일행으로부터 직접 주문을 받고 테이블 서빙을 맡았던 라팔 하라자는 "너무 특별한 손님들을 맞아 처음엔 매우 긴장했으나 미셸 오바마의 따뜻한 배려 덕분에 곧 편안해졌다"면서 "즐겁게 서빙했고, 매우 넉넉한 팁도 받았다"고 밝혔다.

문제는 뉴욕 타임스(NYT)가 자사 블로그에 하라자의 말을 인용, "미셸 오바마가 그리말디스 피자를 가리키며 '시카고 피자보다 더 낫다'고 말했다"고 전하면서 촉발됐다.

인터넷 정치 웹사이트 허핑턴포스트는 "오랫동안 지속되어온 뉴욕과 시카고 사이의 피자 자존심 대결에 다시 불이 붙었다"고 전했다.

얇은 크러스트에 다양한 토핑을 올리는 전형적인 뉴욕 피자와 달리 깊은 크러스트를 가득 채운 두꺼운 치즈 위에 독특한 토마토 소스를 올리는 시카고 피자는 대중화 면에서는 뉴욕 피자에 뒤져있으나 이에 대한 시카고 사람들의 자부심은 유난하다. 그리말디스의 사장 프랭크 시올리는 시카고 트리뷴과의 통화에서 "미셸 오바마가 뉴욕 피자를 시카고 피자와 비교해 더 낫다고 표현하지는 않았다"면서 "미셸 오바마는 단지 '피자가 정말 최고였다. 나는 시카고 출신이다'라고 말했다"고 정정했지만 뉴욕과 시카고 두 도시 간의 피자 논쟁은 계속되고 있다.

허핑턴포스트는 웹사이트를 통해 "뉴욕과 시카고 중 어느 도시가 더 나은 피자를 갖고 있다고 믿는가?"에 대한 설문조사를 진행하고 있다.

27일 현재 뉴욕은 52.88%, 시카고는 47.12%의 지지를 얻고 있다.

Lombardi's

http://www.firstpizza.com/

Zachary's Chicago Pizza - OAKLAND Restaurant

5801 College Ave.
Oakland CA 94618
Phone: (510) 655-6385
Hours: Sun-Thur 11AM-10PM
Fri-Sat 11AM-10:30PM

One block north of Rockridge BART station - Get here on BART

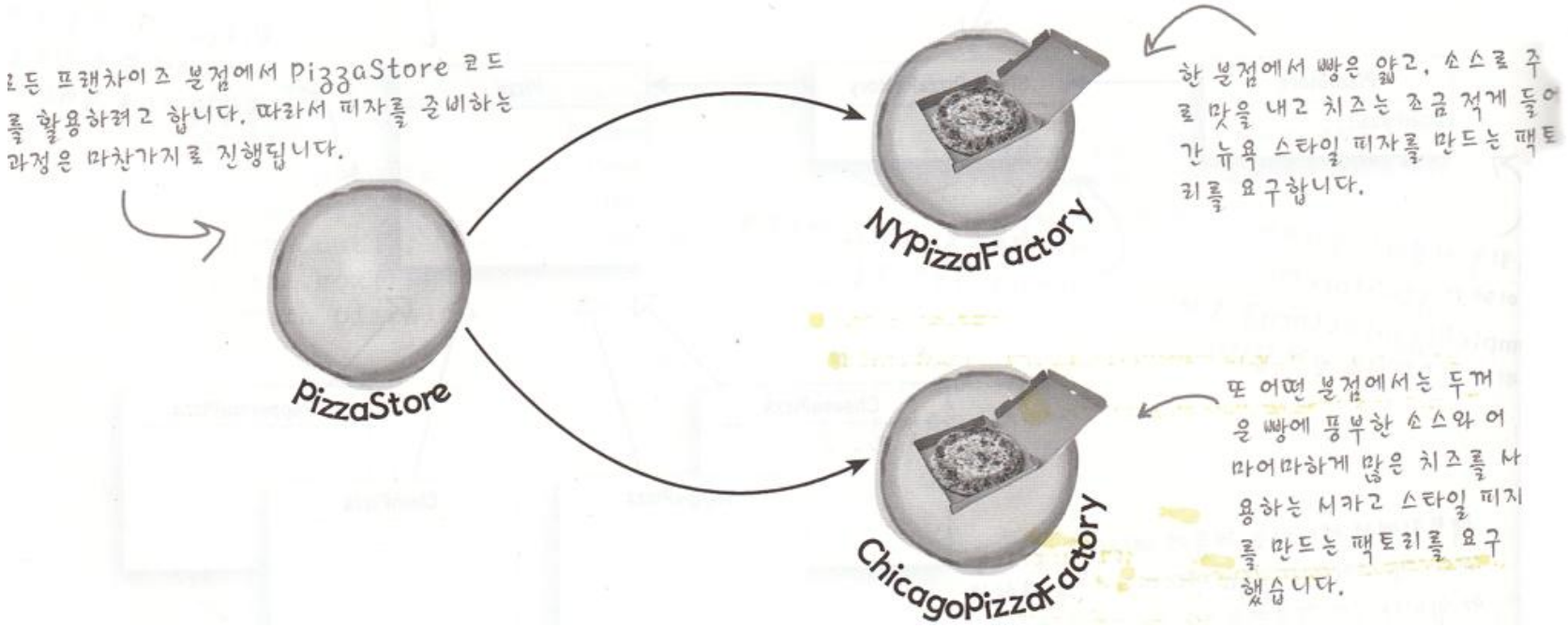
OAKLAND seating is first come, first served. If restaurant is full, you can order your pizzas once you get an estimated wait time for your table. Your pizzas will cook while you wait! More

Zachary's Oakland photos below

Zachary's STUFFED Pizza

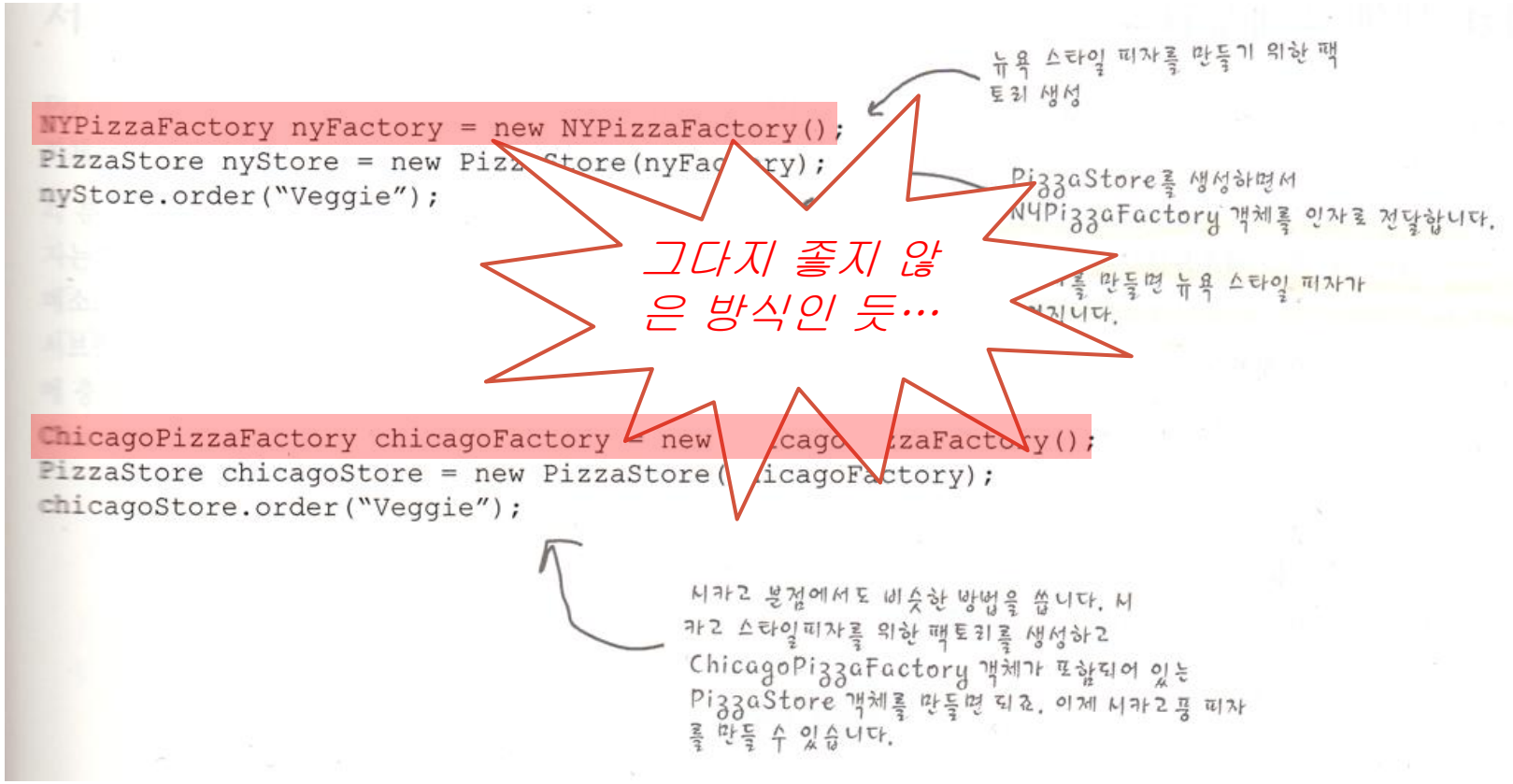
피자 프랜차이즈 사업

- 지역별로 다른 스타일의 피자(뉴욕 스타일, 시카고 스타일 등)를 만들어야 함.



피자 종류별 팩토리 사용

- 피자 종류가 다르면 피자 가게의 피자 제작 과정도 다르지 않을까?
  → 피자 가게와 피자 제작 과정 전체를 하나로 묶어주는 framework가 필요



해결 방법: 피자 가게 프레임워크

- PizzaStore 클래스에 createPizza() 메소드를 다시 넣고 하위 클래스에서 구현하도록 함.

이제 PizzaStore는 추상 클래스가 됩니다. (왜 그런지는 밑에서 알 수 있습니다)

```
public abstract class PizzaStore {

    public Pizza orderPizza(String type) {
        Pizza pizza;

        pizza = createPizza(type);

        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();

        return pizza;
    }

protected abstract Pizza createPizza(String type);
}
```

팩토리 객체가 아닌 PizzaStore에 있는 createPizza를 호출하게 됩니다.

나머지는 똑같습니다.

