소프트웨어 디자인 패턴
프로그램 숙제 1

20190839 경제학과 이승민

목차

1. DependentPizzaStore 버전

DependentPizzaStore 코드는 피자 스타일과 종류에 따라 여러 조건문을 사용하여 구상 클래스의 인스턴스를 생성한다. NY, Chicago, Seoul 스타일을 구분하고, 그에 맞는 피자를 생성하는 방식으로 구현되어 있다. 이와 같이 각 스타일에 맞춘 클래스를 직접 생성하는 방식은 단일 책임 원칙(SRP)을 지키지 못하고, 코드의 변경이 필요할 때마다 전체적인 영향을 줄 수 있다.

기존 코드

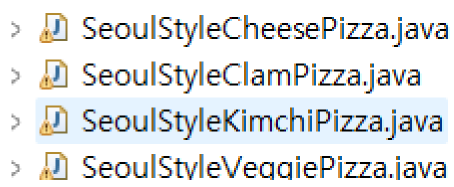
```java
package headfirst.factory.pizzafm;
public class DependentPizzaStore {

	public Pizza createPizza(String style, String type) {
		Pizza pizza = null;
		if (style.equals("NY")) {
			if (type.equals("cheese")) {
				pizza = new NYStyleCheesePizza();
			} else if (type.equals("veggie")) {
				pizza = new NYStyleVeggiePizza();
			} else if (type.equals("clam")) {
				pizza = new NYStyleClamPizza();
			} else if (type.equals("pepperoni")) {
				pizza = new NYStylePepperoniPizza();
			}
		} else if (style.equals("Chicago")) {
			if (type.equals("cheese")) {
				pizza = new ChicagoStyleCheesePizza();
			} else if (type.equals("veggie")) {
				pizza = new ChicagoStyleVeggiePizza();
			} else if (type.equals("clam")) {
				pizza = new ChicagoStyleClamPizza();
			} else if (type.equals("pepperoni")) {
				pizza = new ChicagoStylePepperoniPizza();
			}
		} else {
			System.out.println("Error: invalid type of pizza");
			return null;
		}
		pizza.prepare();
		pizza.bake();
		pizza.cut();
		pizza.box();
```

```
                return pizza;
        }
}
```

위 원래 코드에 아래 부분을 추가했다.

```
                } else if(style.equals("Seoul")) {
                        if (type.equals("Kimchi")) {
                                pizza = new SeoulStyleKimchiPizza();
                        } else if (type.equals("veggie")) {
                                pizza = new SeoulStyleVeggiePizza();
                        } else if (type.equals("clam")) {
                                pizza = new SeoulStyleClamPizza();
                        } else if (type.equals("cheese")) {
                                pizza = new SeoulStyleCheesePizza();
                        }
```

기존 코드에 Seoul 스타일을 추가하면서 조건문에 새로운 스타일과 그에 따른 피자 종류를 추가했다. 아래는 SeoulStyleKimchiPizza 클래스이다:


SeoulStyleCheesePizza.java
SeoulStyleClamPizza.java
SeoulStyleKimchiPizza.java
SeoulStyleVeggiePizza.java

각각의 클래스는 Pizza 인터페이스를 통해 구현하였다.

