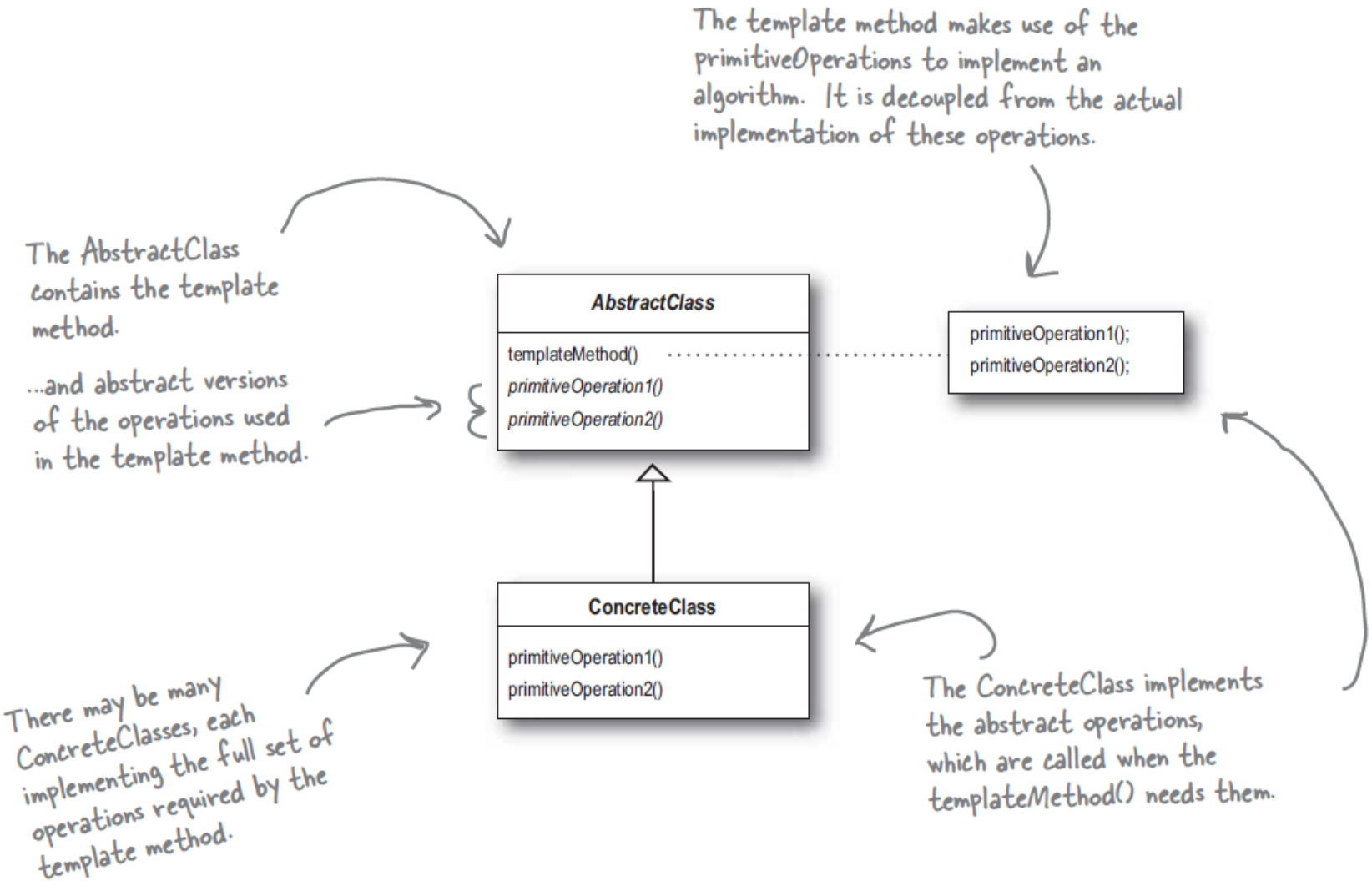
8. 템플릿 메서드 패턴

알고리즘 캡슐화

학습 목표

- 메서드에 정의되는 알고리즘의 골격(skeleton)을 정의하고 그 중 일부 단계를 서브 클래스에서 정의하도록 하는 **템플릿 메서드 패턴**을 이해하고 활용할 수 있다.
- **헐리우드 원칙**(Hollywood Principle) 개념을 이해할 수 있다.
- 템플릿 메서드 패턴을 적용한 **java.util.Arrays** 클래스의 **sort()** 메서드를 사용할 수 있다.

템플릿 메서드 패턴

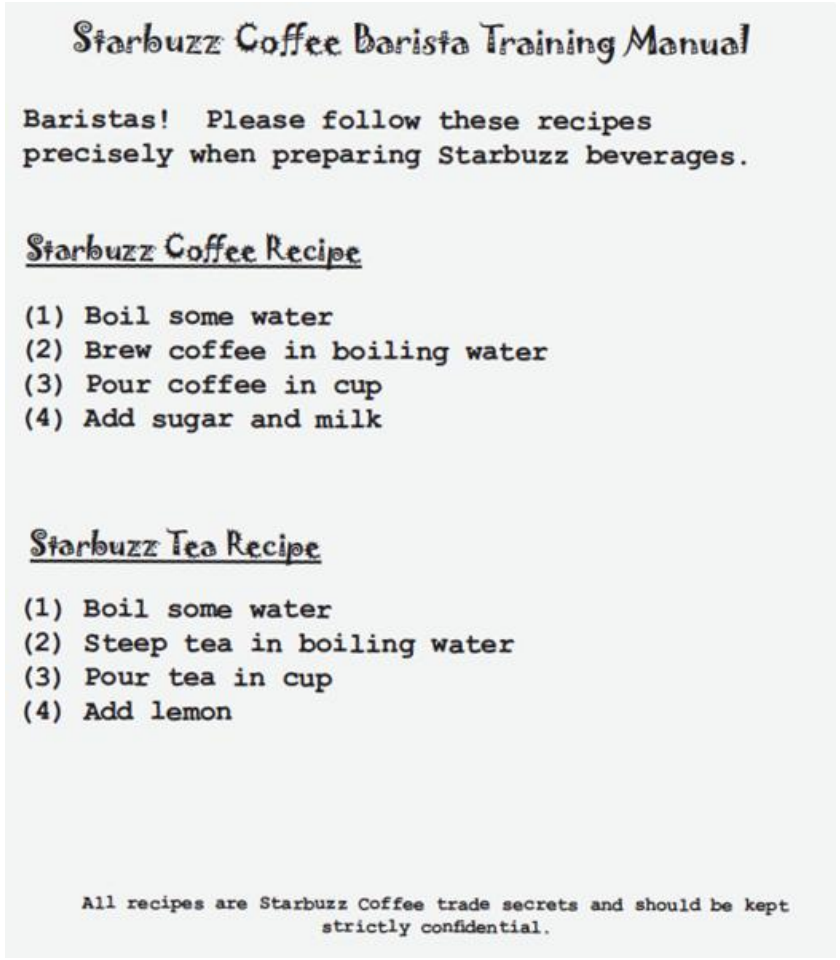

The template method makes use of the primitiveOperations to implement an algorithm. It is decoupled from the actual implementation of these operations.
The AbstractClass contains the template method.
...and abstract versions of the operations used in the template method.
AbstractClass
templateMethod()
primitiveOperation1()
primitiveOperation2()
primitiveOperation1();
primitiveOperation2();
ConcreteClass
primitiveOperation1()
primitiveOperation2()
There may be many ConcreteClasses, each implementing the full set of operations required by the template method.
The ConcreteClass implements the abstract operations, which are called when the templateMethod() needs them.