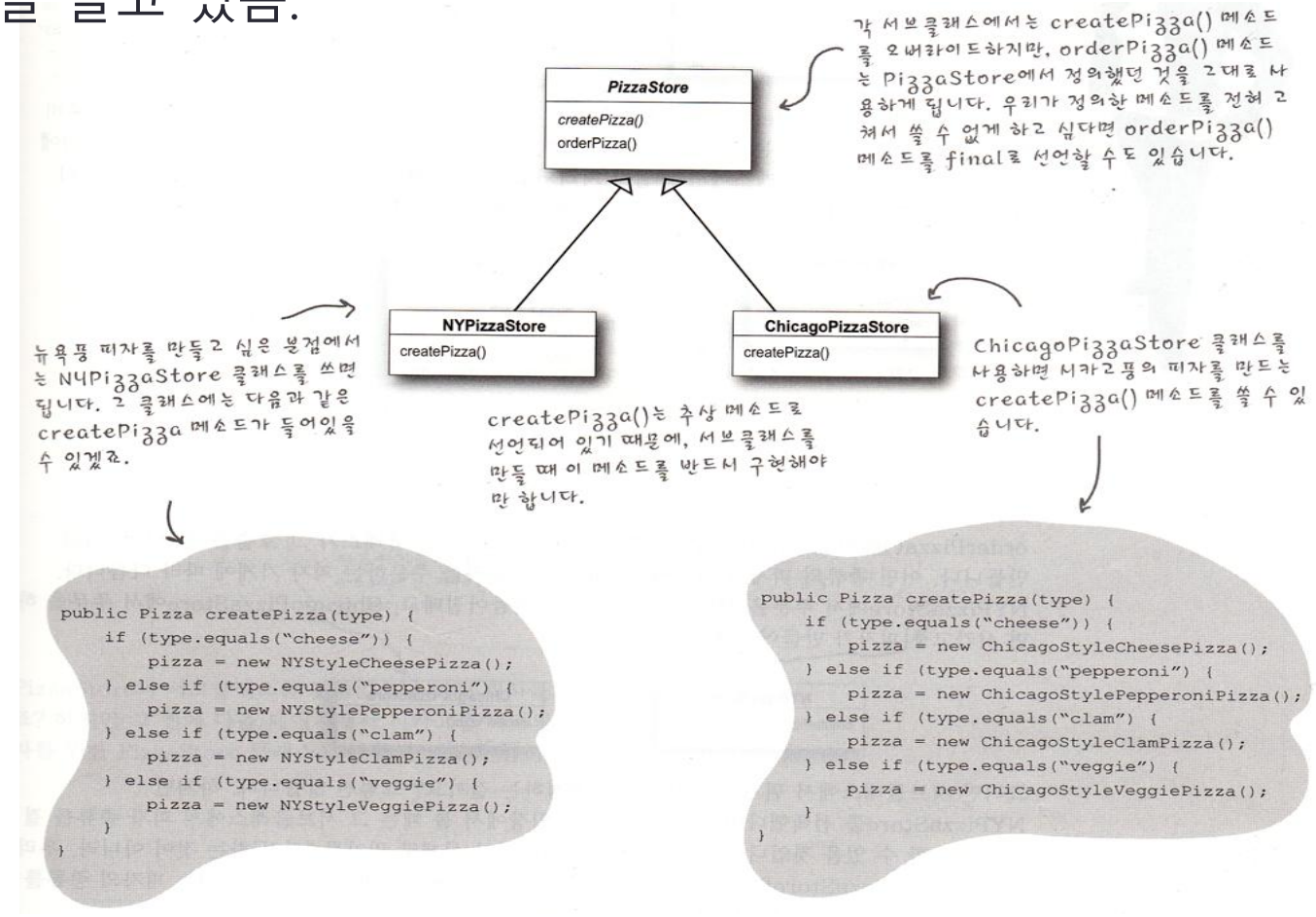
팩토리 객체 대신 이 메소드를 사용합니다.

이제 "팩토리 메소드(factory method)"가 PizzaStore의 추상 메소드로 바뀌었습니다.

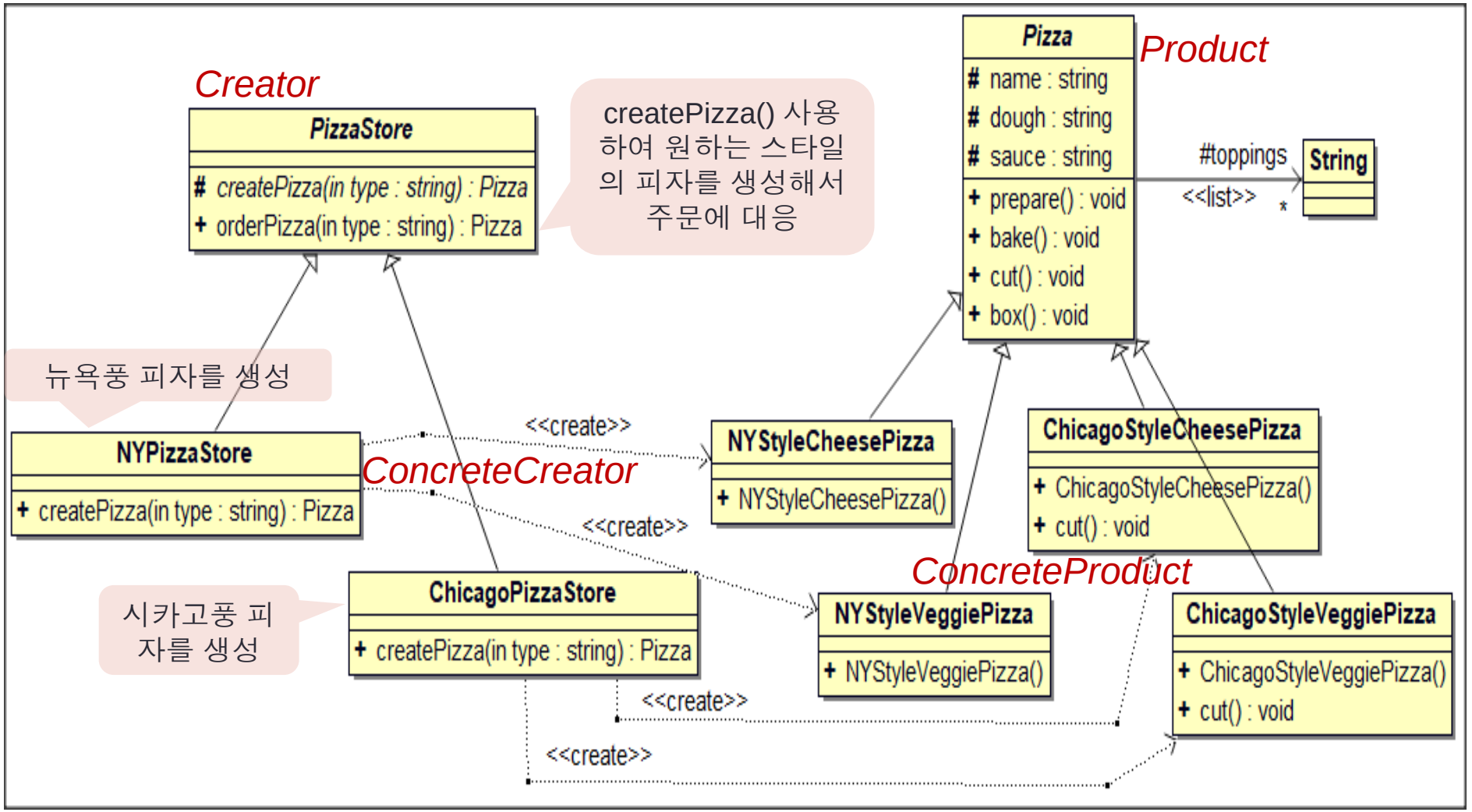
Simple factory에서 구현되었던 부분이 PizzaStore (Creator)의 추상 operation 됨.

p.158

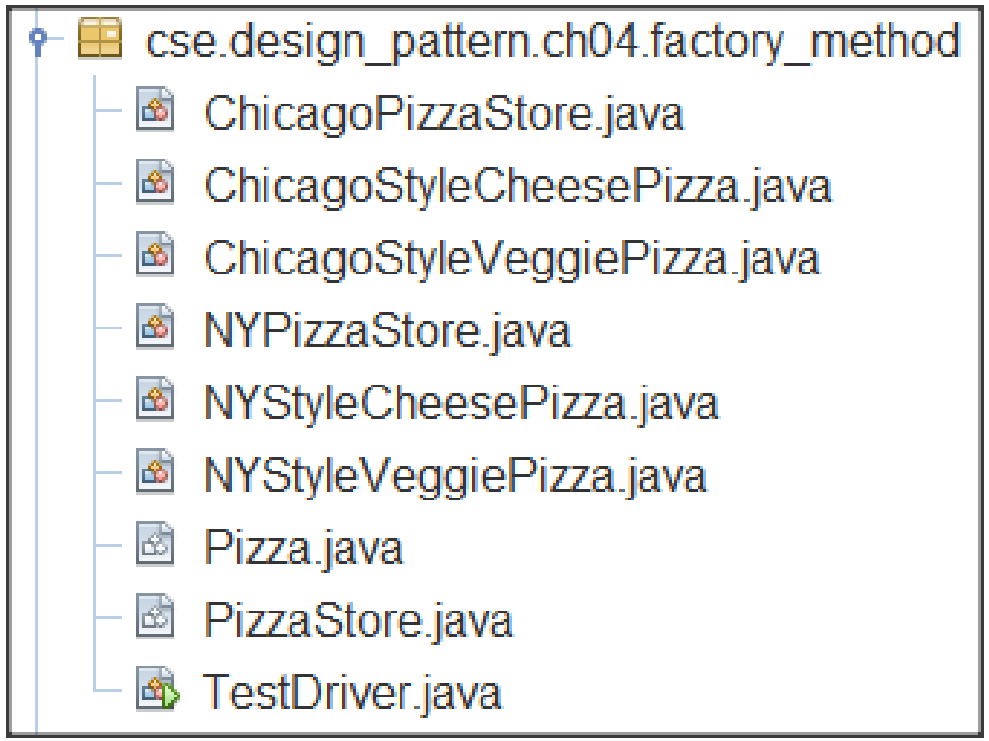
- PizzaStore 클래스의 하위 클래스에서 createPizza() 메소드를 구현
  → 실제 피자 만드는 방법은 뉴욕 피자 가게, 시카고 피자 가게 등에서 더 잘 알고 있음.



피자 가게: 세 번째 설계 (factory_method)



4장: factory_method



PizzaStore.java

DIP: PizzaStore 추상 클래스는 Pizza 추상 클래스에 의존

```
1  package cse.design_pattern.ch04.factory_method;
2
3  public abstract class PizzaStore {
4
5      public abstract Pizza createPizza(String type);
6
7      public Pizza orderPizza(String type) {
8          Pizza pizza = createPizza(type);
9          System.out.println("--- Making a " + pizza.getName() + " ---");
10         pizza.prepare();
11         pizza.bake();
12         pizza.cut();
13         pizza.box();
14         return pizza;
15     }
16
17 }
```

팩토리 메소드이나, 추상 오퍼레이션임. subclass에서 객체 생성을 책임져야 함.

팩토리 메소드를 사용하여 피자 객체를 만들고 최종적으로 박스에 담을 때까지의 과정을 기술

어떤 피자가 만들어지는지 알 수 없다. 즉, 실제로 생성되는 구상 객체가 무엇인지 알 수 없게 한다. *신기하게도 이 상태로 해도 컴파일 가능하다!!!*

NYPizzaStore.java

```
1   package cse.design_pattern.ch04.factory_method;
2
3   public class NYPizzaStore extends PizzaStore {
4
5       public Pizza createPizza(String type) {
6           if (type.equals("cheese")) {
7               return new NYStyleCheesePizza();
8           } else if (type.equals("veggie")) {
9               return new NYStyleVeggiePizza();
10          } else {
11              return null;
12          }
13      }
14
15  }
```

ChicagoPizzaStore.java

```
1   package cse.design_pattern.ch04.factory_method;
2
3   public class ChicagoPizzaStore extends PizzaStore {
4
5       public Pizza createPizza(String type) {
6           if (type.equals("cheese")) {
7               return new ChicagoStyleCheesePizza();
8           } else if (type.equals("veggie")) {
9               return new ChicagoStyleVeggiePizza();
10          } else {
11              return null;
12          }
13      }
14
15  }
```

Pizza.java

```java
package cse.design_pattern.ch04.factory_method;

import java.util.ArrayList;
import java.util.List;

public abstract class Pizza {

    protected List<String> toppings = new ArrayList<>();
    protected String name;
    protected String dough;
    protected String sauce;

    public void prepare() {
        System.out.println("Preparing " + name);
        System.out.println("Tossing dough...");
        System.out.println("Adding sauce...");
        System.out.println("Adding toppings: ");
        toppings.forEach(topping -> {
            System.out.println("   " + topping);
        });
    }
```

```
23      public void bake() {
24          System.out.println("Bake for 25 minutes at 350");
25      }
26
28      public void cut() {
29          System.out.println("Cutting the pizza into diagonal slices");
30      }
31
32      public void box() {
33          System.out.println("Place pizza in official PizzaStore box");
34      }
35
36      public String getName() {
37          return name;
38      }
39
41      @Override
42      public String toString() {
43          StringBuilder display = new StringBuilder();
44          // display.append("---- " + name + " ----₩n");
45          display.append("---- ").append(name).append(" ----₩n");
46          display.append(dough).append("₩n");
47          display.append(sauce).append("₩n");
48          toppings.forEach(topping -> {
49              display.append(topping).append("₩n");
50          });
51          return display.toString();
      }
  }
```