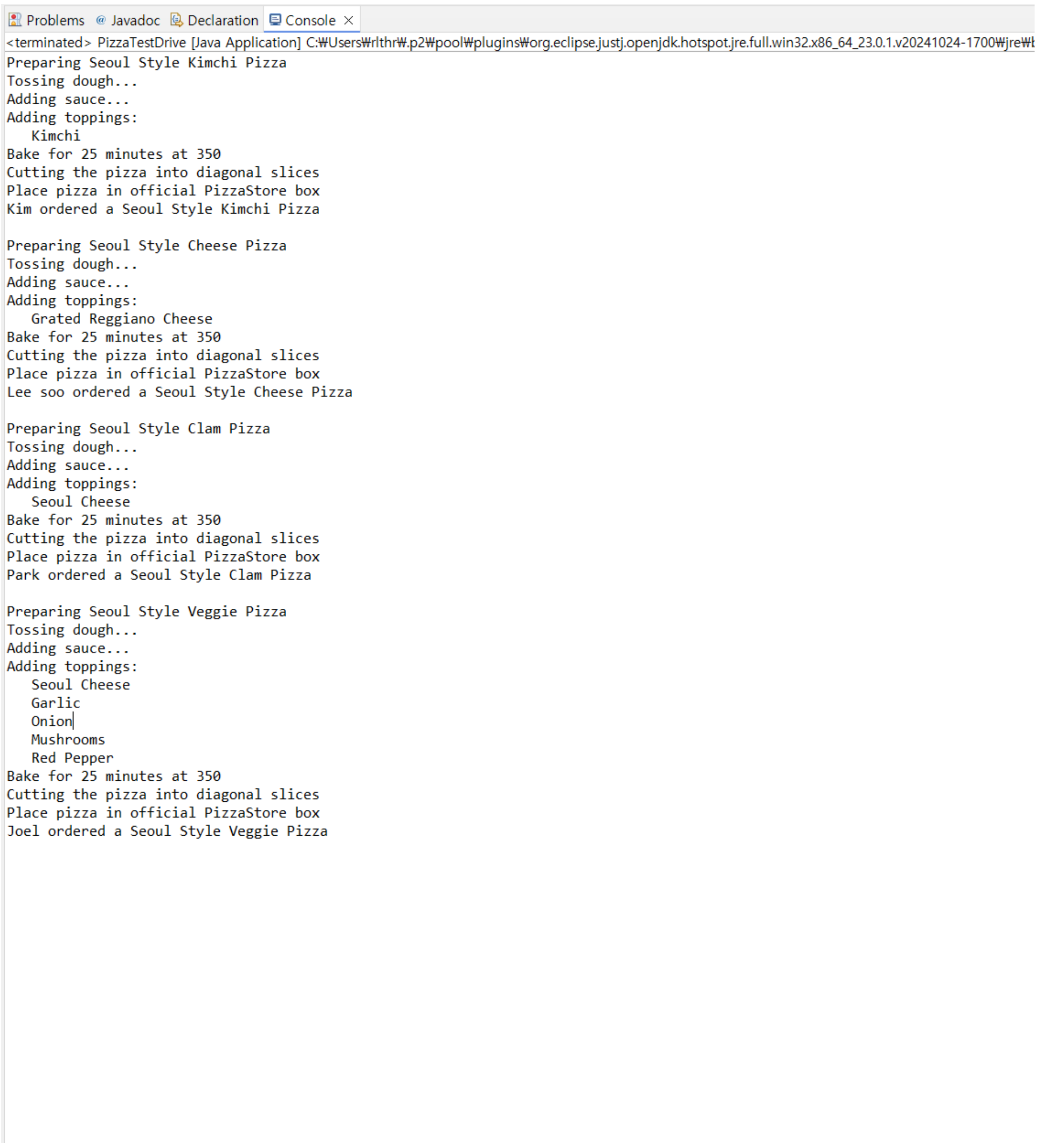
```
package headfirst.factory.pizzafm;
public class SeoulStyleKimchiPizza extends Pizza {

        public SeoulStyleKimchiPizza() {
                name = "Seoul Style Kimchi Pizza";
                dough = "Rice Dough";
                sauce = "Kimchi Sauce";

                toppings.add("Kimchi");
        }
}
```

Dough가 쌀로 만들어졌으며, 김치가 토핑으로 올라간다.