템플릿 메서드 패턴 정의

- 문제
  - 두 개의 다른 컴포넌트(좀더 엄밀히 말하자면 메서드)가 아주 비슷하지만, 공통된 인터페이스나 구현을 재사용할 수 없을 때 이를 어떻게 해결할 것인가?
- 해결 방안
  - 메서드 내 어느 단계가 불변(invariant)인지와 변하는 부분인지 결정한다. 추상클래스에서 불변인 부분을 정의하고, 변하는 부분은 (추상 메서드를 사용하여) 메서드 선언만 한다.
  - 이를 상속받아 구현하는 서브 클래스에서 변하는 부분을 (오버라이딩하여) 구체적으로 정의한다.

- 참고 URL: https://sourcemaking.com/design_patterns/template_method

커피 & 티 만들기



Starbuzz Coffee Barista Training Manual

Baristas! Please follow these recipes precisely when preparing Starbuzz beverages.

Starbuzz Coffee Recipe

(1) Boil some water
(2) Brew coffee in boiling water
(3) Pour coffee in cup
(4) Add sugar and milk

Starbuzz Tea Recipe

(1) Boil some water
(2) Steep tea in boiling water
(3) Pour tea in cup
(4) Add lemon

All recipes are Starbuzz Coffee trade secrets and should be kept strictly confidential.

The recipe for coffee looks a lot like the recipe for tea, doesn't it?

Coffee 클래스

Here's our Coffee class for making coffee.

Here's our recipe for coffee, straight out of the training manual.

Each of the steps is implemented as a separate method.

```
public class Coffee {

    void prepareRecipe() {
        boilWater();
        brewCoffeeGrinds();
        pourInCup();
        addSugarAndMilk();
    }

    public void boilWater() {
        System.out.println("Boiling water");
    }

    public void brewCoffeeGrinds() {
        System.out.println("Dripping Coffee through filter");
    }

    public void pourInCup() {
        System.out.println("Pouring into cup");
    }

    public void addSugarAndMilk() {
        System.out.println("Adding Sugar and Milk");
    }
}
```

Each of these methods implements one step of the algorithm. There's a method to boil water, brew the coffee, pour the coffee in a cup and add sugar and milk.

Tea 클래스

차이점

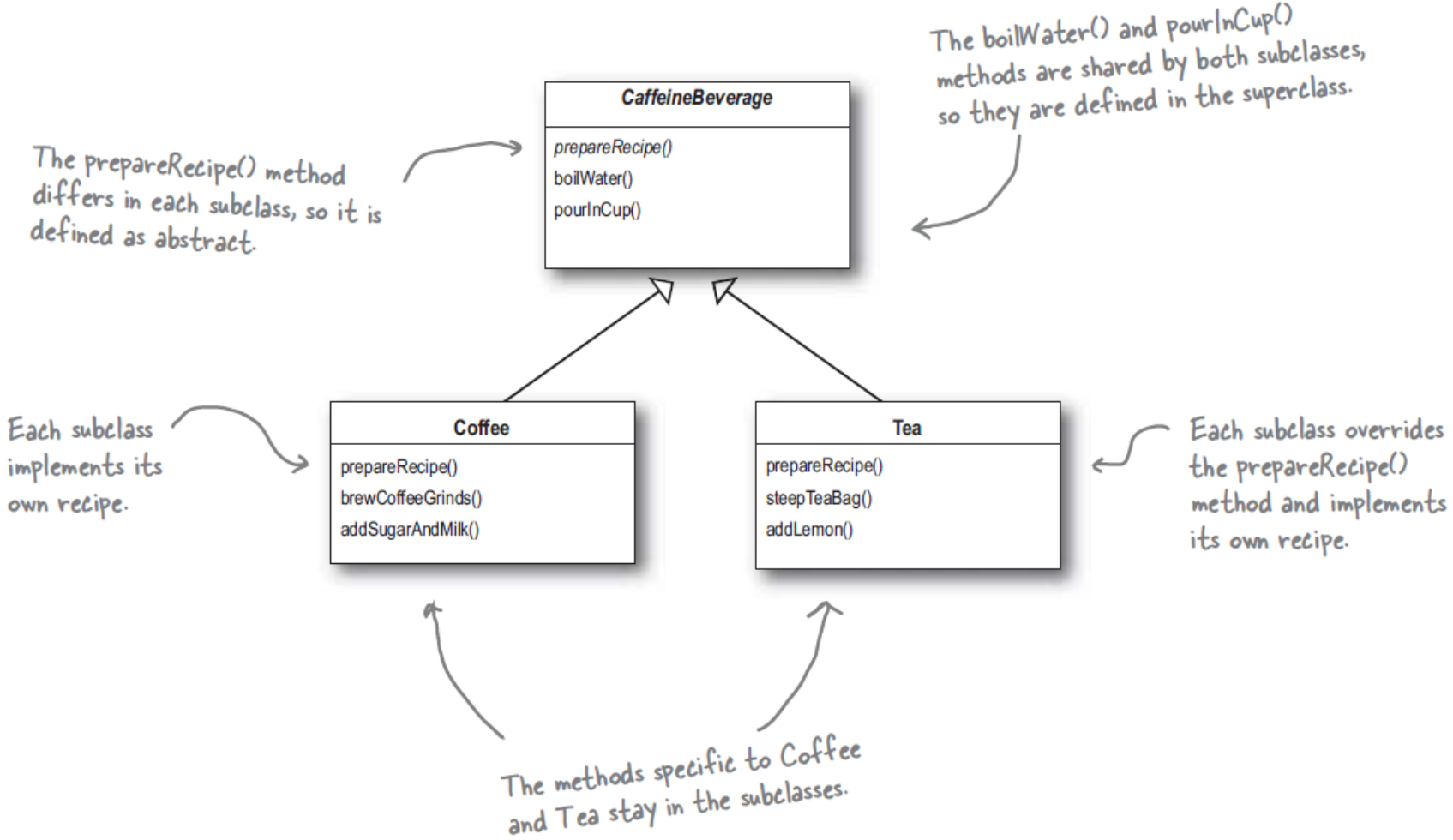
```
public class Tea {

    void prepareRecipe() {
        boilWater();
        steepTeaBag();
        pourInCup();
        addLemon();
    }

    public void boilWater() {
        System.out.println("Boiling water");
    }

    public void steepTeaBag() {
        System.out.println("Steeping the tea");
    }

    public void addLemon() {
        System.out.println("Adding Lemon");
    }

    public void pourInCup() {
        System.out.println("Pouring into cup");
    }
}
```

This looks very similar to the one we just implemented in Coffee; the second and forth steps are different, but it's basically the same recipe.

These two methods are specialized to Tea.

Notice that these two methods are exactly the same as they are in Coffee! So we definitely have some code duplication going on here.

Coffer & Tea 클래스 추상화



공통점 찾기

- 공통된 알고리즘을 사용해서 커피와 차를 만들고 있음

Starbuzz Coffee Recipe

(1) Boil some water
(2) Brew coffee in boiling water
(3) Pour coffee in cup
(4) Add sugar and milk

Starbuzz Tea Recipe

(1) Boil some water
(2) Steep tea in boiling water
(3) Pour tea in cup
(4) Add lemon

1. Boil some water.
2. Use the hot water to extract the coffee or tea.
3. Pour the resulting beverage into a cup.
4. Add the appropriate condiments to the beverage.

These aren't abstracted, but are the same, they just apply to different beverages.

These two are already abstracted into the base class.

prepareRecipe() 추상화

- prepareRecipe()는 기본적으로 동일한 알고리즘 사용하지만, 조금은 다름.