NYStyleCheesePizza.java

```
1   package cse.design_pattern.ch04.factory_method;
2
3   public class NYStyleCheesePizza extends Pizza {
4
5       public NYStyleCheesePizza() {
6           name = "NY Style Sauce and Cheese Pizza";
7           dough = "Thin Crust Dough";
8           sauce = "Marinara Sauce";
9
10          toppings.add("Grated Reggiano Cheese");
11      }
12  }
```

DIP: NYStyleCheesePizza 구상 클래스는 Pizza 추상 클래스를 상속받아 의존

NYStyleVeggiePizza.java

```
1   package cse.design_pattern.ch04.factory_method;
2
3   public class NYStyleVeggiePizza extends Pizza {
4
5       public NYStyleVeggiePizza() {
6           name = "NY Style Veggie Pizza";
7           dough = "Thin Crust Dough";
8           sauce = "Marinara Sauce";
9
10          toppings.add("Grated Reggiano Cheese");
11          toppings.add("Garlic");
12          toppings.add("Onion");
13          toppings.add("Mushrooms");
14          toppings.add("Red Pepper");
15      }
16  }
```

DIP: NYStyleVeggiePizza 구상 클래스는 Pizza 추상 클래스를 상속받아 의존

ChicagoStyleCheesePizza.java

```java
package cse.design_pattern.ch04.factory_method;

public class ChicagoStyleCheesePizza extends Pizza {

    public ChicagoStyleCheesePizza() {
        name = "Chicago Style Deep Dish Cheese Pizza";
        dough = "Extra Thick Crust Dough";
        sauce = "Plum Tomato Sauce";

        toppings.add("Shredded Mozzarella Cheese");
    }

    public void cut() {
        System.out.println("Cutting the pizza into square slices");
    }

}
```

DIP: ChicagoStyleCheesePizza 구상 클래스는 Pizza 추상 클래스를 상속받아 의존

ChicagoStyleVeggiePizza.java

```java
package cse.design_pattern.ch04.factory_method;

public class ChicagoStyleVeggiePizza extends Pizza {

    public ChicagoStyleVeggiePizza() {
        name = "Chicago Deep Dish Veggie Pizza";
        dough = "Extra Thick Crust Dough";
        sauce = "Plum Tomato Sauce";

        toppings.add("Shredded Mozzarella Cheese");
        toppings.add("Black Olives");
        toppings.add("Spinach");
        toppings.add("Eggplant");
    }

    public void cut() {
        System.out.println("Cutting the pizza into square slices");
    }
}
```

DIP: ChicagoStyleVeggiePizza 구상 클래스는 Pizza 추상 클래스를 상속받아 의존

TestDriver.java

```
6   package cse.design_pattern.ch04.factory_method;
7
8   /**
9    *
10   * @author Prof.Jong Min Lee (my_account AT test DOT com)
11   */
12  public class TestDriver {
13
14    public static void main(String[] args) {
15      PizzaStore nyStore = new NYPizzaStore();
16      PizzaStore chicagoStore = new ChicagoPizzaStore();
17
18      Pizza pizza = nyStore.orderPizza("cheese");
19      System.out.println("Ethan ordered a " + pizza.getName() + "₩n");
20
21      pizza = chicagoStore.orderPizza("cheese");
22      System.out.println("Joel ordered a " + pizza.getName() + "₩n");
23
24      pizza = nyStore.orderPizza("veggie");
25      System.out.println("Ethan ordered a " + pizza.getName() + "₩n");
26
27      pizza = chicagoStore.orderPizza("veggie");
28      System.out.println("Joel ordered a " + pizza.getName() + "₩n");
29    }
30  }
```

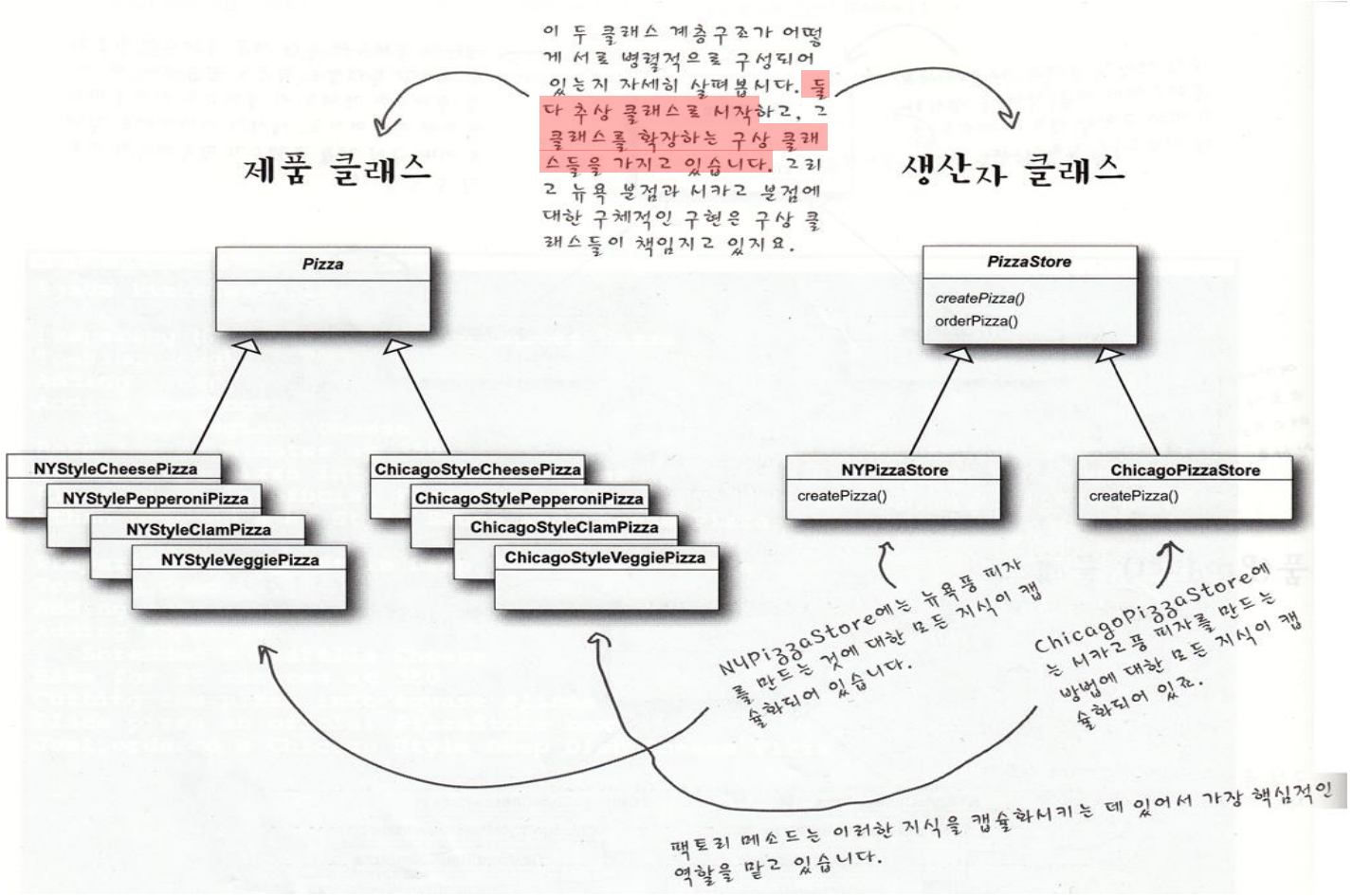
실행 결과

```
--- Making a NY Style Sauce and Cheese Pizza ---
Preparing NY Style Sauce and Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Grated Reggiano Cheese
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a NY Style Sauce and Cheese Pizza

--- Making a Chicago Style Deep Dish Cheese Pizza ---
Preparing Chicago Style Deep Dish Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Shredded Mozzarella Cheese
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Style Deep Dish Cheese Pizza
```

```
--- Making a NY Style Veggie Pizza ---
Preparing NY Style Veggie Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Grated Reggiano Cheese
   Garlic
   Onion
   Mushrooms
   Red Pepper
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a NY Style Veggie Pizza

--- Making a Chicago Deep Dish Veggie Pizza ---
Preparing Chicago Deep Dish Veggie Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Shredded Mozzarella Cheese
   Black Olives
   Spinach
   Eggplant
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Deep Dish Veggie Pizza
```

병렬 클래스 계층 구조

- 제품에 관한 지식을 각 생산자에 캡슐화하는 방법



p.170

DIP: 의존성 뒤집기 원칙

- Dependency Inversion Principle (DIP)
  - 추상화된 것에 의존하도록 만들어라.
  - 구상 클래스에 의존하도록 만들지 않도록 한다.
  - 구상 클래스에 대한 의존성을 줄이기 위해 뒤집었다는 의미를 알려주기 위해 이름을 DIP라고 함.

- 고수준 구성요소가 저수준 구성요소에 의존하면 안 된다.
  - 고수준 구성요소: PizzaStore (추상 Pizza 클래스를 사용)
  - 저수준 구성요소: 피자 객체들

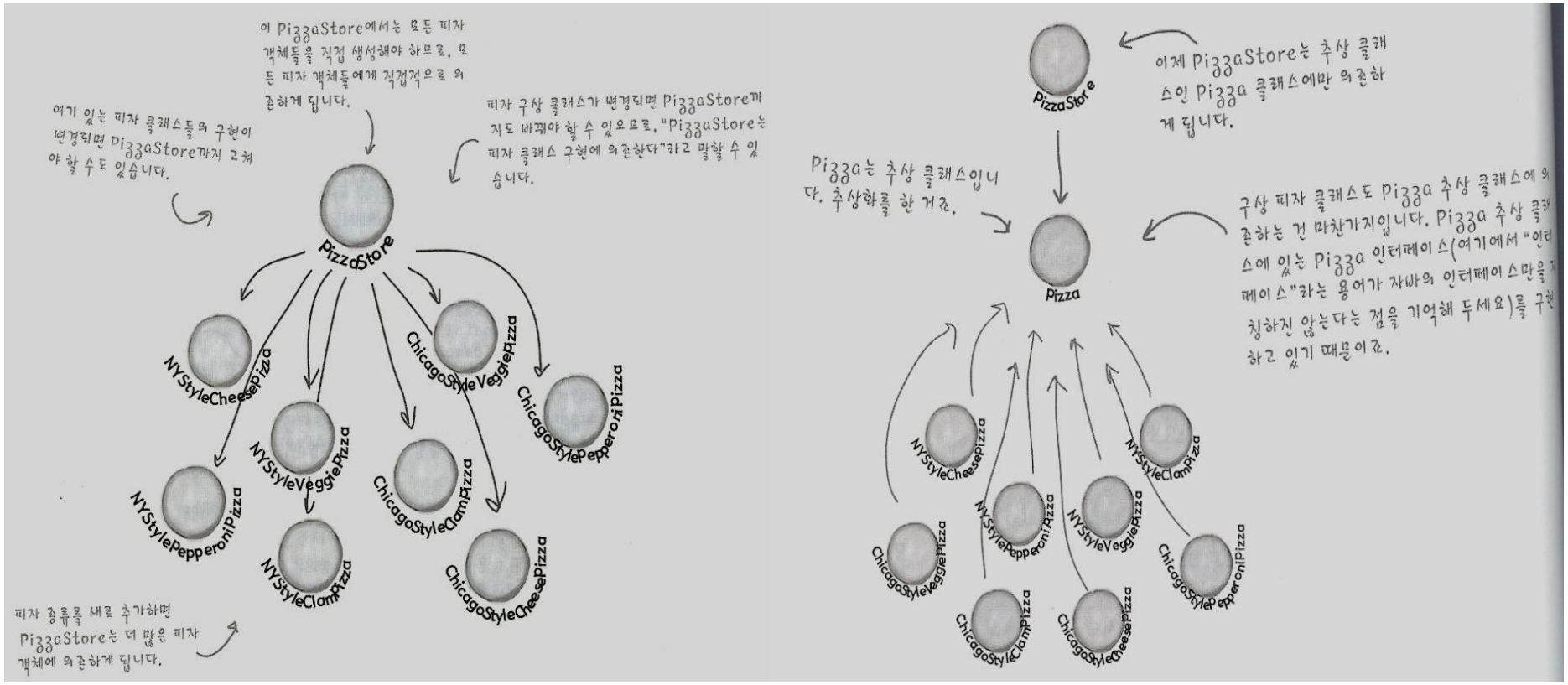
Dependency inversion principle states that

- High level modules should not depend on low-level modules, both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

Or it can be rephrases as “the dependencies should be based on abstractions, not details.”