Seoul 지점의 코드를 테스트 하기 위한 테스트 코드이다.

```
package headfirst.factory.pizzafm;
public class PizzaTestDrive {

        public static void main(String[] args) {

                Pizza pizza;
                DependentPizzaStore dependentPizzaStore = new DependentPizzaStore();

                pizza = dependentPizzaStore.createPizza("Seoul", "Kimchi");
                System.out.println("Kim ordered a " + pizza.getName() + "\n");
                pizza = dependentPizzaStore.createPizza("Seoul", "cheese");
                System.out.println("Lee soo ordered a " + pizza.getName() + "\n");

                pizza = dependentPizzaStore.createPizza("Seoul", "clam");
                System.out.println("Park ordered a " + pizza.getName() + "\n");

                pizza = dependentPizzaStore.createPizza("Seoul", "veggie");
                System.out.println("Joel ordered a " + pizza.getName() + "\n");

        }
}
```

실행결과



```
<terminated> PizzaTestDrive [Java Application] C:₩Users₩rlthr₩.p2₩pool₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.1.v20241024-1700₩jre₩
Preparing Seoul Style Kimchi Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Kimchi
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Kim ordered a Seoul Style Kimchi Pizza

Preparing Seoul Style 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
Lee soo ordered a Seoul Style Cheese Pizza

Preparing Seoul Style Clam Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Seoul Cheese
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Park ordered a Seoul Style Clam Pizza

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

2. Factory Method 가 적용된 버전

Factory Method 를 적용하여 DependentPizzaStore 의 문제점을 해결하려 했다. 핵심은 피자 생성 책임을 각 지역의 피자 가게(PizzaStore 서브클래스)로 분리하는 것이다. 이렇게 하면 새로운 피자 스타일을 추가할 때 각 지역에 맞는 서브클래스만 추가하면 된다

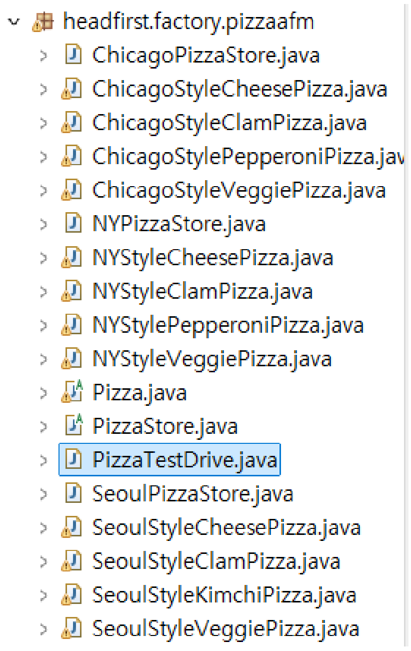
PizzaStore 인터페이스를 통해 SeoulPizzaStore 클래스 추가

```java
package headfirst.factory.pizzaafm;
public abstract class PizzaStore {

         abstract Pizza createPizza(String item);

         public Pizza orderPizza(String type) {
                  Pizza pizza = createPizza(type);
                  System.out.println("--- Making a " + pizza.getName() + " ---");
                  pizza.prepare();
                  pizza.bake();
                  pizza.cut();
                  pizza.box();
                  return pizza;
         }
}
```

SeoulPizzaStore