Coffee

```
void prepareRecipe() {
    boilWater();
    brewCoffeeGrinds();
    pourInCup();
    addSugarAndMilk();
}
```

Tea

```
void prepareRecipe() {
    boilWater();
    steepTeaBag();
    pourInCup();
    addLemon();
}
```

- 서로 다른 단계의 공통점: 물을 끓이고 첨가물을 넣는다는 점은 동일

```
void prepareRecipe() {
    boilWater();
    brew();
    pourInCup();
    addCondiments();
}
```

수정된 CaffeineBeverage 클래스

CaffeineBeverage is abstract, just like in the class design.

```
public abstract class CaffeineBeverage {

    final void prepareRecipe() {
        boilWater();
        brew();
        pourInCup();
        addCondiments();
    }

    abstract void brew();

    abstract void addCondiments();

    void boilWater() {
        System.out.println("Boiling water");
    }

    void pourInCup() {
        System.out.println("Pouring into cup");
    }
}
```

Now, the same prepareRecipe() method will be used to make both Tea and Coffee. prepareRecipe() is declared final because we don't want our subclasses to be able to override this method and change the recipe! We've generalized steps 2 and 4 to brew() the beverage and addCondiments().

Because Coffee and Tea handle these methods in different ways, they're going to have to be declared as abstract. Let the subclasses worry about that stuff!

Remember, we moved these into the CaffeineBeverage class (back in our class diagram).

Coffee & Tea 클래스 수정

```
public class Tea extends CaffeineBeverage {
    public void brew() {
        System.out.println("Steeping the tea");
    }
    public void addCondiments() {
        System.out.println("Adding Lemon");
    }
}
```

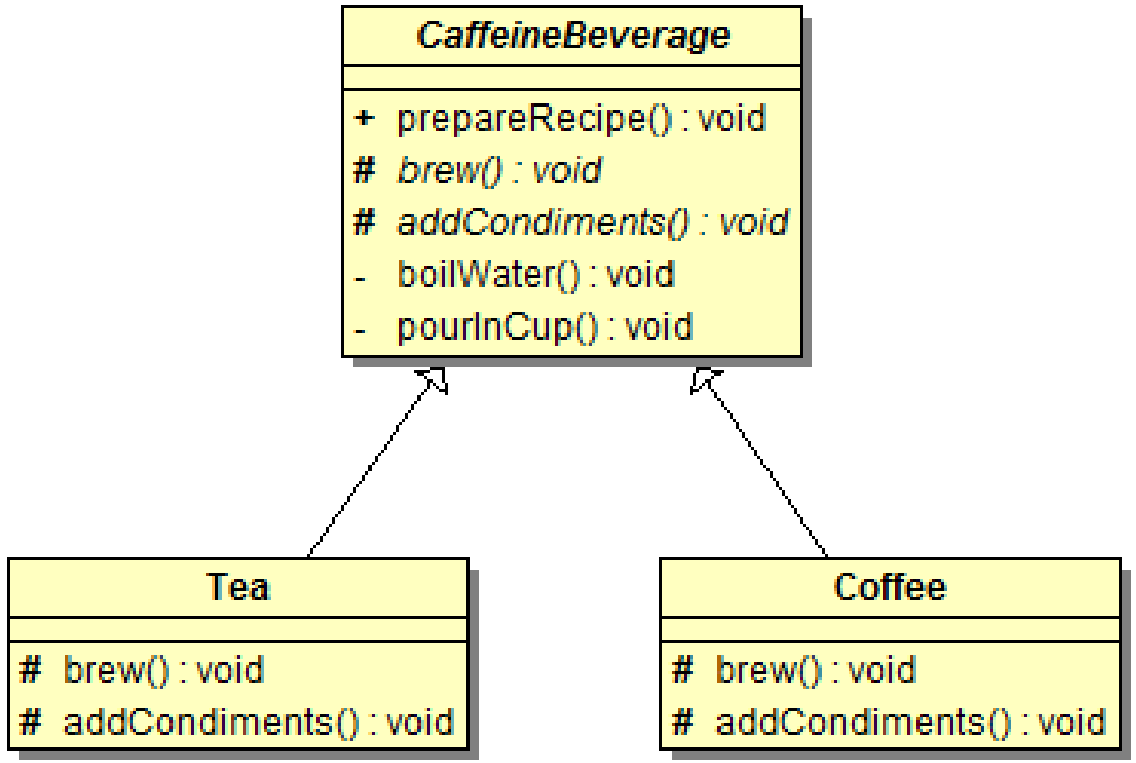
As in our design, Tea and Coffee now extend CaffeineBeverage.

Tea needs to define brew() and addCondiments() – the two abstract methods from Beverage.

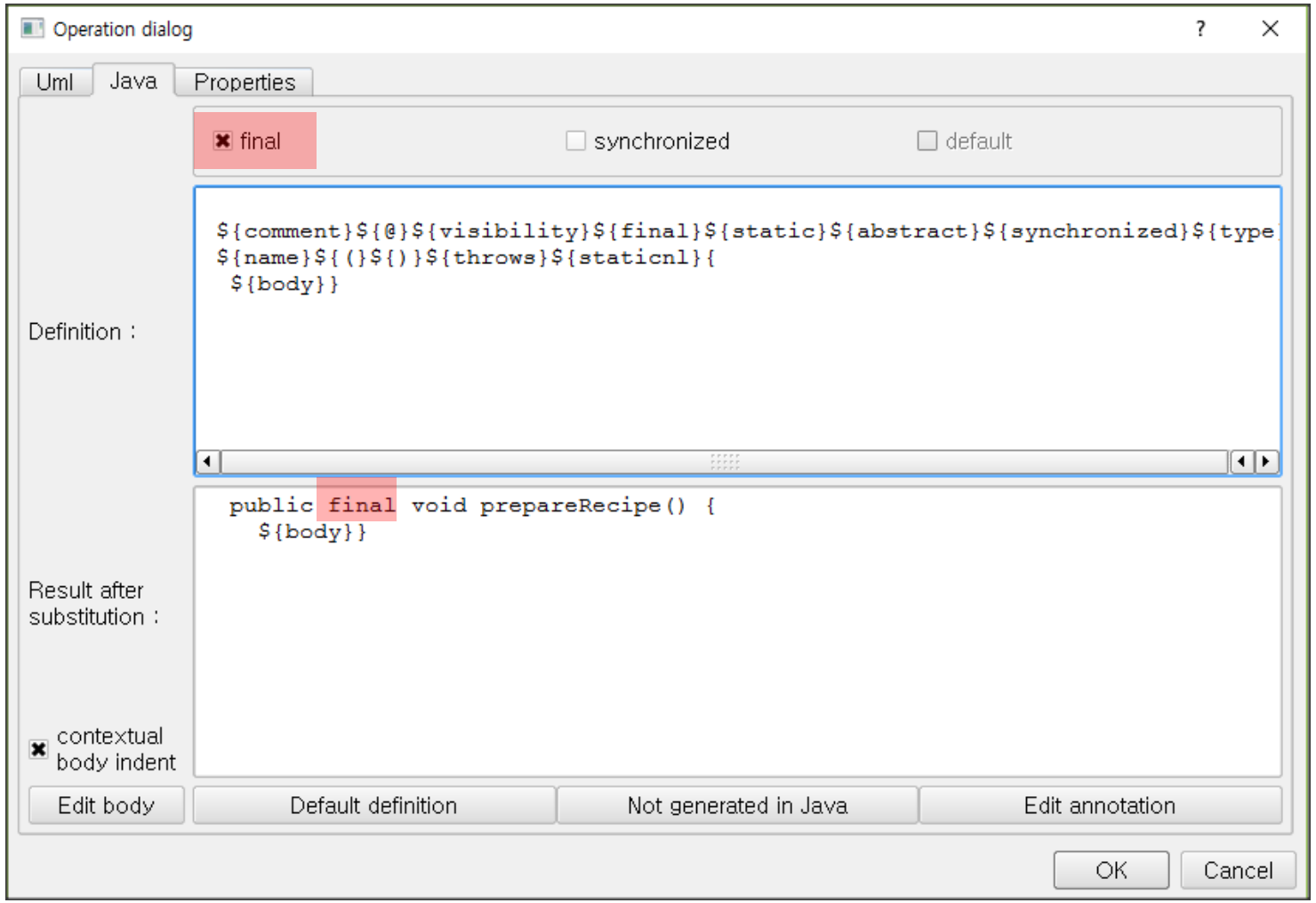
Same for Coffee, except Coffee deals with coffee, and sugar and milk instead of tea bags and lemon.

```
public class Coffee extends CaffeineBeverage {
    public void brew() {
        System.out.println("Dripping Coffee through filter");
    }
    public void addCondiments() {
        System.out.println("Adding Sugar and Milk");
    }
}
```