객체 의존성 예

- 구상클래스에 많이 의존 DIP 적용



One should “depend upon abstractions, [not] concretions.”

Dependency Inversion Principle

- High level or low level modules should not depend upon each other, instead they should depend upon abstractions.
- You do not have to develop the lower level modules before developing the higher level modules. Thus a top-down approach is possible in both design and development.
- 장점
  - The result is that components are less tightly coupled, and there is a high degree of separation of concerns.
  - The individual components are more easily testable and the higher level objects can be tested with mock objects in place of lower level services.

참고1) https://en.wikipedia.org/wiki/Dependency_inversion_principle

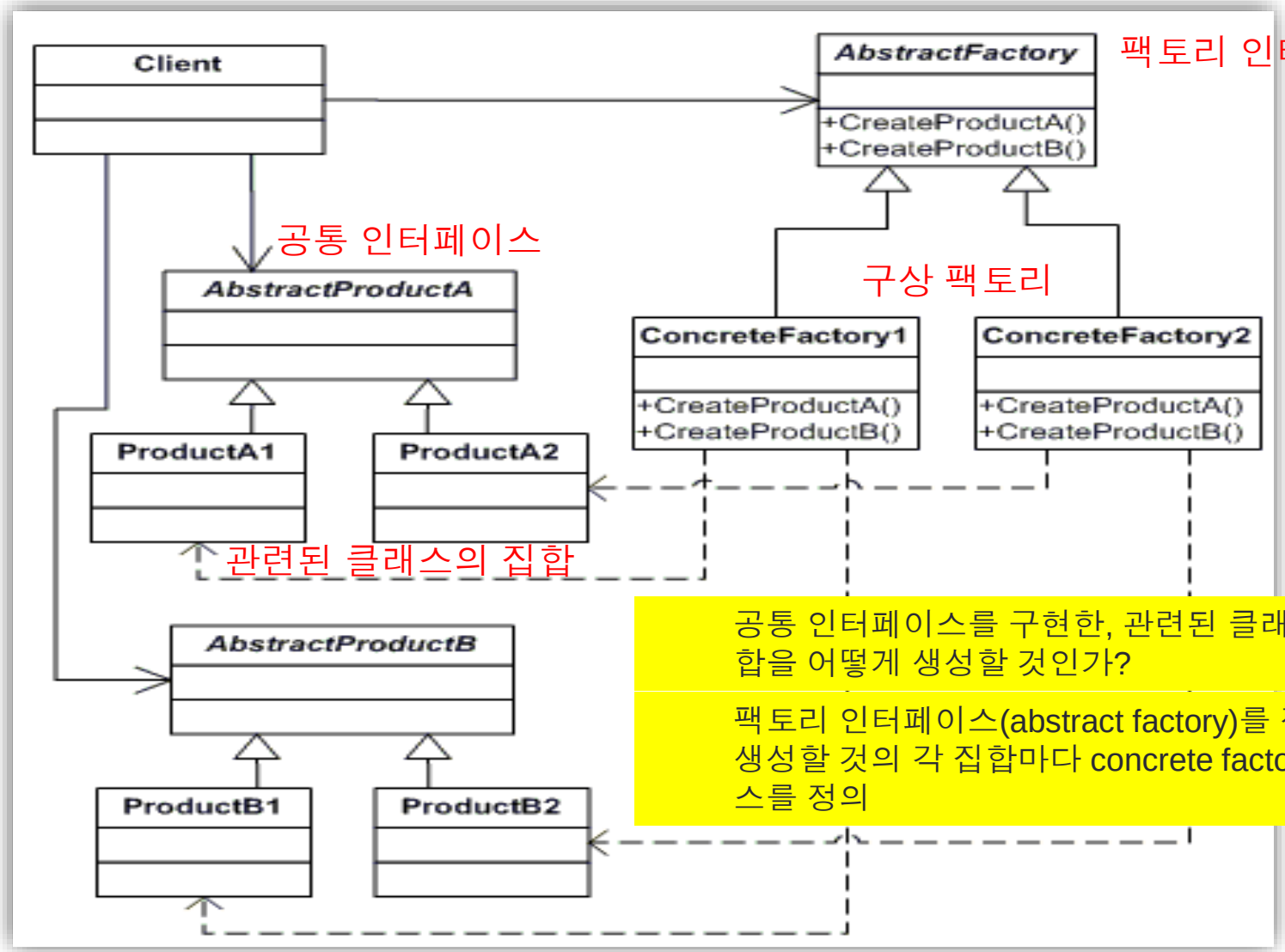
참고2) https://www.phind.com/search?cache=j3e69mrtts50nfnioxyy9tql (IoC 과 DIP 차이점)

추상 팩토리 패턴 적용 설계

추상 팩토리 패턴

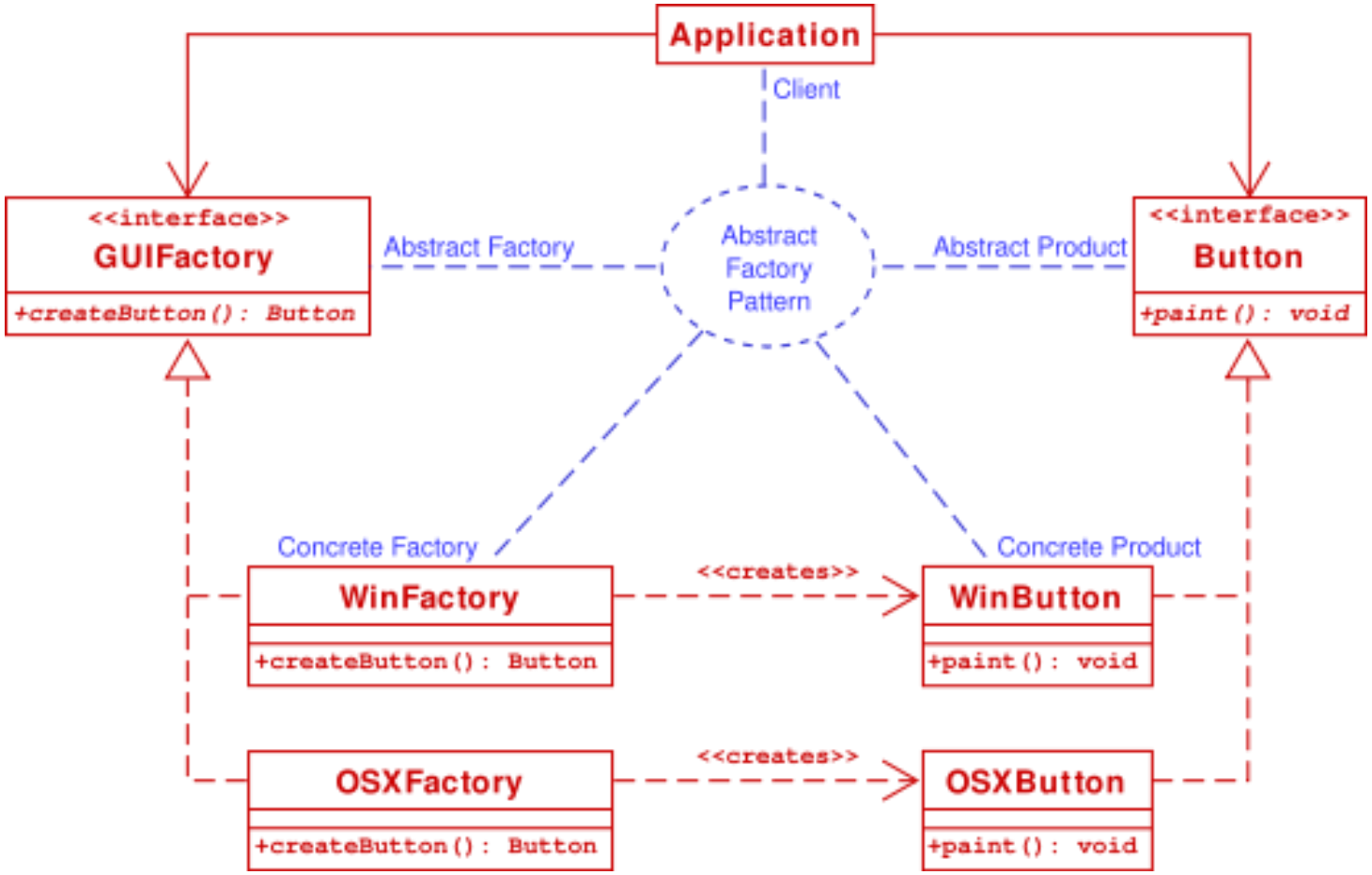
- cf. http://www.dofactory.com/Patterns/PatternAbstract.aspx
- 정의
  - Provide *an interface for creating families of related or dependent objects* without specifying their concrete classes.
  - 인터페이스를 이용하여 서로 연관된, 또는 의존하는 객체를 구상 클래스를 지정하지 않고 생성할 수 있다.
- 클라이언트와 팩토리에서 생산되는 제품을 분리시킬 수 있다.

- 문제점
  - 공통 인터페이스를 구현한, 관련된 클래스의 집합을 어떻게 생성할 것인가?
- 해결책
  - 팩토리 인터페이스(abstract factory)를 정의하라. 생성할 것의 각 집합마다 concrete factory 클래스를 정의하라. 선택적으로 팩토리 인터페이스를 구현하고 이를 확장한 concrete factory에게 공통 서비스를 제공하는 실제 추상 클래스를 정의하라.