```java
package headfirst.factory.pizzaafm;
public class SeoulPizzaStore extends PizzaStore {
    @Override
    Pizza createPizza(String type) {
        if (type.equals("Kimchi")) {
           return new SeoulStyleKimchiPizza();
        } else if (type.equals("cheese")) {
           return new SeoulStyleCheesePizza();
        } else if (type.equals("clam")) {
           return new SeoulStyleClamPizza();
        } else if (type.equals("veggie")) {
           return new SeoulStyleVeggiePizza();
        } else {
           return null;
        }
    }
}
```

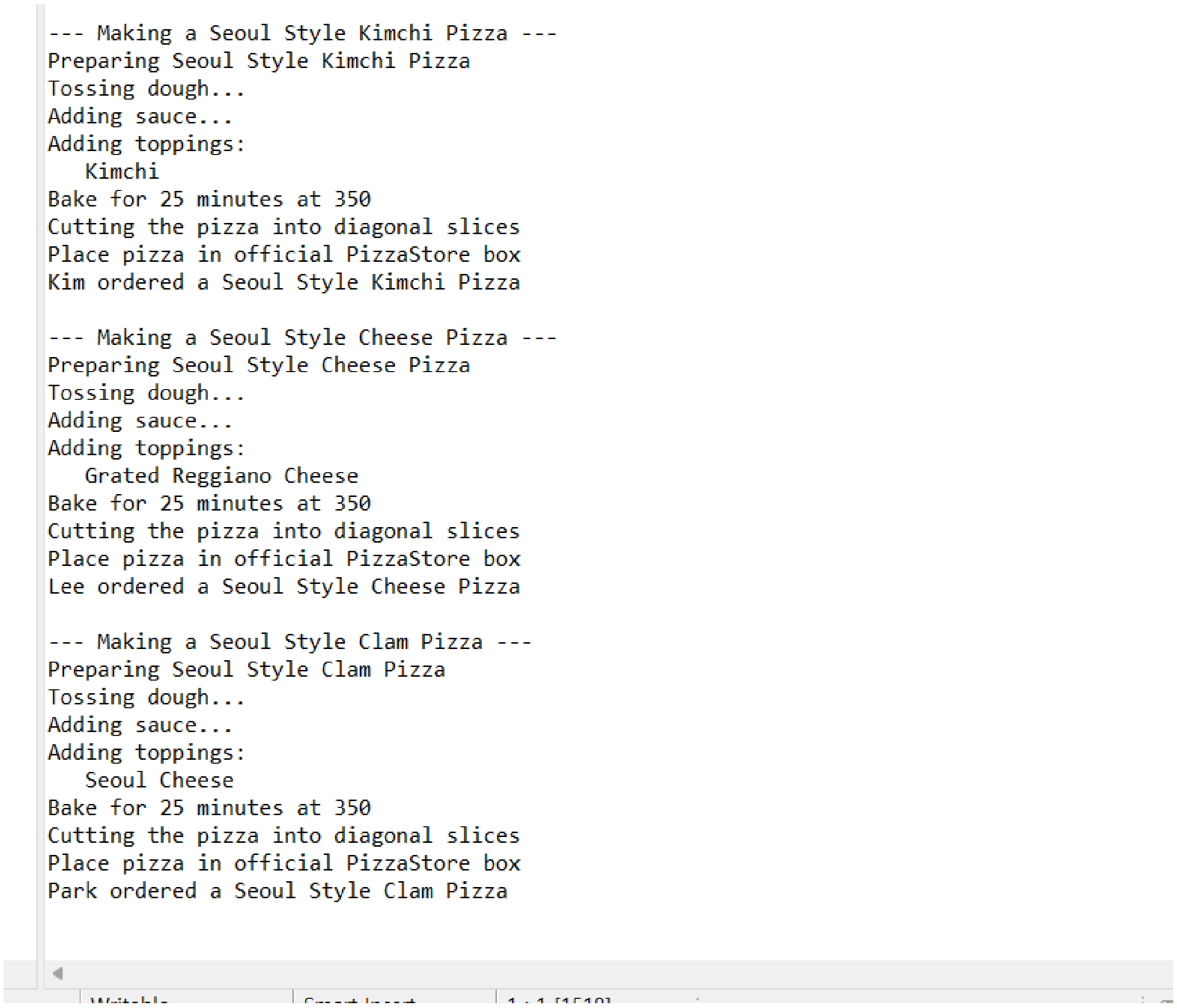
피자를 만드는 과정에서 생성 부분을 추상화하여, 피자 종류가 바뀌더라도 PizzaStore 클래스의 로직은 그대로 유지될 수 있다. 이렇게 함으로써 각 지역의 특색을 반영하는 새로운 피자를 쉽게 추가할 수 있게 된다.


headfirst.factory.pizzaafm
ChicagoPizzaStore.java
ChicagoStyleCheesePizza.java
ChicagoStyleClamPizza.java
ChicagoStylePepperoniPizza.jav
ChicagoStyleVeggiePizza.java
NYPizzaStore.java
NYStyleCheesePizza.java
NYStyleClamPizza.java
NYStylePepperoniPizza.java
NYStyleVeggiePizza.java
Pizza.java
PizzaStore.java
PizzaTestDrive.java
SeoulPizzaStore.java
SeoulStyleCheesePizza.java
SeoulStyleClamPizza.java
SeoulStyleKimchiPizza.java
SeoulStyleVeggiePizza.java

Test Code 추가