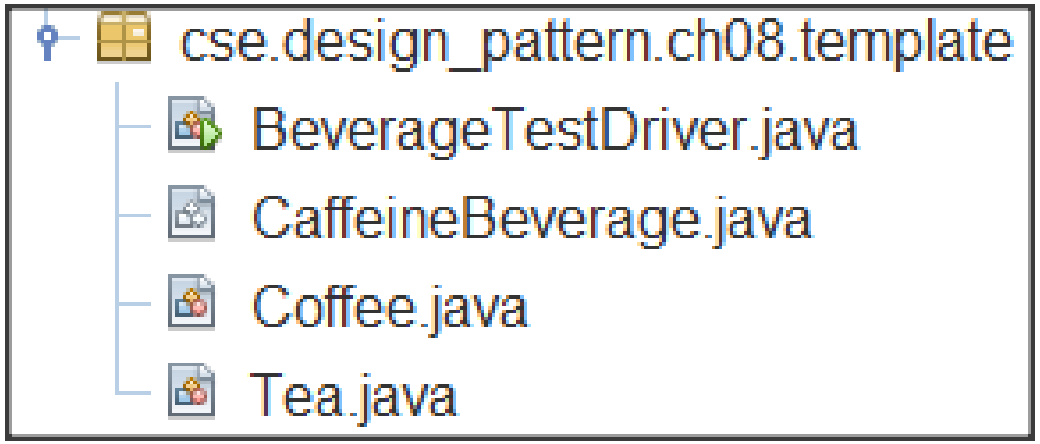
클래스 다이어그램 (template)



prepareRecipe()



실습: template 패키지



CaffeineBeverage.java

```
1   package cse.design_pattern.ch08.template;
2
3   abstract class CaffeineBeverage {
4
5       public final void prepareRecipe() {
6           boilWater();
7           brew();
8           pourInCup();
9           addCondiments();
10      }
11
12      protected abstract void brew();
13
14      protected abstract void addCondiments();
15
16      private void boilWater() {
17          System.out.println("물 끓이는 중");
18      }
19
20      private void pourInCup() {
21          System.out.println("컵에 따르는 중");
22      }
23
24  }
```

Tea.java

```
1   package cse.design_pattern.ch08.template;
2
3   class Tea extends CaffeineBeverage {
4
6       protected void brew() {
7           System.out.println("차를 우려내는 중");
8       }
9
10      protected void addCondiments() {
11          System.out.println("레몬을 추가하는 중");
12      }
13
14  }
```

Coffee.java

```
1   package cse.design_pattern.ch08.template;
2
3   class Coffee extends CaffeineBeverage {
4
5       protected void brew() {
6           System.out.println("필터를 통해서 커피를 우려내는 중");
7       }
8
9       protected void addCondiments() {
10          System.out.println("설탕과 우유를 추가하는 중");
11      }
12
13  }
```

BeverageTestDriver.java & 실행 결과

```
6   package cse.design_pattern.ch08.template;
7
8   /**...4 lines */
12  public class BeverageTestDriver {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          CaffeineBeverage myTea = new Tea();
19          myTea.prepareRecipe();
20
21          System.out.println("====================");
22          CaffeineBeverage myCoffee = new Coffee();
23          myCoffee.prepareRecipe();
24      }
25
26  }
```

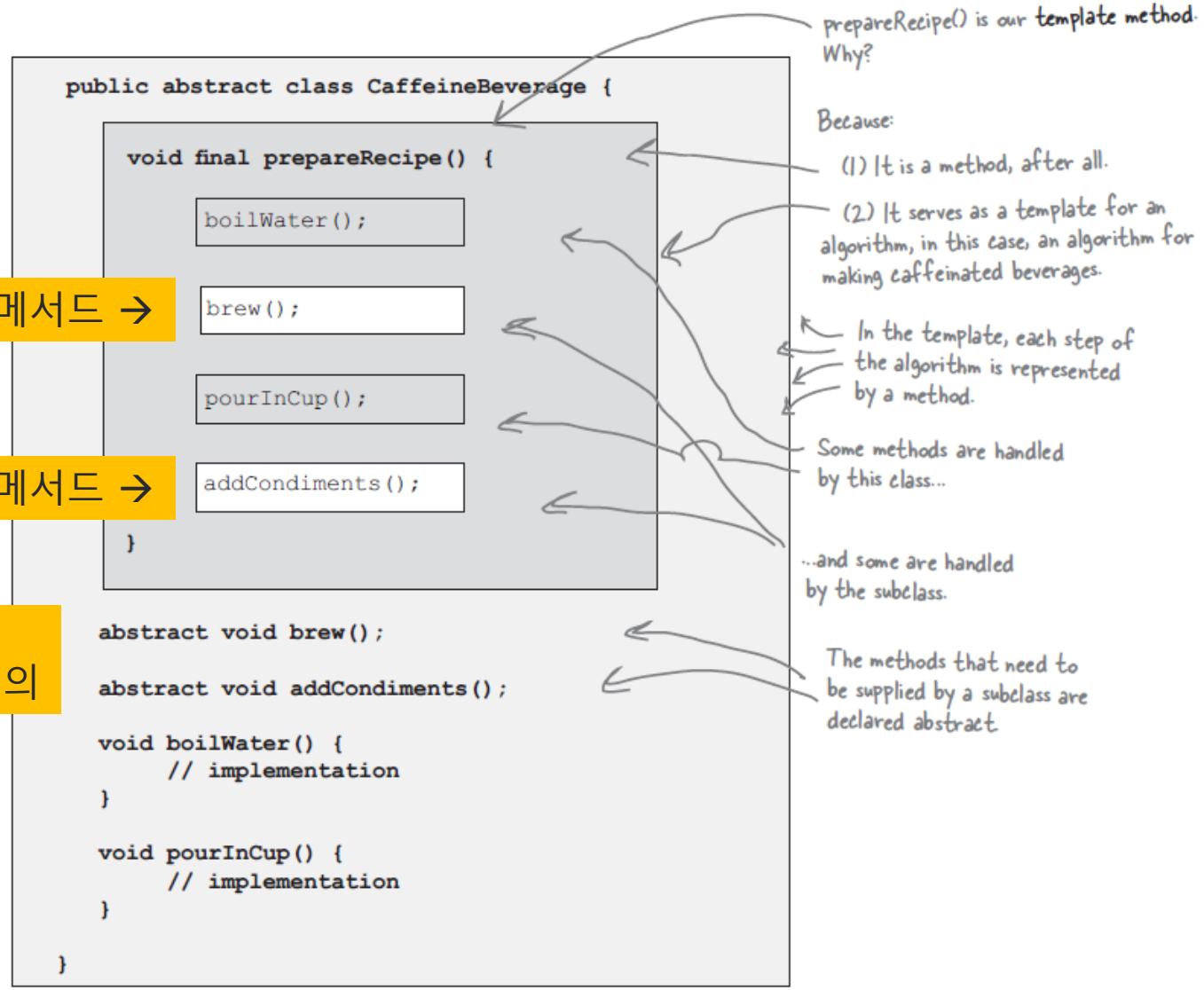
```
--- exec-maven-plugin:1.2.1:exec
물 끓이는 중
차를 우려내는 중
컵에 따르는 중
레몬을 추가하는 중
====================
물 끓이는 중
필터를 통해서 커피를 우려내는 중
컵에 따르는 중
설탕과 우유를 추가하는 중
```