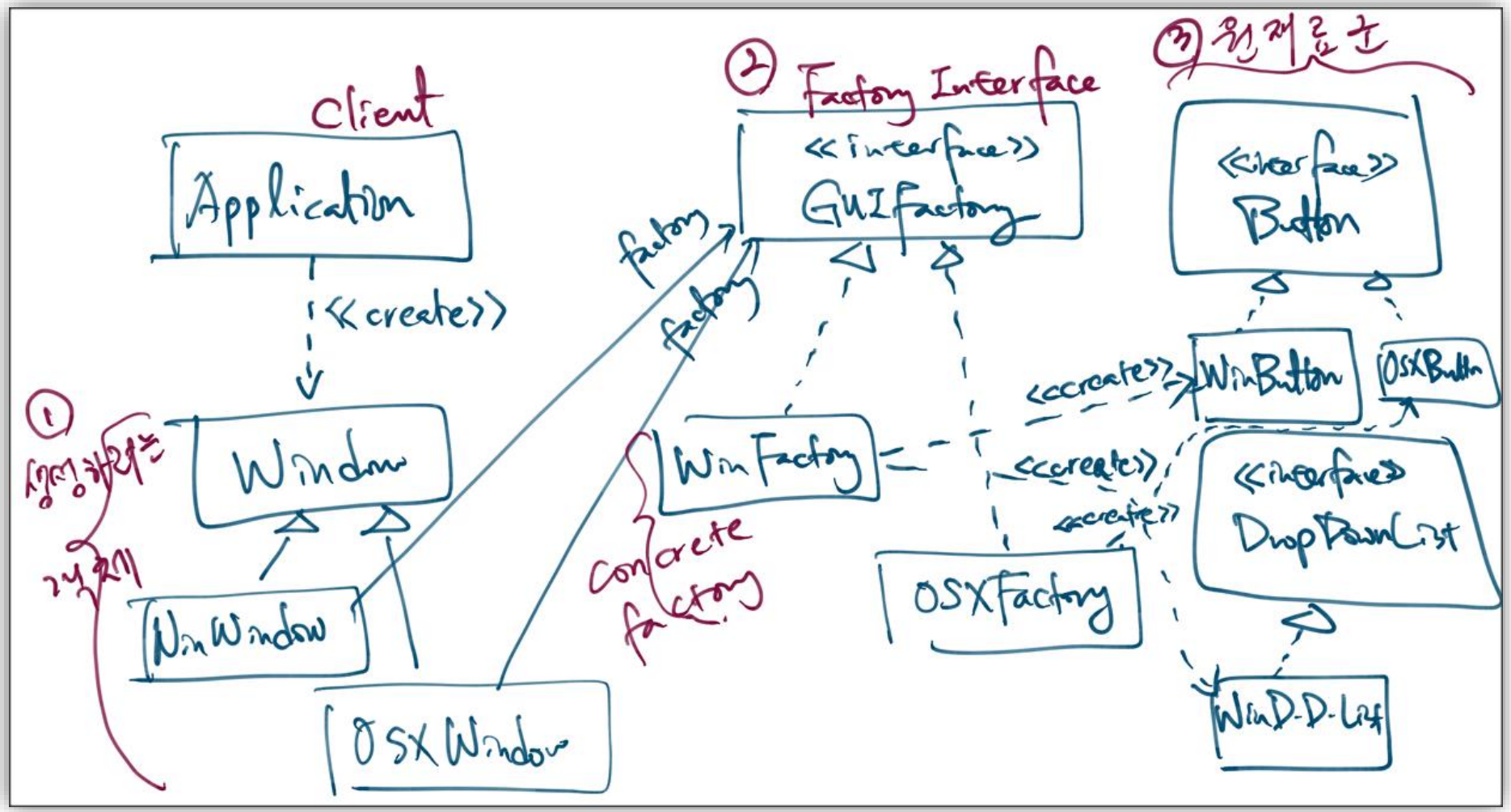

Client
AbstractFactory
+CreateProductA()
+CreateProductB()
팩토리 인터페이스
공통 인터페이스
구상 팩토리
AbstractProductA
ConcreteFactory1
+CreateProductA()
+CreateProductB()
ConcreteFactory2
+CreateProductA()
+CreateProductB()
ProductA1
ProductA2
관련된 클래스의 집합
AbstractProductB
ProductB1
ProductB2
공통 인터페이스를 구현한, 관련된 클래스의 집합을 어떻게 생성할 것인가?
팩토리 인터페이스(abstract factory)를 정의하고, 생성할 것의 각 집합마다 concrete factory 클래스를 정의

추상 팩토리 패턴 사례

- 출처: http://en.wikipedia.org/wiki/Abstract_factory_pattern

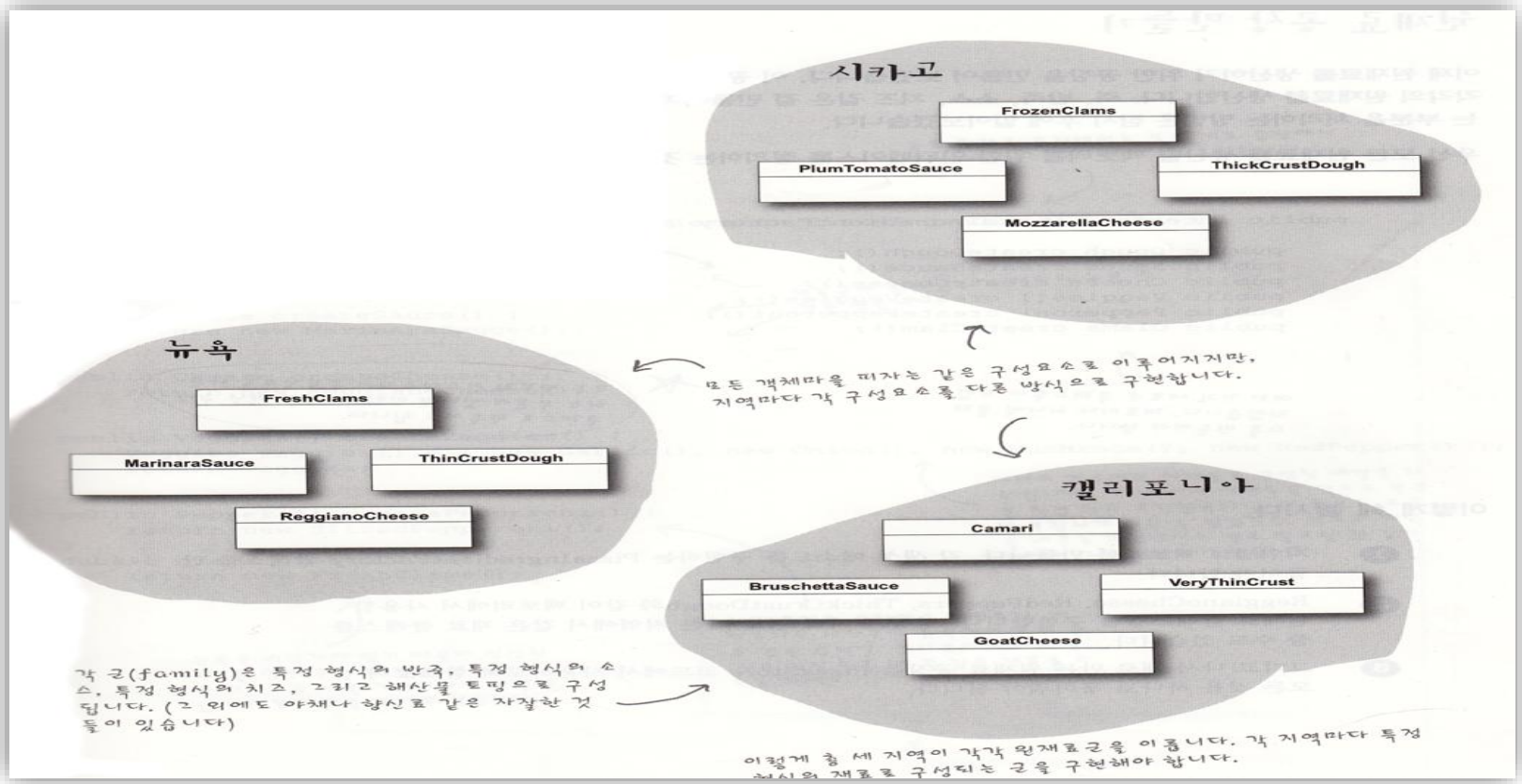


추상 팩토리 패턴 사례 분석



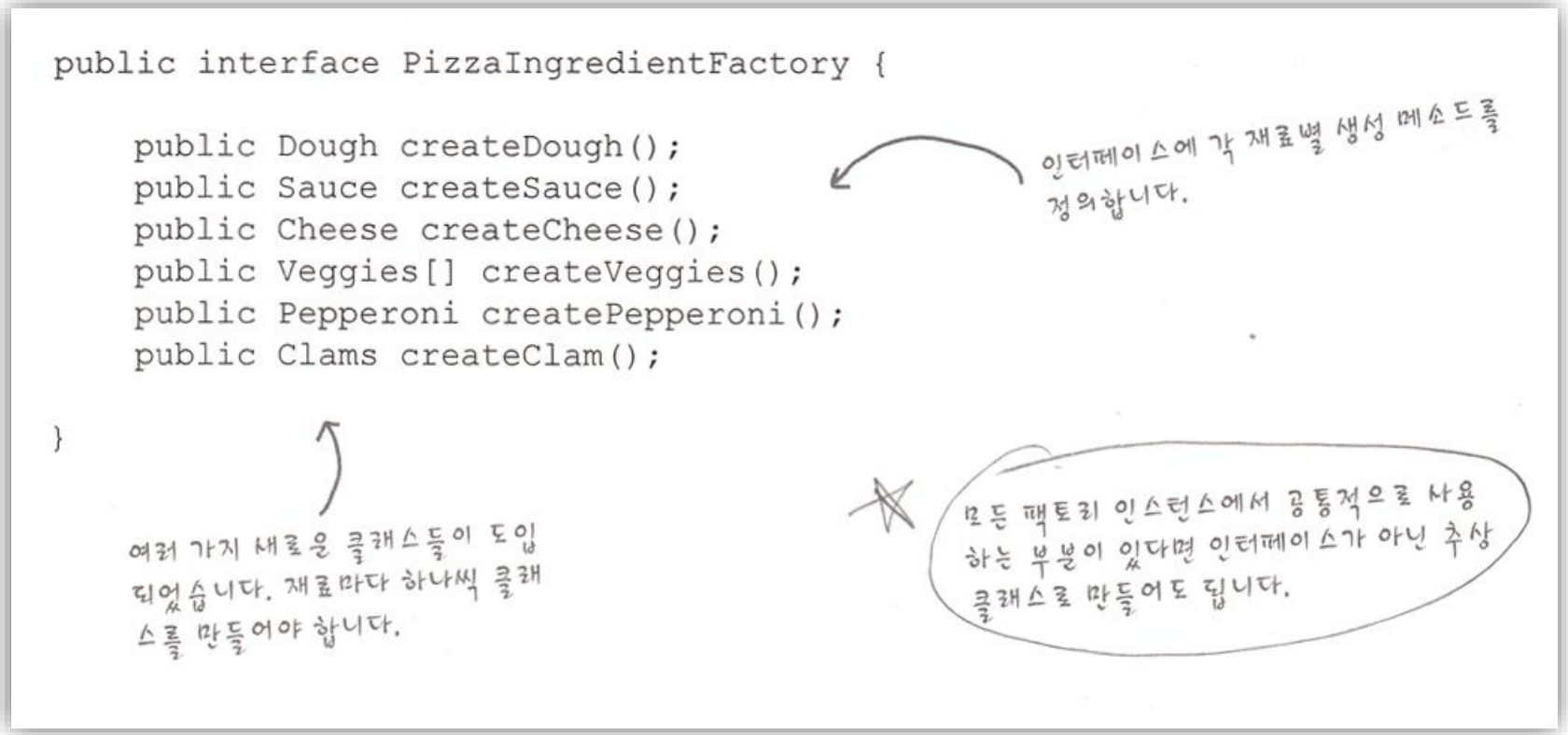
피자 가게: 네 번째 설계

- 뉴욕과 시카고에서 사용하는 재료 종류가 서로 다름.
  → 서로 다른 종류의 재료들을 제공하기 위해 ***원재료군(families of ingredients)***를 처리할 방법 필요