```java
package headfirst.factory.pizzaafm;
public class PizzaTestDrive {

	public static void main(String[] args) {
		PizzaStore nyStore = new NYPizzaStore();
		PizzaStore chicagoStore = new ChicagoPizzaStore();
		PizzaStore seoulStore = new SeoulPizzaStore();

		Pizza pizza = nyStore.orderPizza("cheese");
		System.out.println("Ethan ordered a " + pizza.getName() + "\n");

		pizza = chicagoStore.orderPizza("cheese");
		System.out.println("Joel ordered a " + pizza.getName() + "\n");
		pizza = nyStore.orderPizza("clam");
		System.out.println("Ethan ordered a " + pizza.getName() + "\n");

		pizza = chicagoStore.orderPizza("clam");
		System.out.println("Joel ordered a " + pizza.getName() + "\n");
		pizza = nyStore.orderPizza("pepperoni");
		System.out.println("Ethan ordered a " + pizza.getName() + "\n");

		pizza = chicagoStore.orderPizza("pepperoni");
		System.out.println("Joel ordered a " + pizza.getName() + "\n");
		pizza = nyStore.orderPizza("veggie");
		System.out.println("Ethan ordered a " + pizza.getName() + "\n");

		pizza = chicagoStore.orderPizza("veggie");
		System.out.println("Joel ordered a " + pizza.getName() + "\n");

	pizza = seoulStore.orderPizza("Kimchi");
	System.out.println("Kim ordered a " + pizza.getName() + "\n");
	pizza = seoulStore.orderPizza("cheese");
	System.out.println("Lee ordered a " + pizza.getName() + "\n");
	pizza = seoulStore.orderPizza("clam");
	System.out.println("Park ordered a " + pizza.getName() + "\n");
	  }
}
```

실행결과



```
--- Making a Seoul Style Kimchi Pizza ---
Preparing Seoul Style Kimchi Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Kimchi
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Kim ordered a Seoul Style Kimchi Pizza

--- Making a Seoul Style Cheese Pizza ---
Preparing Seoul Style 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
Lee ordered a Seoul Style Cheese Pizza

--- Making a Seoul Style Clam Pizza ---
Preparing Seoul Style Clam Pizza
Tossing dough...
Adding sauce...
Adding toppings:
   Seoul Cheese
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Park ordered a Seoul Style Clam Pizza
```

3. Abstract Factory 가 적용된 버전

구상 클래스에 의존하지 않고도 서로 연관되거나 의존적인 객체로 이루어진 제품군을 생산하는 인터페이스 제공한다. 구상 클래스는 서브클래스에서 만들고, 클라이언트와 팩토리에서 생산되는 제품을 분리할 수 있다.

SeoulPizzaIngredientFactory 구현

```java
package headfirst.factory.pizzaaf;
public class SeoulPizzaIngredientFactory implements PizzaIngredientFactory {

    public Dough createDough() {
        return new RiceDough();
    }
    public Sauce createSauce() {
        return new KimchiSauce();
    }

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

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

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

    public Clams createClam() {
        return new FreshClams();
    }

}
```

RiceDough 클래스, KimchiSauce 클래스와 Kimchi 클래스를 추가하였다.