템플릿 메서드 개념

추상 메서드 →

추상 메서드 →

추상 메서드:
서브클래스에서 정의



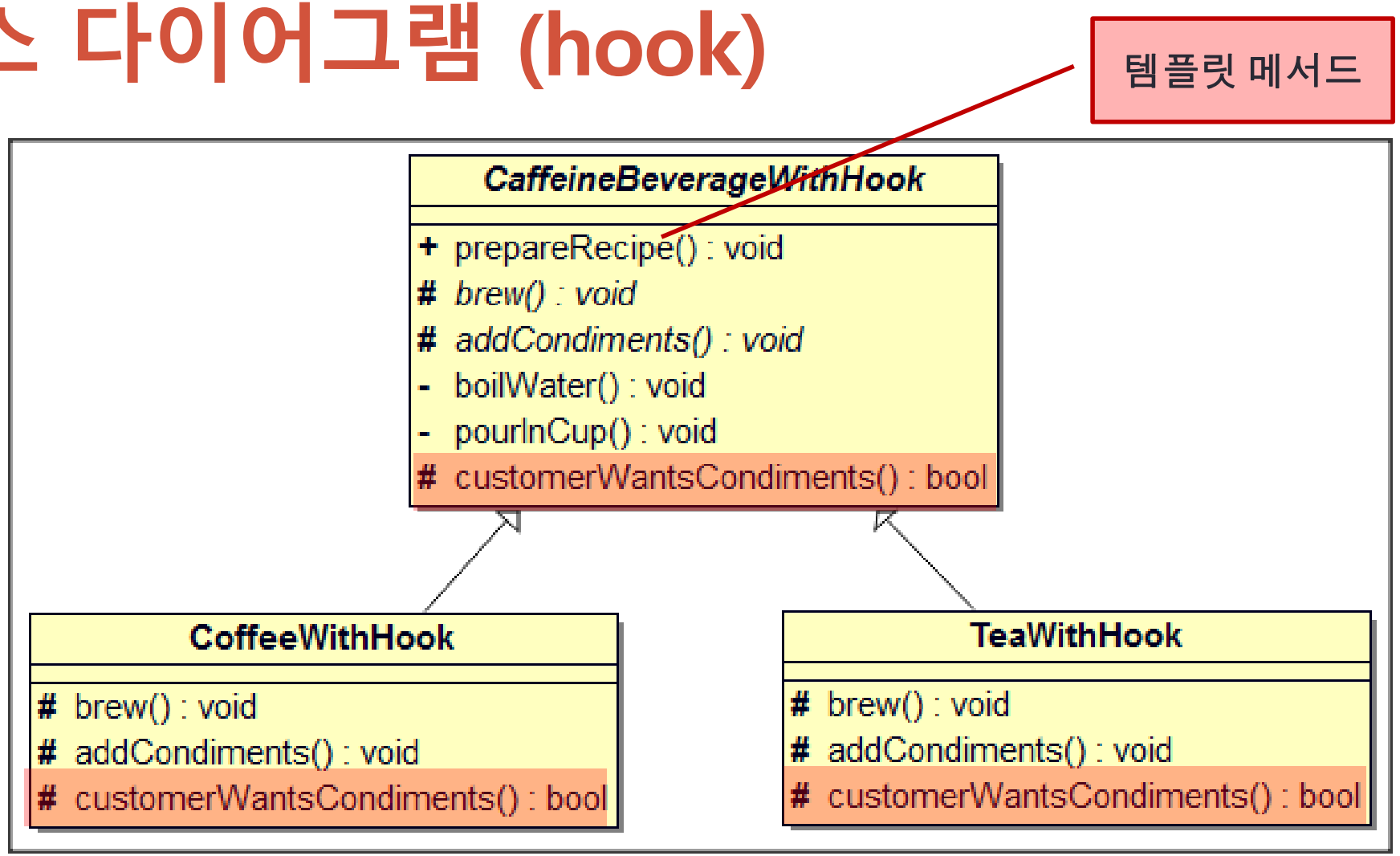
후크(hook) 개념

- 후크(hook)는 **추상 클래스에서 정의된 메서드로** 아무 것도 없거나 기본적인 구현만 제공하는 메서드를 의미
- 후크는 **서브 클래스에서 적절히 변경**하여 원하는 알고리즘을 만들 수 있음.
- 서브클래스가 알고리즘의 특정 단계를 제공해야 한다면 추상 메서드를 사용 (전통적인 템플릿 메서드 패턴)
- **알고리즘의 특정 단계가 선택적으로 적용된다면 후크 사용** (서브클래스에서 재정의할 수도 있고 안 해도 됨)
- p.331 Q&A 참고

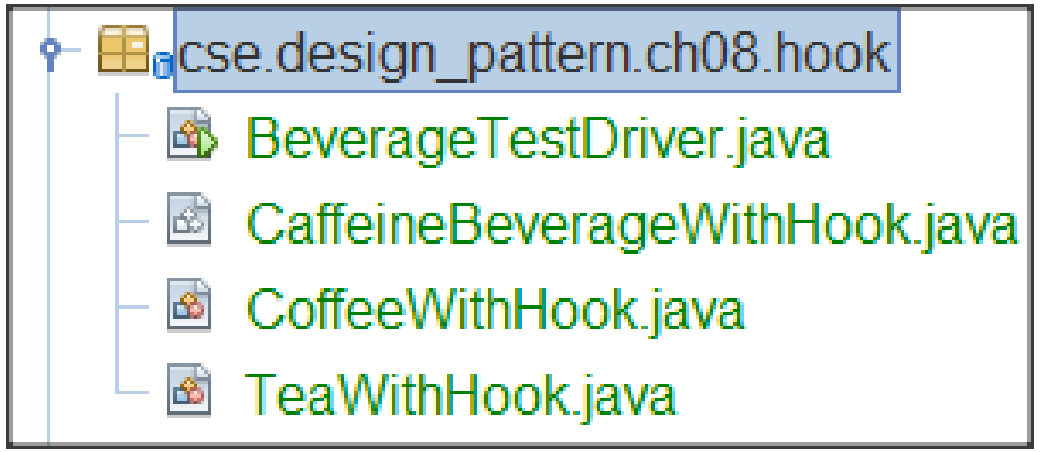
🧠 **Hook 메서드란?**

- **정의**: 추상 클래스에서 정의된 *선택적 오버라이딩 대상* 메서드
- **역할**: 서브클래스가 필요에 따라 동작을 변경할 수 있도록 "갈고리(hook)"처럼 걸어두는 메서드
- **형태**: 일반적으로 `abstract`가 아닌 `protected` 또는 `public` 메서드로 기본 구현을 포함함

클래스 다이어그램 (hook)



실습: hook 패키지


cse.design_pattern.ch08.hook
BeverageTestDriver.java
CaffeineBeverageWithHook.java
CoffeeWithHook.java
TeaWithHook.java

