10.1 Exercises

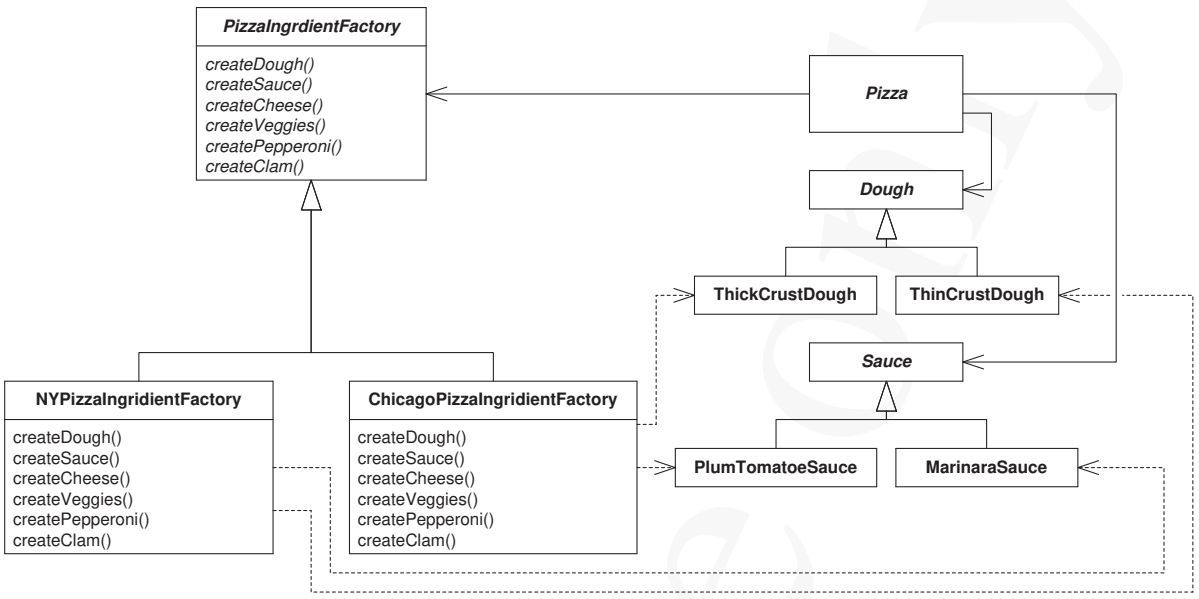

Figure 10.1: Abstract factory, partial UML diagram, for creating a pizza used to order pizza from a any franchised pizza store.

Exercise 10.1. The abstract factory design pattern is used to provide an interface for creating families of related or dependent objects without specifying their concrete classes. Figure 10.1 shows a partial UML diagram for an abstract pizza ingredient factory. Your group task consists of the following:

1. Create a Java application that will return a franchised pizza based on the type of pizza and the franchise it is made at

   ```
   MyPizzaApp <franchise> <pizzaKind>
   ```

   For example, if I wanted to order a cheese pizza from the NY pizza franchise, I would enter:

   ```
   MyPizzaApp <NYPizzaStore> <cheese>
   ```

   At the top level the application would execute:

   ```
   // The driver for ordering a pizza from any franchise
   PizzaStore nyPizzaStore = new NYPizzaStore();
   Pizza pizza = nyPizzaStore.orderPizza("cheese");
   ```

2. A partial listing of the code, in JAVA, is listed at the end. Add the details and complete the code with support for two franchise types: NY and Chicago. In particular, add the following classes:

(a) Concrete class `ChicagoPizzaStore`.

(b) Abstract classes: `Sauce, Cheese, Veggies, Pepperoni, Clam`.

(c) Concrete NY and Chicago sauce classes: `PlumTomatoeSauce, MarinaraSauce`.

(d) Concrete NY and Chicago cheese classes: `MozzarellaCheese, ReggianoCheese`.

(e) Concrete veggies classes: `Garlic, Onion, Mushroom, RedPepper`.

(f) Concrete NY and Chicago pepperoni class: `SlicedPepperoni`.

(g) Concrete NY and Chicago clam classes: `FrozenClams, FreshClams`.

(h) Concrete `ChicagoPizzaIngredientFactory`.

(i) Concrete pizzas: `PepperoniPizza, ClamPizza, VeggiePizza`.

3. Provide links to the Github repositories and show a screen-shot of the output when ordering a cheese pizza from the NY store and a veggie pizza from the Chicago store.

4. Draw the UML diagram of the `PizzaStore` and the concrete classes for the NY and Chicago stores.

5. Add a sequence diagrams that shows the process of ordering a NY style veggie pizza.

6. Complete the UML diagram from Figure 10.1.

7. This will count as two assignments. Thus it is a two week assignment.

8. The initial stub code, with additional details, is listed below.

The abstract class `PizzaStore` contains the abstract method `createPizza()` which needs to be implemented by all concrete implementation of the `PizzaStore`. The `orderPizza()` function is the same for all the concrete classes.