```java
package headfirst.factory.pizzaaf;
public class RiceDough implements Dough {
    public String toString() {
        return "Rice Dough";
    }
}


package headfirst.factory.pizzaaf;
public class KimchiSauce implements Sauce {
    public String toString() {
        return "Kimchi Sauce";
    }
}

package headfirst.factory.pizzaaf;
public class Kimchi implements Veggies {
    public String toString() {
          return "Kimchi";
    }
}
```

SeoulPizzaStore 클래스

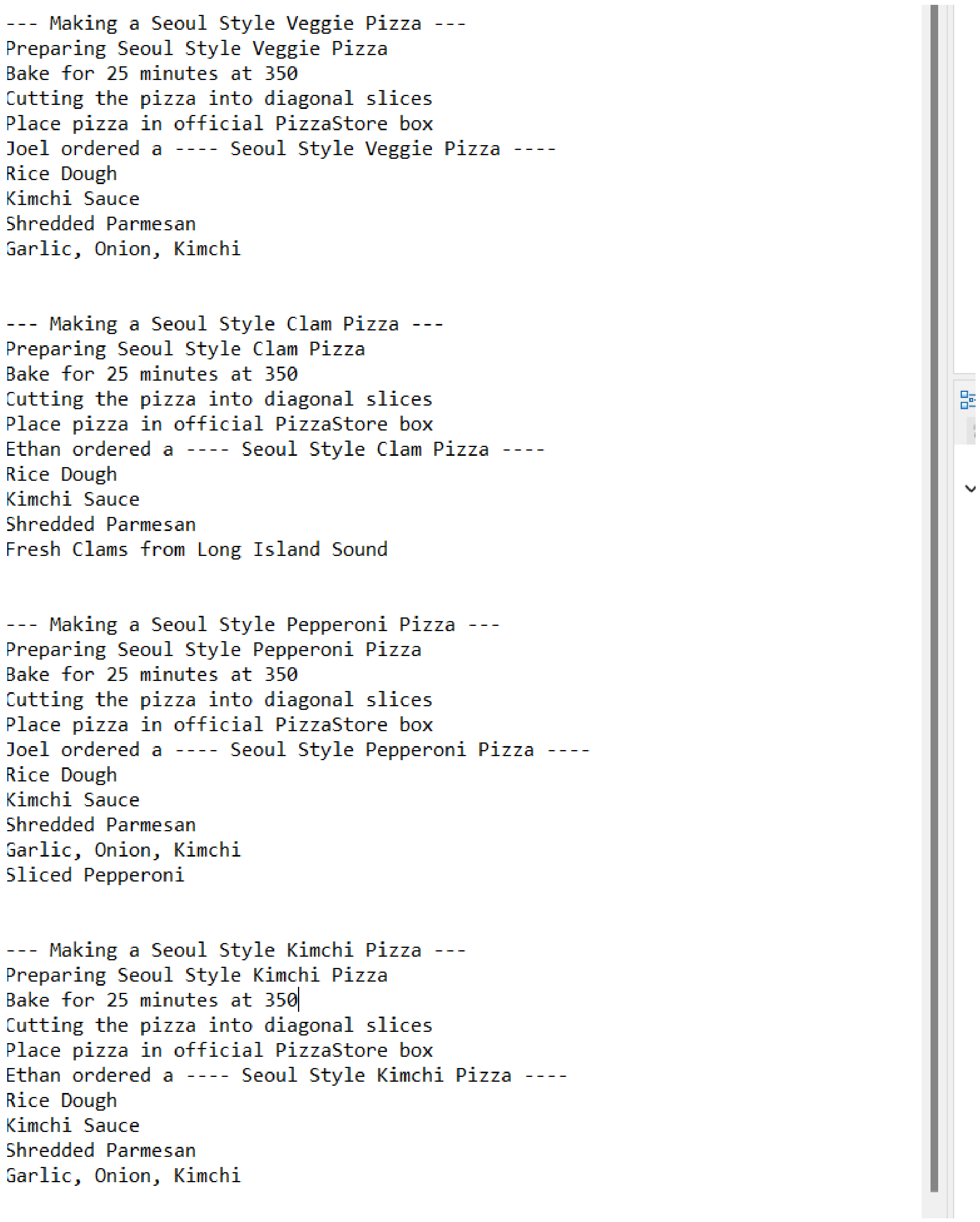
서울 지점의 PizzaStore 에서는 SeoulPizzaIngredientFactory 를 사용하여 피자를 생성한다. 각 피자는 지역 재료 팩토리를 통해 생성된다.

