Patrón de diseño “Template”

Preparando bebidas con cafeína...

Todo el mundo sabe preparar un café de sobresito y un te, o al menos los pasos.

Comparando algoritmos

Café:

1) Hervir agua

2) Poner café molido en el agua hirviendo

3) Servir en una taza

4) Agregar azúcar y leche

Té:

1) Hervir agua

2) Poner el té en el agua hirviendo

3) Servir en una taza

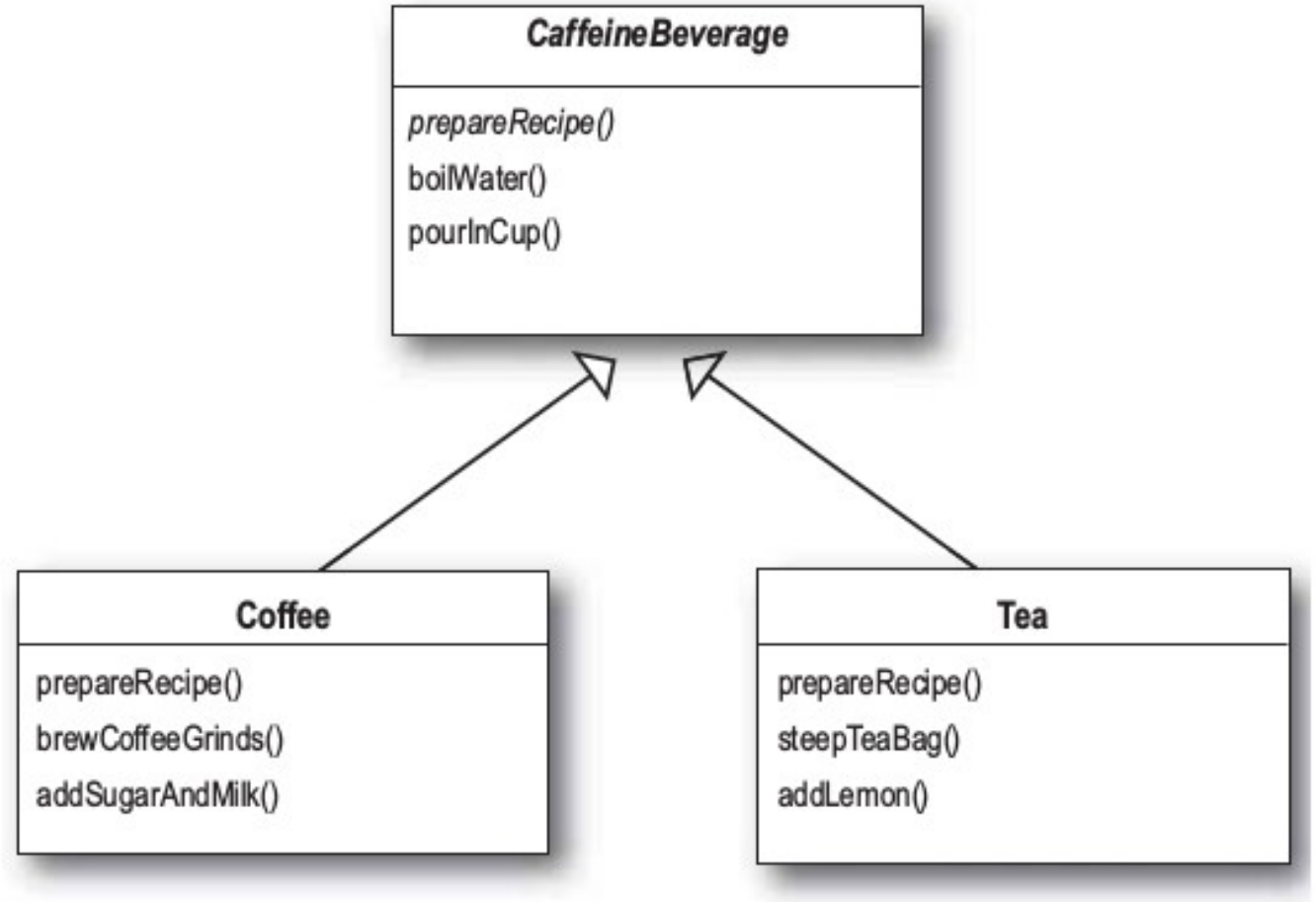
4) Agregar limón

```
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");
    }
}
```

```
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");
    }
}
```

¿Que podemos generalizar?

¿Que podemos generalizar?



¿Eso es todo?

Hemos generalizado las partes que son iguales pero… ¿Solo podemos generalizar eso?

¿Eso es todo?

Hemos generalizado las partes que son iguales pero… ¿Solo podemos generalizar eso?

Los métodos “Poner el café en el agua hirviendo” y “Poner el té en el agua hirviendo”, así como “Agregar azúcar y leche” y “Agregar limón” no son tan diferentes, ¿no podríamos generalizarlos de alguna manera?

Template method pattern

El patrón “Template” o “Template method pattern” define el esqueleto de un algoritmo en un método, difiriendo en algunos pasos en sus subclases. Este deja a sus subclases redefinir ciertas partes de el algoritmo sin cambiar la estructura del algoritmo.