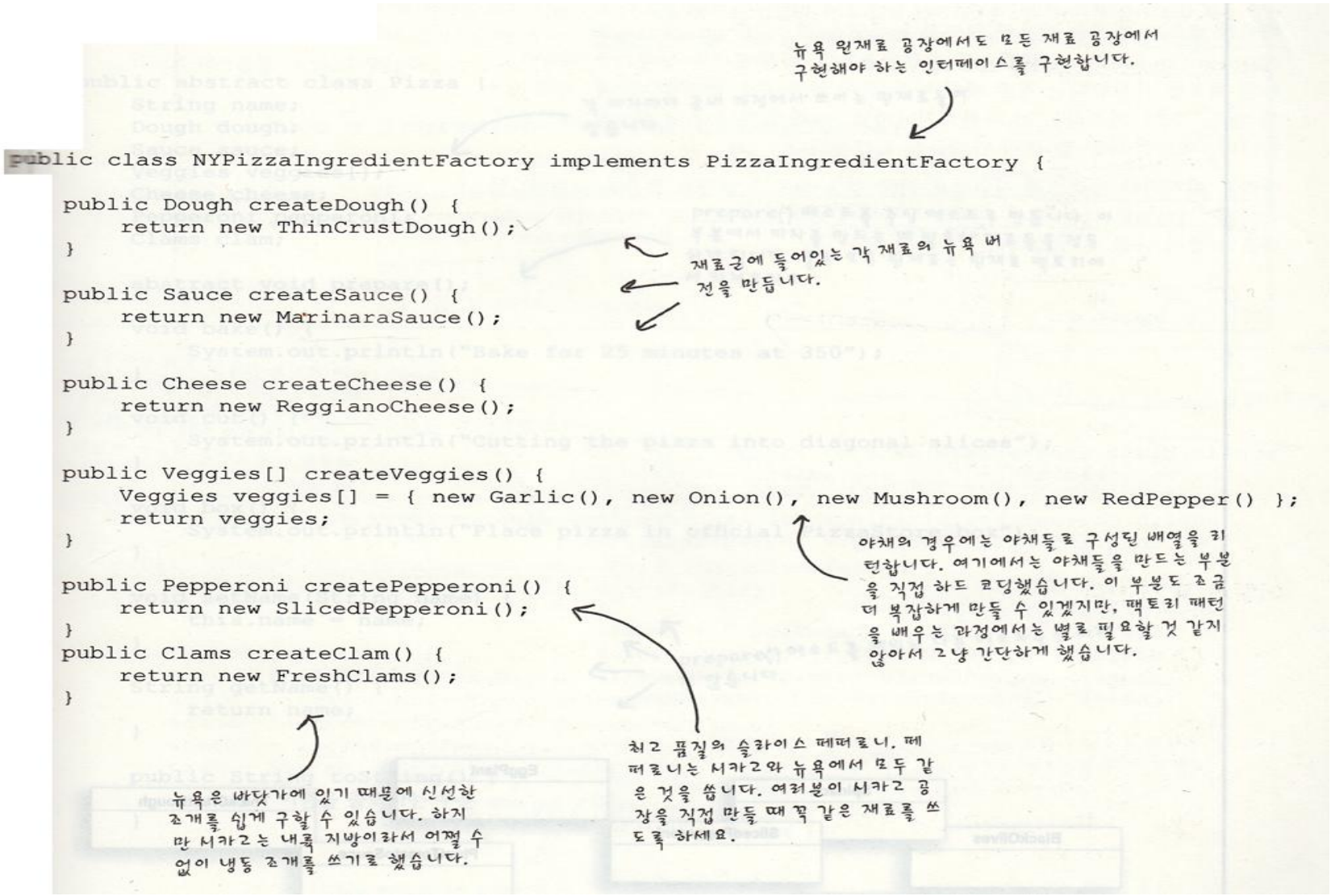


원재료 공장 만들기

- 원재료 생산 공장 : PizzaIngredientFactory 인터페이스
  - 원재료군에 들어있는 각각의 원재료를 생산하기 위해 먼저 정의



뉴욕 원재료 공장



뉴욕 원재료 공장에서도 모든 재료 공장에서 구현해야 하는 인터페이스를 구현합니다.

```
public class NYPizzaIngredientFactory implements PizzaIngredientFactory {

    public Dough createDough() {
        return new ThinCrustDough();
    }

    public Sauce createSauce() {
        return new MarinaraSauce();
    }

    public Cheese createCheese() {
        return new ReggianoCheese();
    }

    public Veggies[] createVeggies() {
        Veggies veggies[] = { new Garlic(), new Onion(), new Mushroom(), new RedPepper() };
        return veggies;
    }

    public Pepperoni createPepperoni() {
        return new SlicedPepperoni();
    }
    public Clams createClam() {
        return new FreshClams();
    }
```

재료군에 들어있는 각 재료의 뉴욕 버전을 만듭니다.

야채의 경우에는 야채들로 구성된 배열을 리턴합니다. 여기에서는 야채들을 만드는 부분을 직접 하드 코딩했습니다. 이 부분도 조금 더 복잡하게 만들 수 있겠지만, 팩토리 패턴을 배우는 과정에서는 별로 필요할 것 같지 않아서 그냥 간단하게 했습니다.

뉴욕은 바닷가에 있기 때문에 신선한 조개를 쉽게 구할 수 있습니다. 하지만 시카고는 내륙 지방이라서 어쩔 수 없이 냉동 조개를 쓰기로 했습니다.

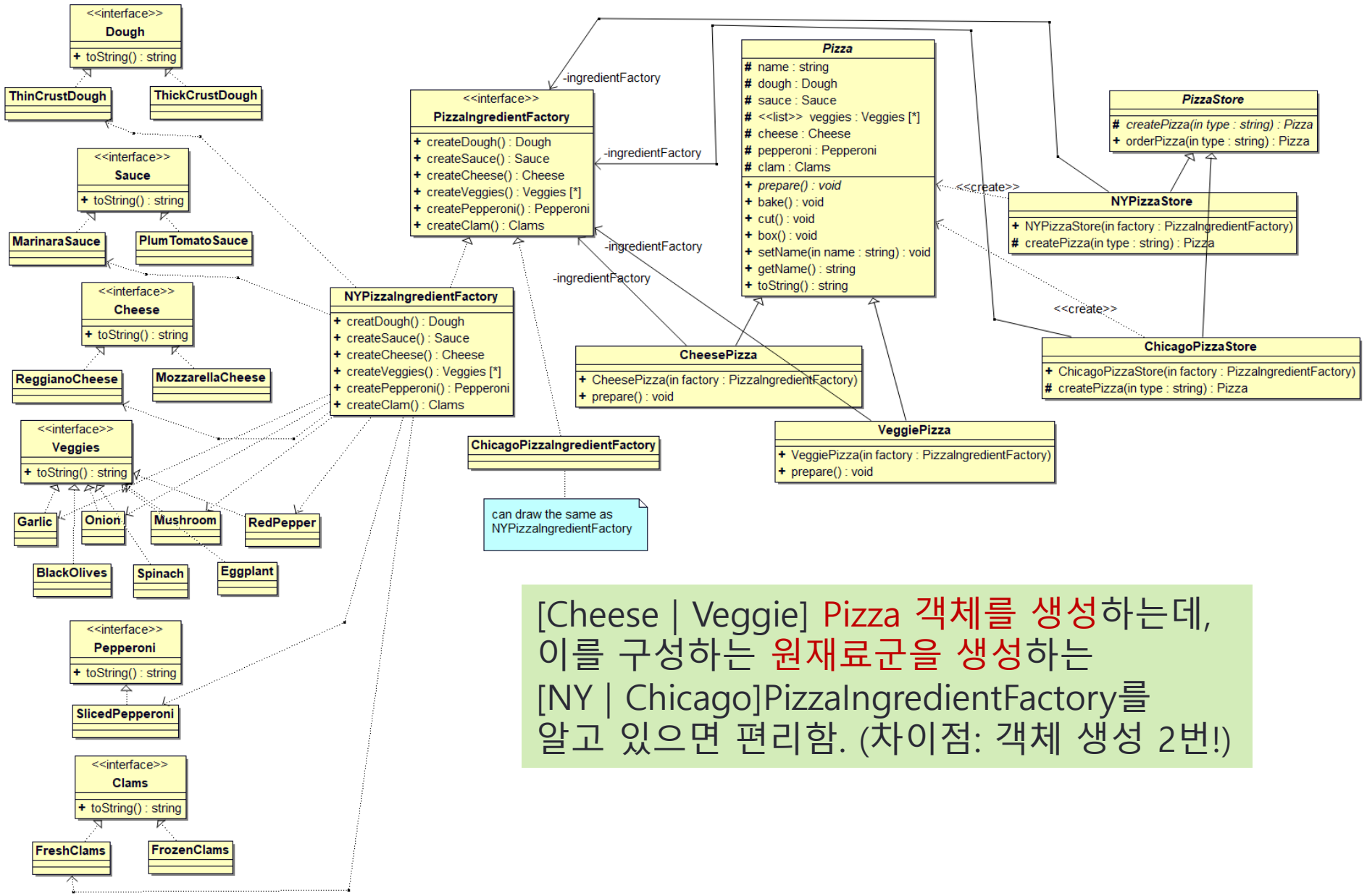
최고 품질의 슬라이스 페퍼로니. 페퍼로니는 시카고와 뉴욕에서 모두 같은 것을 씁니다. 여러분이 시카고 공장을 직접 만들 때 꼭 같은 재료를 쓰도록 하세요.

해결 방법

- 추상 팩토리 패턴 적용
  → 제품군을 위한 인터페이스 제공
  → 군(familiy)의 의미: 피자 가게의 경우 피자를 만들기 위해 필요한 모든 재료(반죽, 소스, 치즈, 고기, 야채 등)를 의미
  → 이와 같이 인터페이스를 이용하는 코드를 만들면 제품을 생산하는 실제 팩토리와 이를 이용하는 코드를 분리시킬 수 있음.

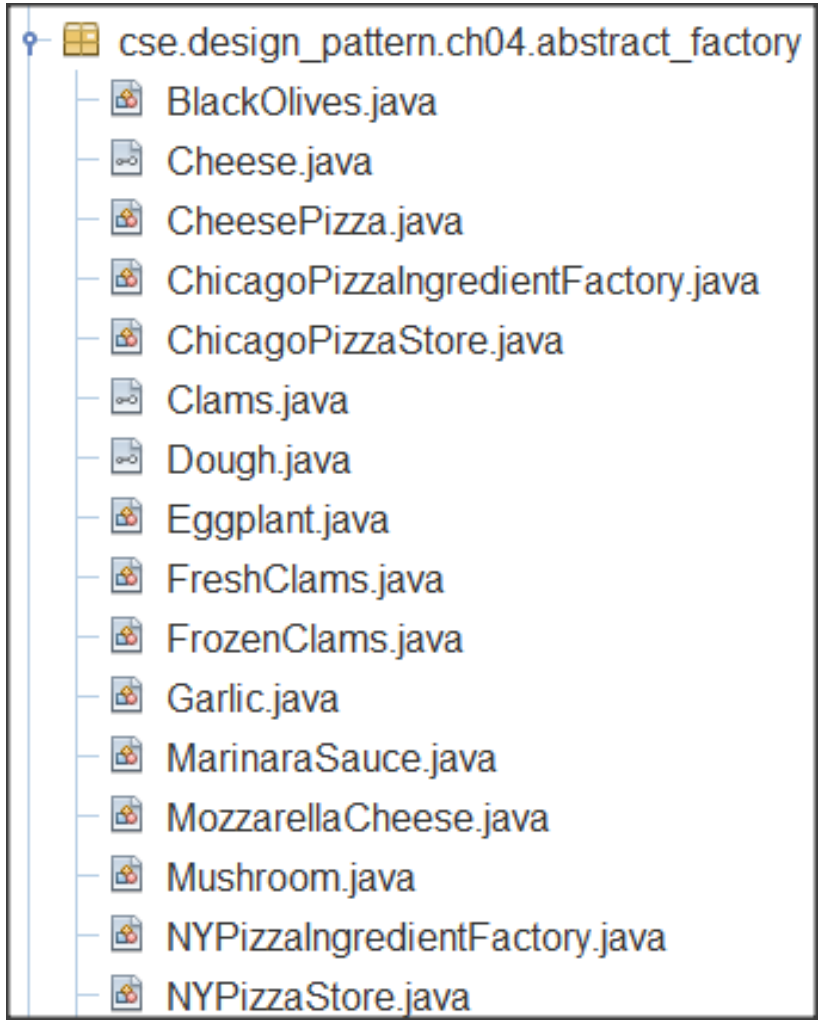
- Abstract Factory
  - Abstract product를 생성하기 위한 오퍼레이션에 대한 인터페이스 제공
  - PizzaIngredientFactory 인터페이스
- Concrete Factory
  - 구상 제품 객체를 생성하는 오퍼레이션 구현
  - NYPizzaIngredientFactory, ChicagoPizzaIngredientFactory