```java
package headfirst.factory.pizzaaf;
public class SeoulPizzaStore extends PizzaStore {
    @Override
    protected Pizza createPizza(String item) {
        Pizza pizza = null;
        PizzaIngredientFactory ingredientFactory =
            new SeoulPizzaIngredientFactory();
        if (item.equals("cheese")) {
            pizza = new CheesePizza(ingredientFactory);
            pizza.setName("Seoul Style Cheese Pizza");
        } else if (item.equals("veggie")) {
            pizza = new VeggiePizza(ingredientFactory);
            pizza.setName("Seoul Style Veggie Pizza");
        } else if (item.equals("clam")) {
            pizza = new ClamPizza(ingredientFactory);
            pizza.setName("Seoul Style Clam Pizza");
        } else if (item.equals("pepperoni")) {
            pizza = new PepperoniPizza(ingredientFactory);
            pizza.setName("Seoul Style Pepperoni Pizza");
        } else if (item.equals("kimchi")) {
            pizza = new KimchiPizza(ingredientFactory);
            pizza.setName("Seoul Style Kimchi Pizza");
        }

        return pizza;
    }
}
```

아래는 테스트 코드이다

```
package headfirst.factory.pizzaaf;
public class PizzaTestDrive {

         public static void main(String[] args) {
                  PizzaStore nyStore = new NYPizzaStore();
                  PizzaStore chicagoStore = new ChicagoPizzaStore();
                  PizzaStore seoulStore = new SeoulPizzaStore();
                Pizza pizza = seoulStore.orderPizza("cheese");
                System.out.println("Ethan ordered a " + pizza + "\n");
                pizza = seoulStore.orderPizza("veggie");
                System.out.println("Joel ordered a " + pizza + "\n");
                pizza = seoulStore.orderPizza("clam");
                System.out.println("Ethan ordered a " + pizza + "\n");
                pizza = seoulStore.orderPizza("pepperoni");
                System.out.println("Joel ordered a " + pizza + "\n");
                pizza = seoulStore.orderPizza("kimchi");
                System.out.println("Ethan ordered a " + pizza + "\n");
         }
}
```

출력결과



```
--- Making a Seoul Style Veggie Pizza ---
Preparing Seoul Style Veggie Pizza
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Joel ordered a ---- Seoul Style Veggie Pizza ----
Rice Dough
Kimchi Sauce
Shredded Parmesan
Garlic, Onion, Kimchi


--- Making a Seoul Style Clam Pizza ---
Preparing Seoul Style Clam Pizza
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a ---- Seoul Style Clam Pizza ----
Rice Dough
Kimchi Sauce
Shredded Parmesan
Fresh Clams from Long Island Sound


--- Making a Seoul Style Pepperoni Pizza ---
Preparing Seoul Style Pepperoni Pizza
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Joel ordered a ---- Seoul Style Pepperoni Pizza ----
Rice Dough
Kimchi Sauce
Shredded Parmesan
Garlic, Onion, Kimchi
Sliced Pepperoni


--- Making a Seoul Style Kimchi Pizza ---
Preparing Seoul Style Kimchi Pizza
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a ---- Seoul Style Kimchi Pizza ----
Rice Dough
Kimchi Sauce
Shredded Parmesan
Garlic, Onion, Kimchi
```

감사합니다.