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

Listing 10.1: Abstract `PizzaStore` Class

The abstract implementation of the NY Pizza Store is shown below. Create a similar implementation for the Chicago Pizza Store.

```
public class NYPizzaStore extends PizzaStore {

  protected Pizza createPizza(String item) {
    Pizza pizza = null;
    PizzaIngredientFactory ingredientFactory =
      new NYPizzaIngredientFactory();

    if (item.equals("cheese")) {

      pizza = new CheesePizza(ingredientFactory);
      pizza.setName("New York Style Cheese Pizza");

    } else if (item.equals("veggie")) {

      pizza = new VeggiePizza(ingredientFactory);
      pizza.setName("New York Style Veggie Pizza");

    } else if (item.equals("clam")) {

      pizza = new ClamPizza(ingredientFactory);
      pizza.setName("New York Style Clam Pizza");

    } else if (item.equals("pepperoni")) {

      pizza = new PepperoniPizza(ingredientFactory);
      pizza.setName("New York Style Pepperoni Pizza");

    }
    return pizza;
  }
}
```

Listing 10.2: Concrete `NYPizzaStore` Class

The `PizzaStore` creates the pizza. As shown in Figure 10.1, `Pizza` is an abstract class and it contains the abstract classes: `Dough, Sauce, Veggies, Cheese, Pepperoni` and `Clams`, which are part of our pizza. Why are the classes listed above abstract? Change the association link in the UML figure to better represent the fact that these classes are part of our pizza. The abstract pizza class implements the functions `bake(), cut(), ...,` which are called from the abstract pizza store. Here is the code for the abstract `Pizza` class.

```
public abstract class Pizza {
  String name;

  Dough dough;
  Sauce sauce;
  Veggies veggies[];
  Cheese cheese;
  Pepperoni pepperoni;
  Clams clam;

  abstract void prepare();

```

```
13    void bake() {
14      System.out.println("Bake for 25 minutes at 350");
15    }
16
17    void cut() {
18      System.out.println("Cutting the pizza into diagonal slices");
19    }
20
21    void box() {
22      System.out.println("Place pizza in official PizzaStore box");
23    }
24
25    void setName(String name) {
26      this.name = name;
27    }
28
29    String getName() {
30      return name;
31    }
32
33    public String toString() {
34      StringBuffer result = new StringBuffer();
35      result.append("---- " + name + " ----\n");
36      if (dough != null) {
37        result.append(dough);
38        result.append("\n");
39      }
40      if (sauce != null) {
41        result.append(sauce);
42        result.append("\n");
43      }
44      if (cheese != null) {
45        result.append(cheese);
46        result.append("\n");
47      }
48      if (veggies != null) {
49        for (int i = 0; i < veggies.length; i++) {
50          result.append(veggies[i]);
51          if (i < veggies.length-1) {
52            result.append(", ");
53          }
54        }
55        result.append("\n");
56      }
57      if (clam != null) {
58        result.append(clam);
59        result.append("\n");
60      }
61      if (pepperoni != null) {
62        result.append(pepperoni);
63        result.append("\n");
64      }
65      return result.toString();
```

```
66    }
67  }
```

Listing 10.3: Abstract `Pizza` Class

The abstract `Pizza` class implements the concrete type of pizza, such as `CheesePizza`, via the abstract `PizzaIngredientFactory` class. `PizzaIngredientFactory` creates the concrete ingredients via the concrete `NYPizzaIngredientFactory` and `ChicagoPizzaIngredientFactory`. Here is the concrete `CheesePizza` with the `PizzaIngredientFactory` member.

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

Listing 10.4: Concrete `CheesePizza` Class

The `PizzaIngredientFactory` contains only abstract methods.

```
1  public interface PizzaIngredientFactory {
2
3    public Dough createDough();
4    public Sauce createSauce();
5    public Cheese createCheese();
6    public Veggies[] createVeggies();
7    public Pepperoni createPepperoni();
8    public Clams createClam();
9
10 }
```

Listing 10.5: Abstract `PizzaIngredientFactory`

The concrete `NYPizzaIngredientFactory` creates the concrete types of dough, sauce, ..., as shown in Figure 10.1. This is listed next. Write a similar concrete class that implements the `ChicagoPizzaIngredientFactory`.

```
1  public class NYPizzaIngredientFactory implements PizzaIngredientFactory {
2
3    public Dough createDough() {
4      return new ThinCrustDough();
5    }
6
7    public Sauce createSauce() {
8      return new MarinaraSauce();
9    }
10
```

```
11    public Cheese createCheese() {
12      return new ReggianoCheese();
13    }
14
15    public Veggies[] createVeggies() {
16      Veggies veggies[] = { new Garlic(), new Onion(), new Mushroom(), new
       RedPepper() };
17      return veggies;
18    }
19
20    public Pepperoni createPepperoni() {
21      return new SlicedPepperoni();
22    }
23
24    public Clams createClam() {
25      return new FreshClams();
26    }
27  }
```

Listing 10.6: Concrete `NYPizzaIngredient` Factory

We are almost done. All that's left is to look at the abstract and concrete ingredients classes. Let's look at the abstract `Dough` class. Implement the rest of the abstract ingredients:`Sauce`, `Cheese`, `Veggies`, `Pepperoni`, and `Clams`.

```
1  public interface Dough {
2    public String toString();
3  }
```

Listing 10.7: Abstract `Dough` class

The concrete implementation of `ThickCrustDough` is shown next. Implement the concrete implementation of `ThinCrustDough`.

```
1  public class ThickCrustDough implements Dough {
2    public String toString() {
3      return "ThickCrust style extra thick crust dough";
4    }
5  }
```

Listing 10.8: Concrete `ThickCrustDough` class