CaffeineBeverageWithHook.java

```java
package cse.design_pattern.ch08.hook;

public abstract class CaffeineBeverageWithHook {

    public final void prepareRecipe() {
        boilWater();
        brew();
        pourInCup();
        if (customerWantsCondiments()) {
            addCondiments();
        }
    }

    protected abstract void brew();

    protected abstract void addCondiments();

    private void boilWater() {
        System.out.println("물 끓이는 중");
    }

    private void pourInCup() {
        System.out.println("컵에 따르는 중");
    }

    protected boolean customerWantsCondiments() {
        return true;
    }

}
```

hook

TeaWithHook.java

```
1   package cse.design_pattern.ch08.hook;
2
3   import java.io.BufferedReader;
4   import java.io.IOException;
5   import java.io.InputStreamReader;
6
7   public class TeaWithHook extends CaffeineBeverageWithHook {
8
9     protected void brew() {
10      System.out.println("차를 우려내는 중");
11    }
12
13    protected void addCondiments() {
14      System.out.println("레몬을 추가하는 중");
15    }
16
17    protected boolean customerWantsCondiments() {
18
19      String answer = getUserInput();
20
21      if (answer.toLowerCase().startsWith("y")) {
22        return true;
23      } else {
24        return false;
25      }
26    }
```

hook를 재정의하여 원하는 기능 추가

```
28      private String getUserInput() {
29          String answer = null;
30
31          System.out.print("차에 레몬을 넣어 드릴까요? (y/n): ");
32
33          BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
34          try {
35              answer = in.readLine();
36          } catch (IOException ioe) {
37              System.err.println(ioe);
38          }
39          if (answer == null) {
40              return "no";
41          }
42          return answer;
43      }
44
45  }
```

CoffeeWithHook.java

```
1   package cse.design_pattern.ch08.hook;
2
3   import java.io.BufferedReader;
4   import java.io.IOException;
5   import java.io.InputStreamReader;
6
    public class CoffeeWithHook extends CaffeineBeverageWithHook {
8
      protected void brew() {
10      System.out.println("필터를 통해서 커피를 우려내는 중");
11    }
12
      protected void addCondiments() {
14      System.out.println("우유와 설탕을 추가하는 중");
15    }
16
      protected boolean customerWantsCondiments() {
18
19      String answer = getUserInput();
20
        if (answer.toLowerCase().startsWith("y")) {
22        return true;
23      } else {
24        return false;
25      }
26    }
```

```
28      private String getUserInput() {
29          String answer = null;
30
31          System.out.print("커피에 우유와 설탕을 넣어 드릴까요? (y/n): ");
32
33          BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
34          try {
35              answer = in.readLine();
36          } catch (IOException ioe) {
37              System.err.println(ioe);
38          }
39          if (answer == null) {
40              return "no";
41          }
42          return answer;
43      }
44
45  }
```

BeverageTestDriver.java

```
6   package cse.design_pattern.ch08.hook;
7
8   /**...4 lines */
12  public class BeverageTestDriver {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          CaffeineBeverageWithHook myTea = new TeaWithHook();
19          myTea.prepareRecipe();
20
21          System.out.println("====================");
22          CaffeineBeverageWithHook myCoffee = new CoffeeWithHook();
23          myCoffee.prepareRecipe();
24      }
25
26  }
```