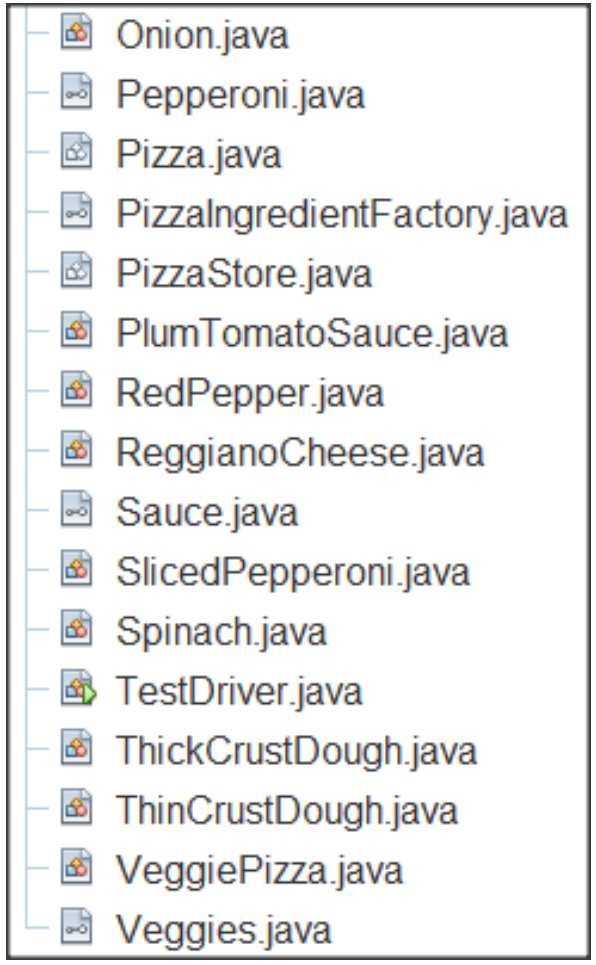
- Abstract Product
  - 제품 객체의 타입을 위한 인터페이스 선언
  - Dough 인터페이스
  - Sauce 인터페이스
  - Cheese 인터페이스
  - Clams 인터페이스
- Product
  - 해당 제품에 대응하는 구상 팩토리에서 생성하는 제품 객체를 정의
  - ThinCrustDough, ThickCrustDough, MarianaSource *etc.*
- Client
  - 추상 팩토리와 추상 제품 클래스에 의해 선언되는 인터페이스를 사용
  - NYPizzaStore 클래스, ChicagoPizzaStore 클래스



[Cheese | Veggie] Pizza 객체를 생성하는데,
이를 구성하는 원재료군을 생성하는
[NY | Chicago]PizzaIngredientFactory를
알고 있으면 편리함. (차이점: 객체 생성 2번!)

4장: abstract_factory





PizzaIngredientFactory.java

```
1   package cse.design_pattern.ch04.abstract_factory;
2
3   import java.util.List;
4
6   public interface PizzaIngredientFactory {
7
8       Dough createDough();
9
10      Sauce createSauce();
11
12      Cheese createCheese();
13
14      List<Veggies> createVeggies(); // 반환값 자료형 반영
15
16      Pepperoni createPepperoni();
17
18      Clams createClam();
19
20  }
```

NYPizzaIngredientFactory.java

```
1   package cse.design_pattern.ch04.abstract_factory;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
    public class NYPizzaIngredientFactory
        implements PizzaIngredientFactory {
8
      public Dough createDough() {
10      return new ThinCrustDough();
11    }
12
      public Sauce createSauce() {
14      return new MarinaraSauce();
15    }
16
      public Cheese createCheese() {
18      return new ReggianoCheese();
19    }
```

```
    public List<Veggies> createVeggies() {
22    List<Veggies> veggies = new ArrayList<>();
23    veggies.add(new Garlic());
24    veggies.add(new Onion());
25    veggies.add(new Mushroom());
26    veggies.add(new RedPepper());
27    return veggies;
28  }
29
    public Pepperoni createPepperoni() {
31    return new SlicedPepperoni();
32  }
33
    public Clams createClam() {
35    return new FreshClams();
36  }
37}
```

ChicagoPizzaIngredientFactory.java

```java
package cse.design_pattern.ch04.abstract_factory;

import java.util.ArrayList;
import java.util.List;

public class ChicagoPizzaIngredientFactory
    implements PizzaIngredientFactory {

  public Dough createDough() {
    return new ThickCrustDough();
  }

  public Sauce createSauce() {
    return new PlumTomatoSauce();
  }

  public Cheese createCheese() {
    return new MozzarellaCheese();
  }

  public List<Veggies> createVeggies() {
    List<Veggies> veggies = new ArrayList<>();
    veggies.add(new BlackOlives());
    veggies.add(new Spinach());
    veggies.add(new Eggplant());
    return veggies;
  }

  public Pepperoni createPepperoni() {
    return new SlicedPepperoni();
  }

  public Clams createClam() {
    return new FrozenClams();
  }
}
```

Dough 인터페이스

```java
package cse.design_pattern.ch04.abstract_factory;

public interface Dough {
    @Override
    String toString();
}
```

```java
package cse.design_pattern.ch04.abstract_factory;

public class ThinCrustDough implements Dough {

    @Override
    public String toString() {
        return "Thin Crust Dough";
    }
}
```

```java
package cse.design_pattern.ch04.abstract_factory;

public class ThickCrustDough implements Dough {

    @Override
    public String toString() {
        return "ThickCrust style extra thick crust dough";
    }
}
```

Sauce 인터페이스

```java
package cse.design_pattern.ch04.abstract_factory;

public interface Sauce {
    @Override
    String toString();
}
```

```java
package cse.design_pattern.ch04.abstract_factory;

public class MarinaraSauce implements Sauce {

    @Override
    public String toString() {
        return "Marinara Sauce";
    }
}
```

```java
package cse.design_pattern.ch04.abstract_factory;

public class PlumTomatoSauce implements Sauce {

    @Override
    public String toString() {
        return "Tomato sauce with plum tomatoes";
    }
}
```

Cheese 인터페이스

```java
package cse.design_pattern.ch04.abstract_factory;

public interface Cheese {
    @Override
    String toString();
}
```

```java
package cse.design_pattern.ch04.abstract_factory;

public class ReggianoCheese implements Cheese {

    @Override
    public String toString() {
        return "Reggiano Cheese";
    }
}
```

```java
package cse.design_pattern.ch04.abstract_factory;

public class MozzarellaCheese implements Cheese {

    @Override
    public String toString() {
        return "Shredded Mozzarella";
    }
}
```

Veggies 인터페이스

```java
package cse.design_pattern.ch04.abstract_factory;

public interface Veggies {
    @Override
    String toString();
}
```

```java
public class Garlic implements Veggies {

    @Override
    public String toString() {
        return "Garlic";
    }
}
```

```java
public class Onion implements Veggies {

    @Override
    public String toString() {
        return "Onion";
    }
}
```

```java
public class Mushroom implements Veggies {

    @Override
    public String toString() {
        return "Mushrooms";
    }
}
```

```java
public class RedPepper implements Veggies {

    @Override
    public String toString() {
        return "Red Pepper";
    }
}
```

```java
public class BlackOlives implements Veggies {

   @Override
   public String toString() {
      return "Black Olives";
   }
}
```

```java
public class Spinach implements Veggies {

   @Override
   public String toString() {
      return "Spinach";
   }
}
```

```java
public class Eggplant implements Veggies {

   @Override
   public String toString() {
      return "Eggplant";
   }
}
```

Pepperoni 인터페이스

```
1  package cse.design_pattern.ch04.abstract_factory;
2
3  public interface Pepperoni {
4      @Override
5      String toString();
6  }
```

```
1  package cse.design_pattern.ch04.abstract_factory;
2
3  public class SlicedPepperoni implements Pepperoni {
4
5      @Override
6      public String toString() {
7          return "Sliced Pepperoni";
8      }
9  }
```

Clams 인터페이스

```java
package cse.design_pattern.ch04.abstract_factory;

public interface Clams {
    @Override
    String toString();
}
```

```java
package cse.design_pattern.ch04.abstract_factory;

public class FreshClams implements Clams {

    @Override
    public String toString() {
        return "Fresh Clams from Long Island Sound";
    }
}
```

```java
package cse.design_pattern.ch04.abstract_factory;

public class FrozenClams implements Clams {

    @Override
    public String toString() {
        return "Frozen Clams from Chesapeake Bay";
    }
}
```

Pizza.java

```java
package cse.design_pattern.ch04.abstract_factory;

import java.util.List;

public abstract class Pizza {

    protected String name;
    protected Dough dough;
    protected Sauce sauce;
    protected List<Veggies> veggies;
    protected Cheese cheese;
    protected Pepperoni pepperoni;
    protected Clams clam;

    public abstract void prepare();

    void bake() {
        System.out.println("Bake for 25 minutes at 350");
    }

    void cut() {
        System.out.println("Cutting the pizza into diagonal slices");
    }

    void box() {
        System.out.println("Place pizza in official PizzaStore box");
    }

    void setName(String name) {
        this.name = name;
    }

    String getName() {
        return name;
    }
```

```
    public String toString() {
38      StringBuilder result = new StringBuilder();
39      result.append("---- ").append(name).append(" ----₩n");
40      if (dough != null) {
41          result.append(dough).append("₩n");
42      }
43      if (sauce != null) {
44          result.append(sauce).append("₩n");
45      }
46      if (cheese != null) {
47          result.append(cheese).append("₩n");
48      }
49      if (veggies != null) {
50          for (int i = 0; i < veggies.size(); i++) {
51              result.append(veggies.get(i));
52              if (i < veggies.size() - 1) {
53                  result.append(", ");
54              }
55          }
56          result.append("₩n");
57      }
58      if (clam != null) {
59          result.append(clam).append("₩n");
60      }
61      if (pepperoni != null) {
62          result.append(pepperoni).append("₩n");
63      }
64      return result.toString();
65  }
66 }
```

CheesePizza.java

```
1   package cse.design_pattern.ch04.abstract_factory;
2
3   public class CheesePizza extends Pizza {
4
5       private PizzaIngredientFactory ingredientFactory;
6
7       public CheesePizza(PizzaIngredientFactory ingredientFactory) {
8           this.ingredientFactory = ingredientFactory;
9       }
10
11      public void prepare() {
12          System.out.println("Preparing " + name);
13          dough = ingredientFactory.createDough();
14          sauce = ingredientFactory.createSauce();
15          cheese = ingredientFactory.createCheese();
16      }
17  }
```

- 객체 생성시 특정 PizzaIngredientFactory 객체를 연결하여
- CheesePizza 객체 생성에 필요한 원재료군 객체를 생성

VeggiePizza.java

```
1   package cse.design_pattern.ch04.abstract_factory;
2
3   public class VeggiePizza extends Pizza {
4
5       private PizzaIngredientFactory ingredientFactory;
6
7       public VeggiePizza(PizzaIngredientFactory factory) {
8           this.ingredientFactory = factory;
9       }
10
11      public void prepare() {
12          System.out.println("Preparing " + name);
13          dough = ingredientFactory.createDough();
14          sauce = ingredientFactory.createSauce();
15          cheese = ingredientFactory.createCheese();
16          // CheesePizza와의 차이점
17          //   --> Pizza.prepare(): 추상 오퍼레이션이어야 하는 이유
18          // Template method pattern 적용 가능
19          veggies = ingredientFactory.createVeggies();
20      }
21  }
```

- 객체 생성시 특정 PizzaIngredientFactory 객체를 연결하여
- VeggiePizza 객체 생성에 필요한 원재료군 객체를 생성

PizzaStore.java

```
1   package cse.design_pattern.ch04.abstract_factory;
2
3   public abstract class PizzaStore {
4
5       protected abstract Pizza createPizza(String type);
6
7       public Pizza orderPizza(String type) {
8           Pizza pizza = createPizza(type);
9           System.out.println("--- Making a " + pizza.getName() + " ---");
10          pizza.prepare();
11          pizza.bake();
12          pizza.cut();
13          pizza.box();
14          return pizza;
15      }
16  }
```