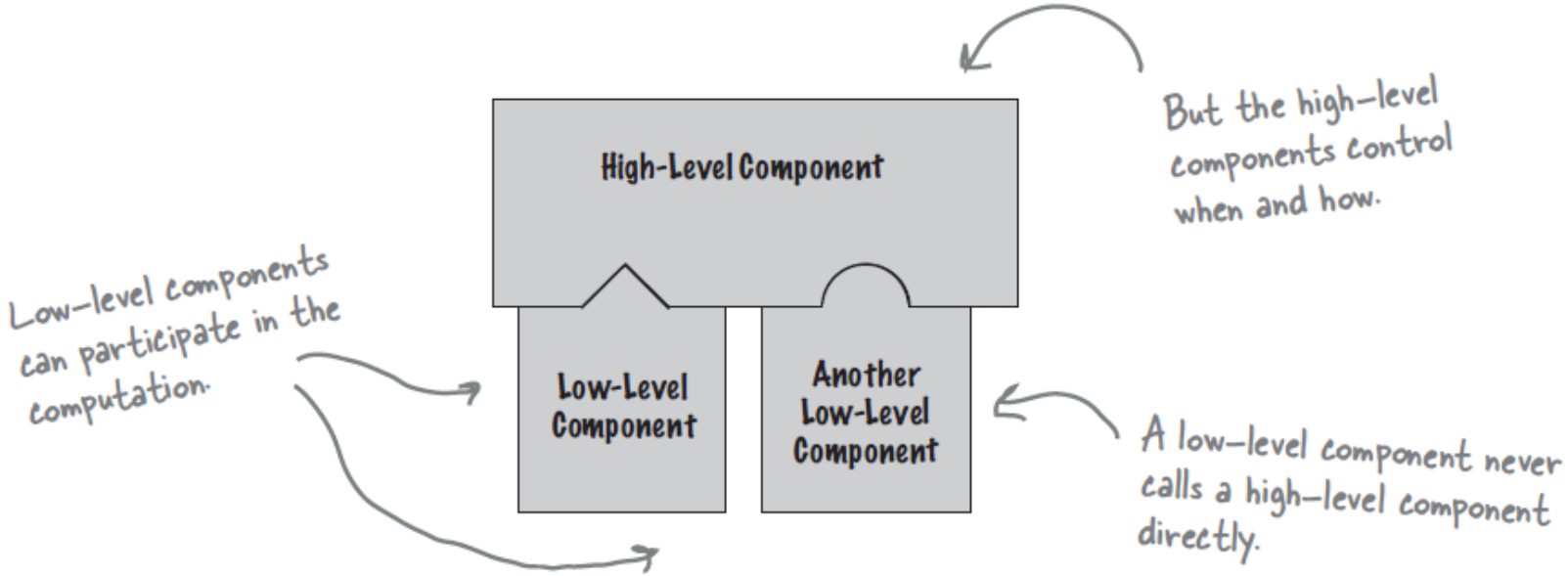
실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-c
물 끓이는 중
차를 우려내는 중
컵에 따르는 중
차에 레몬을 넣어 드릴까요? (y/n): n
==================
물 끓이는 중
필터를 통해서 커피를 우려내는 중
컵에 따르는 중
커피에 우유와 설탕을 넣어 드릴까요? (y/n): y
우유와 설탕을 추가하는 중
```

헐리우드 원칙

- Don't call us, we'll call you.
- 제어 역전(IoC; Inversion of Control)이라고도 함.
- 의존성 부패(dependency rot)을 방지하는 방법을 제공
  - 의존성 부패는 고수준 컴포넌트가 저수준 컴포넌트를 의존할 때 발생
  - 의존성 부패 발생시 시스템이 설계된 방법을 이해하기 쉽지 않아짐.
  - **저수준 컴포넌트가 시스템에 훅하도록 허용**하지만, **고수준 컴포넌트에서 저수준 컴포넌트가 필요한 시기와 방법을 결정**하도록 함.

- 참고 URL
  - http://wiki.c2.com/?HollywoodPrinciple
  - https://en.wikipedia.org/wiki/Inversion_of_control
  - http://matthewtmead.com/blog/hollywood-principle-dont-call-us-well-call-you-4/

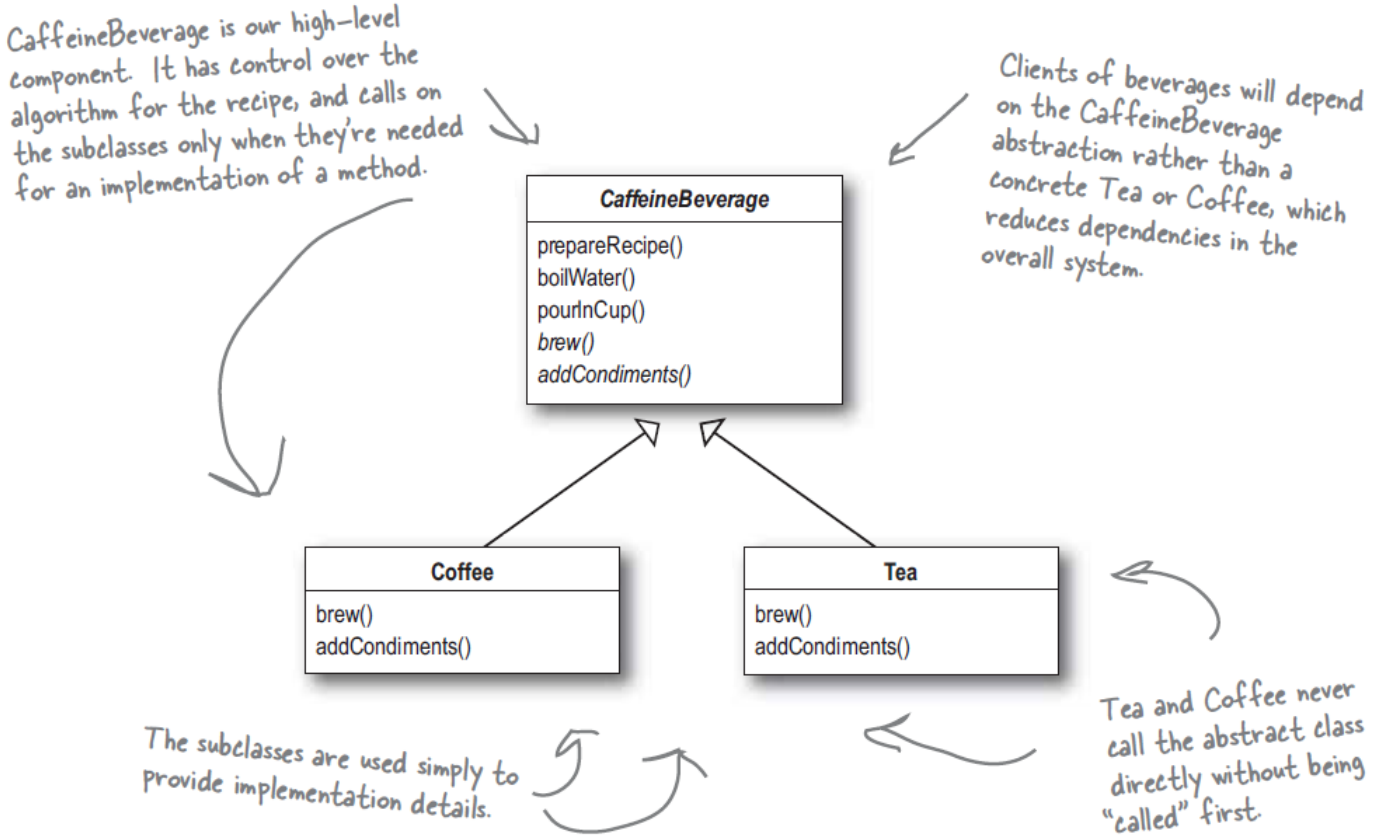

High-Level Component
But the high-level components control when and how.
Low-level components can participate in the computation.
Low-Level Component
Another Low-Level Component
A low-level component never calls a high-level component directly.

헐리우드 원칙 vs. DIP

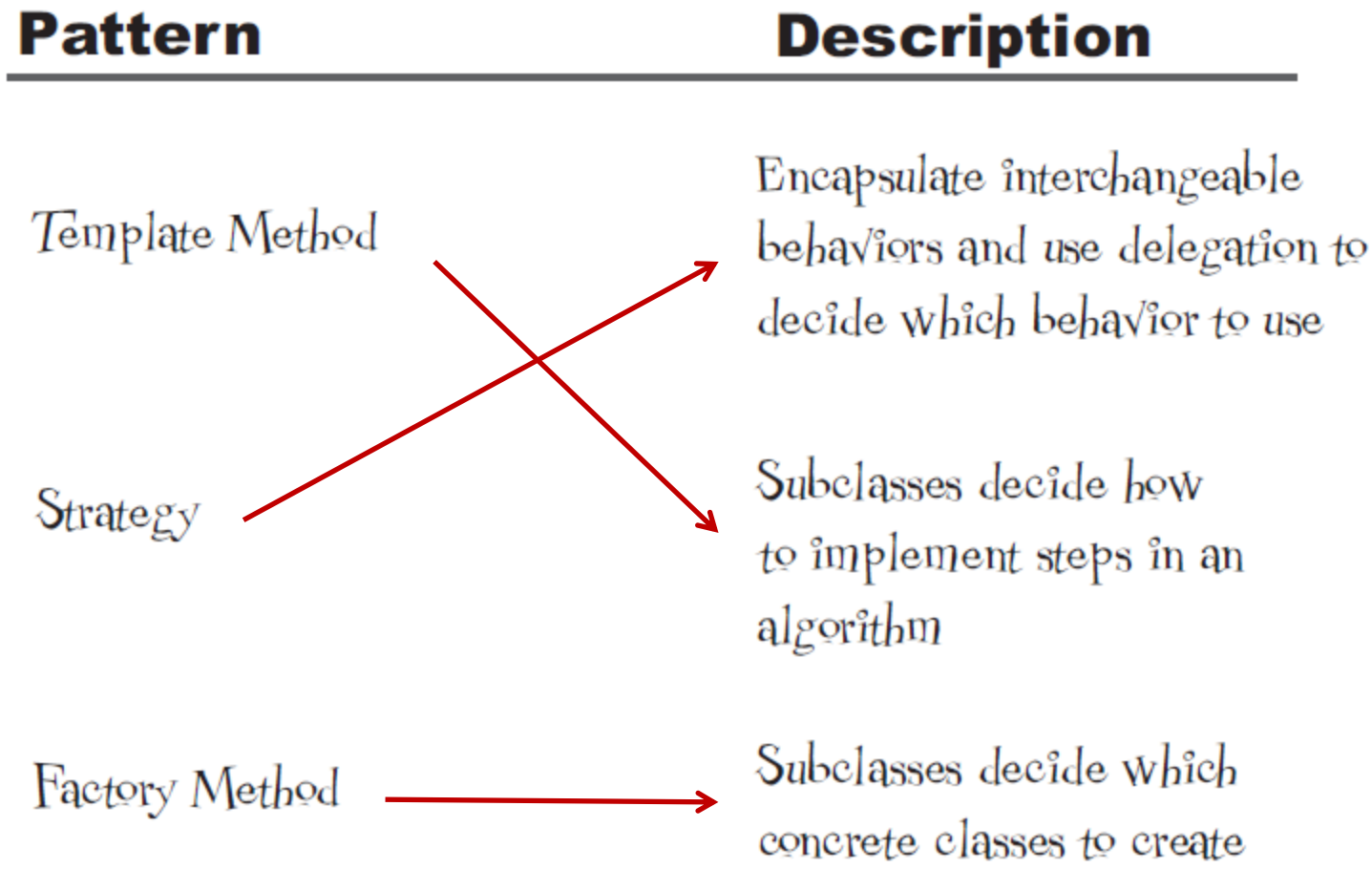
- DIP
  - 구상 클래스 사용을 줄이고 대신 추상화된 것을 사용
- 헐리우드 원칙
  - 저수준 구성요소가 계산에 참여할 수는 있으면서도
  - 저수준 구성요소와 고수준 계층 사이에 의존성을 만들어내지 않도록
  - 프레임워크 또는 구성요소를 구축하기 위한 기법

템플릿 메서드 패턴과의 관련성

- 템플릿 메서드 패턴은 서브 클래스에게 “호출하지 마세요, 우리가 호출할 것입니다 " 라고 말함.



패턴 맞추기


Pattern
Description
Template Method
Strategy
Factory Method
Encapsulate interchangeable behaviors and use delegation to decide which behavior to use
Subclasses decide how to implement steps in an algorithm
Subclasses decide which concrete classes to create

템플릿 메서드 패턴 사용 예

- java.util.Arrays 클래스의 sort(Object[] a) 정적 메서드