NYPizzaStore.java

```
1   package cse.design_pattern.ch04.abstract_factory;
2
3   public class NYPizzaStore extends PizzaStore {
4
5     private PizzaIngredientFactory ingredientFactory;
6
7     public NYPizzaStore() {
8       this.ingredientFactory = new NYPizzaIngredientFactory();
9     }
10
11    protected Pizza createPizza(String type) {
12      Pizza pizza = null;
13
14      // ClamPizza와 PepperoniPizza도 같은 방식으로 하면 됨.
15      if (type.equals("cheese")) {
16        pizza = new CheesePizza(ingredientFactory);
17        pizza.setName("New York Style Cheese Pizza");
18      } else if (type.equals("veggie")) {
19        pizza = new VeggiePizza(ingredientFactory);
20        pizza.setName("New York Style Veggie Pizza");
21      }
22      return pizza;
23    }
24  }
```

ChicagoPizzaStore.java

```java
package cse.design_pattern.ch04.abstract_factory;

public class ChicagoPizzaStore extends PizzaStore {

    private PizzaIngredientFactory ingredientFactory;

    public ChicagoPizzaStore() {
        this.ingredientFactory = new ChicagoPizzaIngredientFactory();
    }

    protected Pizza createPizza(String type) {
        Pizza pizza = null;

        // ClamPizza와 PepperoniPizza도 같은 방식으로 하면 됨.
        if (type.equals("cheese")) {
            pizza = new CheesePizza(ingredientFactory);
            pizza.setName("Chicago Style Cheese Pizza");
        } else if (type.equals("veggie")) {
            pizza = new VeggiePizza(ingredientFactory);
            pizza.setName("Chicago Style Veggie Pizza");
        }
        return pizza;
    }
}
```

TestDriver.java

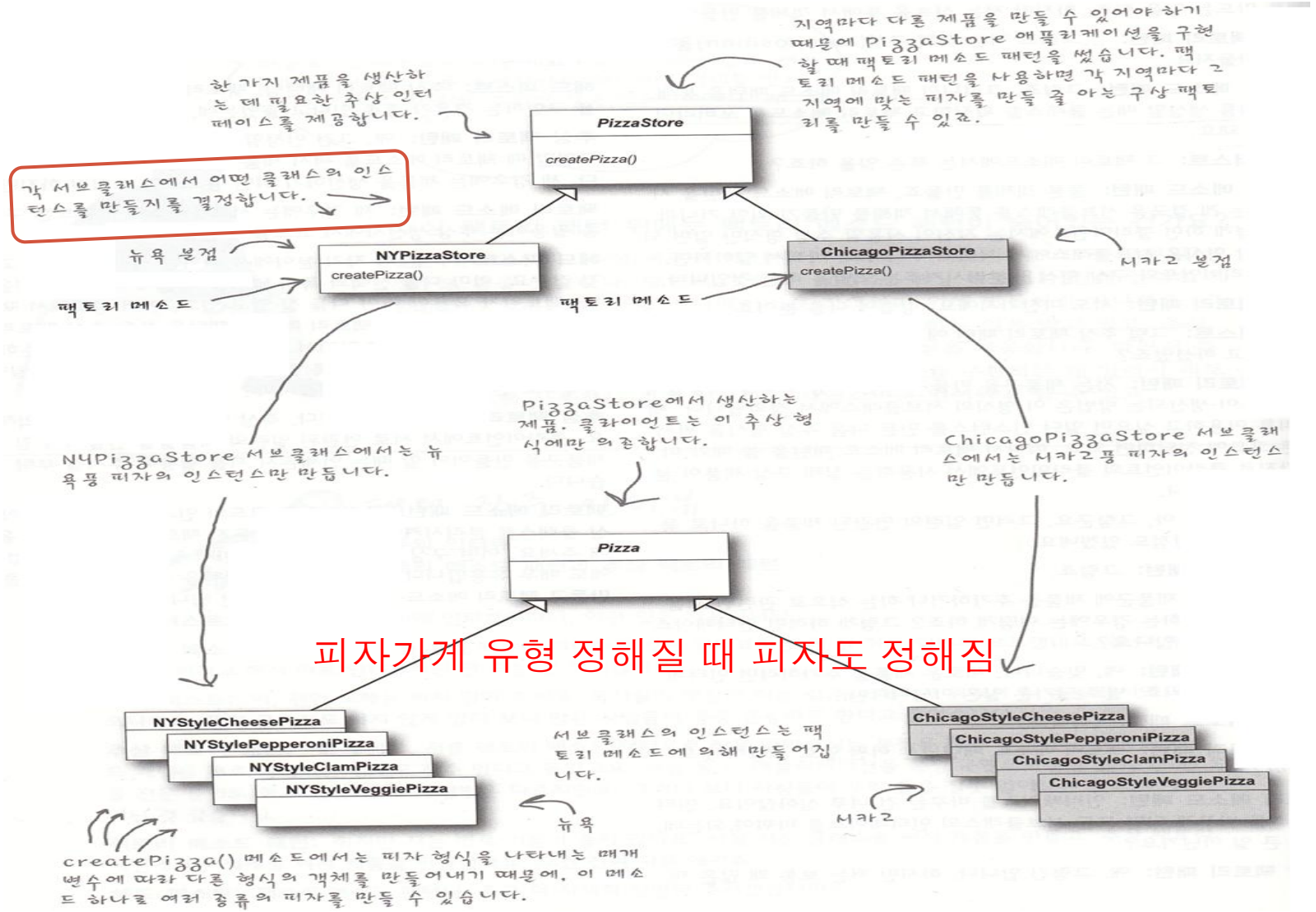
```
6   package cse.design_pattern.ch04.abstract_factory;
7
8   /**...4 lines */
12  public class TestDriver {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          PizzaStore nyStore = new NYPizzaStore();
19          PizzaStore chicagoStore = new ChicagoPizzaStore();
20
21          Pizza pizza = nyStore.orderPizza("cheese");
22          System.out.println("Ethan ordered a " + pizza + "₩n");
23
24          pizza = chicagoStore.orderPizza("cheese");
25          System.out.println("Joel ordered a " + pizza + "₩n");
26
27          pizza = nyStore.orderPizza("veggie");
28          System.out.println("Ethan ordered a " + pizza + "₩n");
29
30          pizza = chicagoStore.orderPizza("veggie");
31          System.out.println("Joel ordered a " + pizza + "₩n");
32      }
33  }
```

실행 결과

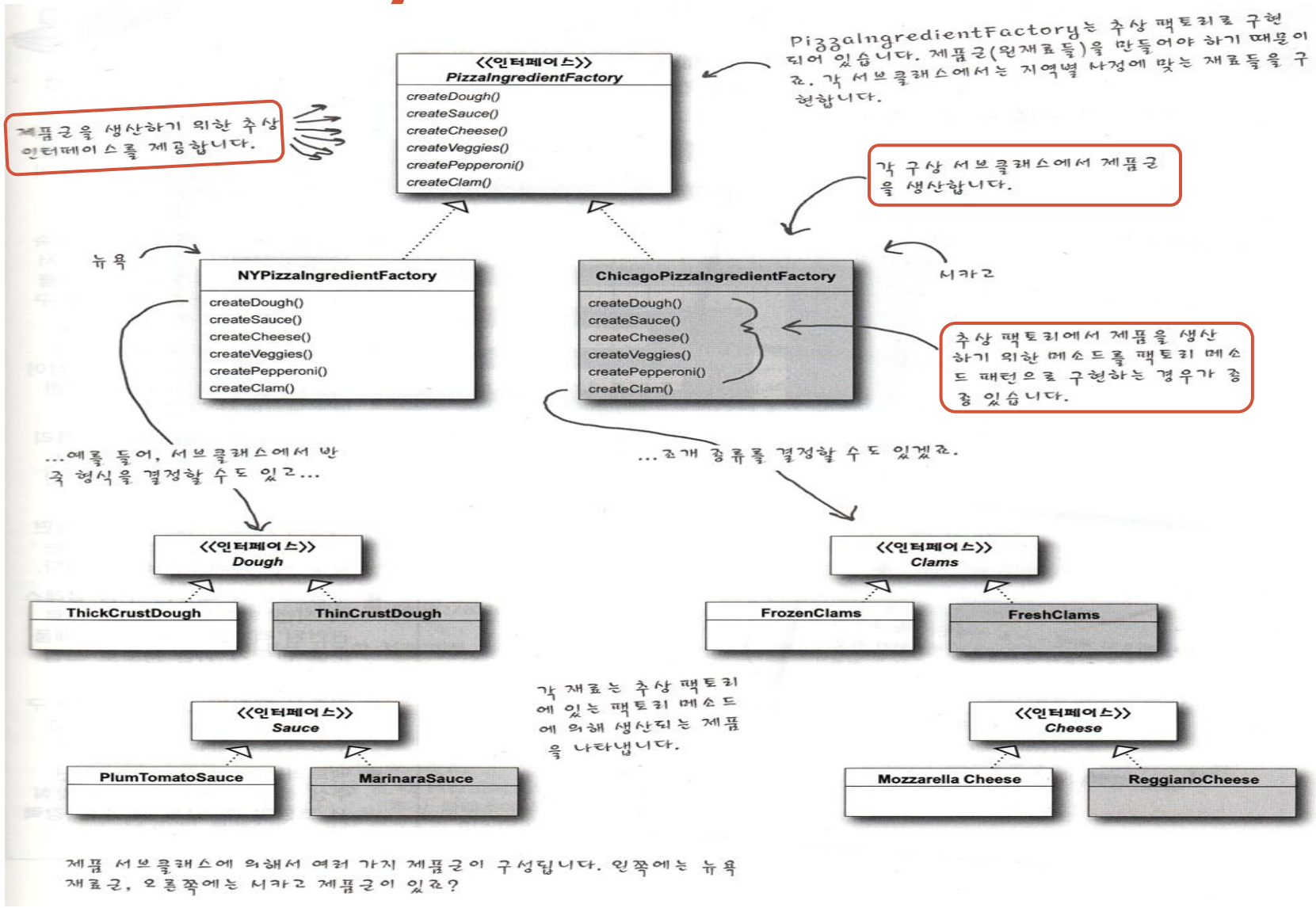
```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desir
--- Making a New York Style Cheese Pizza ---
Preparing New York Style Cheese Pizza
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a ---- New York Style Cheese Pizza ----
Thin Crust Dough
Marinara Sauce
Reggiano Cheese


--- Making a Chicago Style Cheese Pizza ---
Preparing Chicago Style Cheese Pizza
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Joel ordered a ---- Chicago Style Cheese Pizza ----
ThickCrust style extra thick crust dough
Tomato sauce with plum tomatoes
Shredded Mozzarella
```

```
--- Making a New York Style Veggie Pizza ---
Preparing New York Style Veggie Pizza
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a ---- New York Style Veggie Pizza ----
Thin Crust Dough
Marinara Sauce
Reggiano Cheese
Garlic, Onion, Mushrooms, Red Pepper


--- Making a Chicago Style Veggie Pizza ---
Preparing Chicago Style Veggie Pizza
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Joel ordered a ---- Chicago Style Veggie Pizza ----
ThickCrust style extra thick crust dough
Tomato sauce with plum tomatoes
Shredded Mozzarella
Black Olives, Spinach, Eggplant
```

Factory Method Pattern



Abstract Factory Pattern



Factory: 어떤 패턴을 사용?

- new MyProduct() 가 코드 여러 곳에서 중복
  → simple factory (pattern?) 사용

- MyProductA, MyProductB와 같이 유사한 product 객체 생성 필요 → factory method pattern 사용: concrete Creator에서 어떤 product 생산할 지 결정 → Creator가 factory 기능 겸함

- MyProductA, MyProductB와 같이 유사한 product 생산할 때 원재료군(a family of ingredients; 객체를 구성하는 작은 요소 객체의 집합) 생성 필요 → abstract factory pattern 사용: 원재료군 생성 책임을 추상 팩토리 인터페이스에서 정의 → 구상 팩토리에서 원재료군 구상 객체 생성