```
public static void sort(Object[] a)

지정된 오브젝트 배열을 승순으로 정렬함. 배열의 모든 요소는 Comparable 인터페이스를 구현
```

- java.lang.Comparable<T> 인터페이스

```
int compareTo(T o)

객체를 매개변수에서 참조하는 객체 o와 비교
반환값:
  * -1: <
  *  0: ==
  * +1: >
```

참고. List 클래스의 sort() 메서드

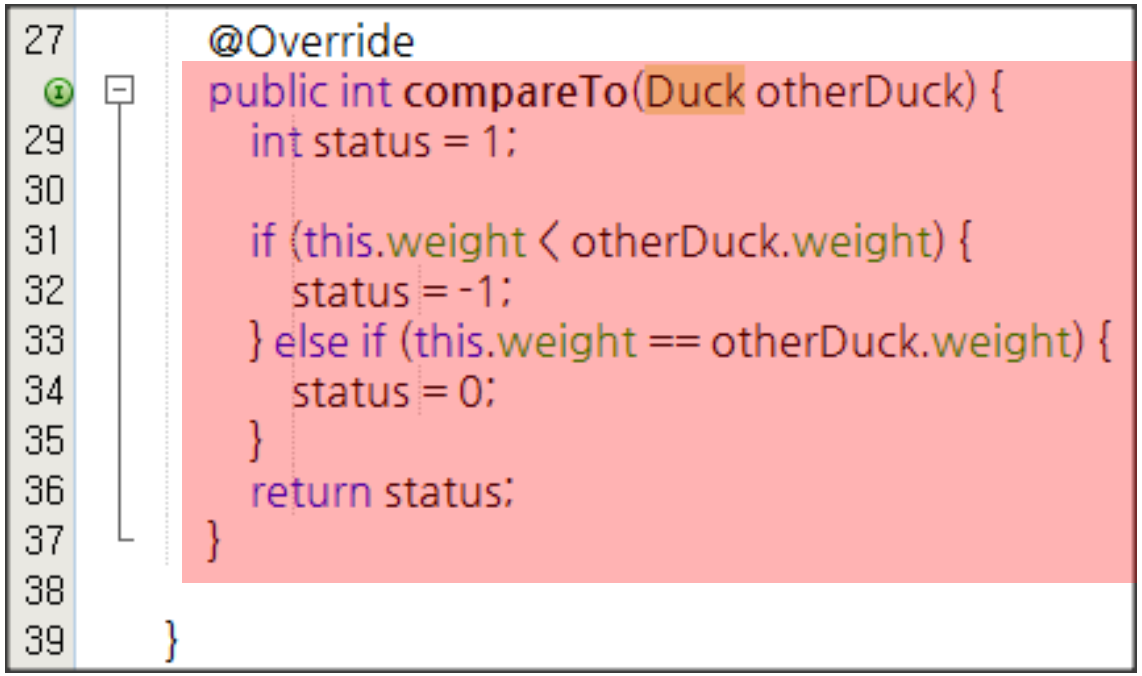
- 형식: default void sort(Comparator<? super E> c)
  Sorts this list according to the order induced by the specified Comparator.

- Interface Comparator<T>
  - int compare(T o1, T o2)
    Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

실습: array_sort

Duck.java

```
6   package cse.design_pattern.ch08.array_sort;
7
8   /**...4 lines */
12  public class Duck implements Comparable<Duck> {
13
        private String name;
        private int weight;
16
17      public Duck(String name, int weight) {
18          this.name = name;
19          this.weight = weight;
20      }
21
22      @Override
        public String toString() {
24          return name + ", 체중: " + weight;
25      }
```



```
27      @Override
        public int compareTo(Duck otherDuck) {
29          int status = 1;
30
31          if (this.weight < otherDuck.weight) {
32              status = -1;
33          } else if (this.weight == otherDuck.weight) {
34              status = 0;
35          }
36          return status;
37      }
38
39  }
```

DuckSortTestDrive.java

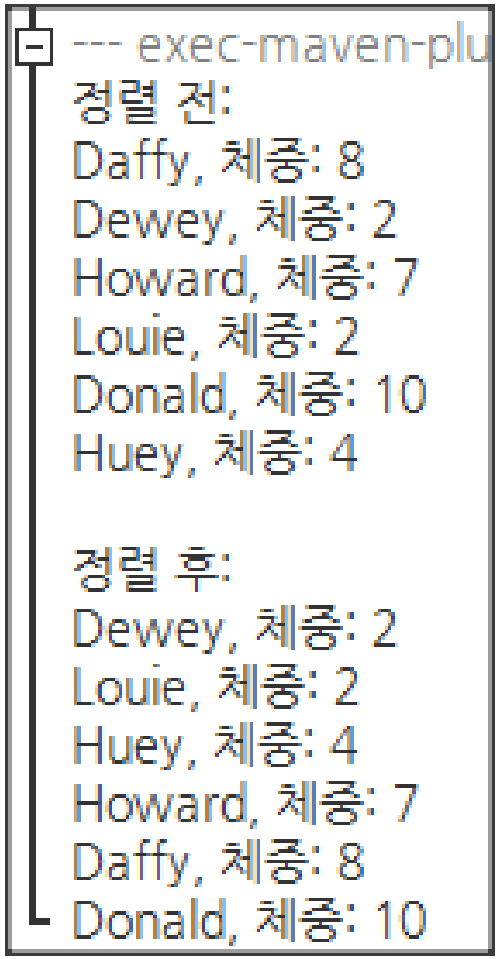
```
6   package cse.design_pattern.ch08.array_sort;
7
8   import java.util.Arrays;
9
10  /**...4 lines */
14  public class DuckSortTestDrive {
15
16      /**...3 lines */
19      public static void main(String[] args) {
20          Duck[] ducks = {
21              new Duck("Daffy", 8),
22              new Duck("Dewey", 2),
23              new Duck("Howard", 7),
24              new Duck("Louie", 2),
25              new Duck("Donald", 10),
26              new Duck("Huey", 4)
27          };
```

```
29          System.out.println("정렬 전:");
30          display(ducks);
31
32          Arrays.sort(ducks);
33
34          System.out.println("₩n정렬 후:");
35          display(ducks);
36
37      }
38
39      public static void display(Duck[] ducks) {
41          for (int i = 0; i < ducks.length; i++) {
42              System.out.println(ducks[i]);
43          }
44      }
45  }
```

(참고) List.sort() 사용법: Duck 클래스에 getWeight() 정의되어 있다고 가정

```
List<Duck> duckList = Arrays.asList(ducks);
duckList.sort((d1, d2) -> d1.getWeight() - d2.getWeight());  // Comparator 인터페이스 구현
```

실행 결과


--- exec-maven-plu
정렬 전:
Daffy, 체중: 8
Dewey, 체중: 2
Howard, 체중: 7
Louie, 체중: 2
Donald, 체중: 10
Huey, 체중: 4
정렬 후:
Dewey, 체중: 2
Louie, 체중: 2
Huey, 체중: 4
Howard, 체중: 7
Daffy, 체중: 8
Donald, 체중: 10