Coffee

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

Tea

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

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

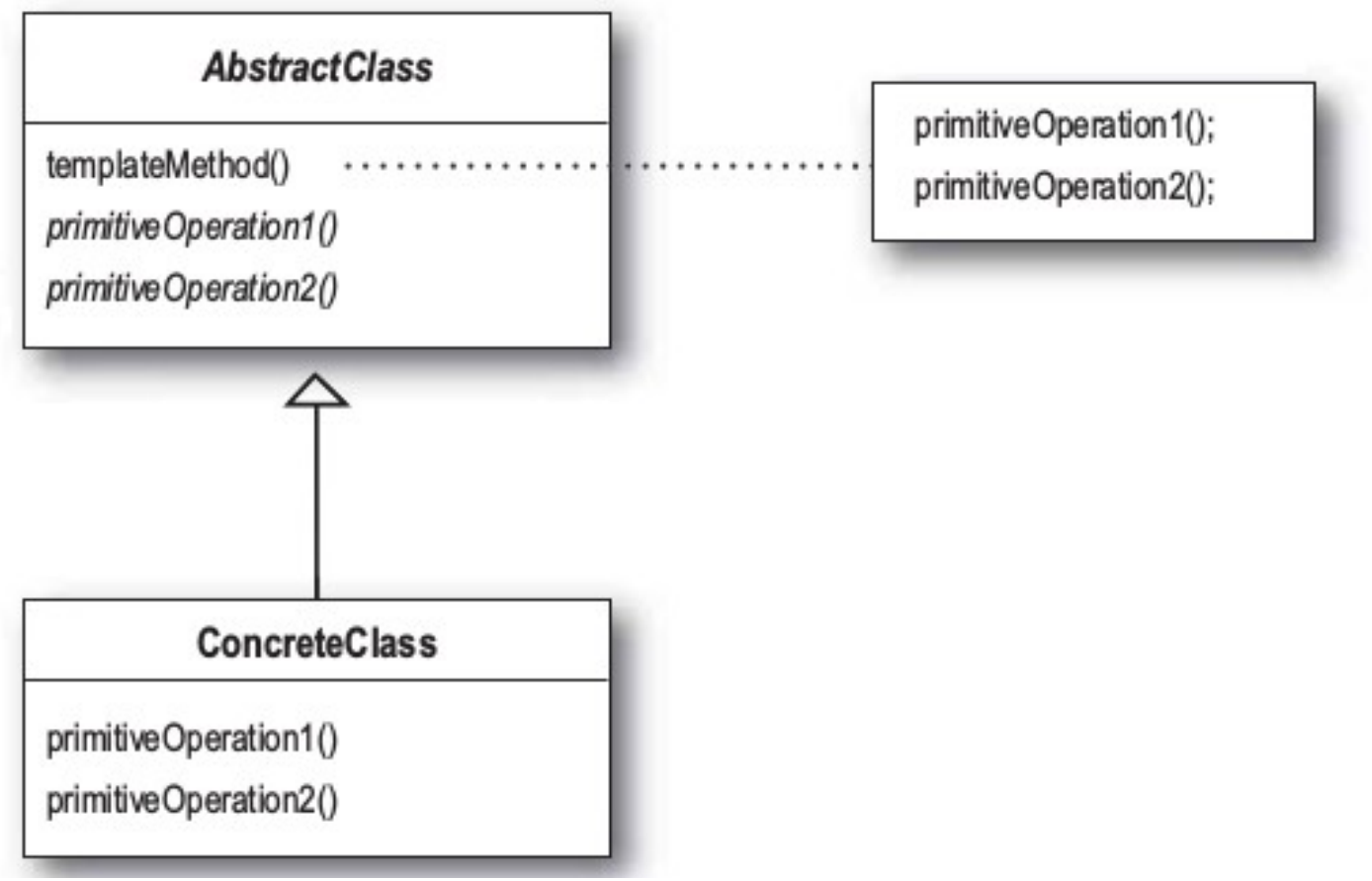
```
public abstract class CaffeineBeverage {

    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");
    }
}
```

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

El Método de Template define los pasos de un algoritmo y permite que las subclases proporcionen la implementación de uno o más pasos.

Diagrama de clases general



Métodos Hooks

Hooks

También podemos tener métodos concretos que no hacen nada por defecto o implementación por defecto; llamamos a estos "hook" (ganchos). Esto da subclases la capacidad de "engancharse" al Algoritmo en varios puntos, si lo desean; un subclase también es libre de ignorar el gancho.

```
abstract class AbstractClass {

    final void templateMethod() {
        primitiveOperation1();
        primitiveOperation2();
        concreteOperation();
        hook();
    }

    abstract void primitiveOperation1();

    abstract void primitiveOperation2();

    final void concreteOperation() {
        // implementation here
    }

    void hook() {}

}
```

Aplicándolo al problema

Supongamos que ahora el café puede o no llevar azucar y leche, el té siempre lleva limón.

¿Como arreglarías la implementación para que funcionara así?

```
public abstract class CaffeineBeverageWithHook {

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

    abstract void brew();

    abstract void addCondiments();

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

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

    boolean customerWantsCondiments() {
        return true;
    }
}
```

```
public class CoffeeWithHook extends CaffeineBeverageWithHook {

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

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

    public boolean customerWantsCondiments() {
        String answer = getUserInput();

        if (answer.toLowerCase().startsWith("y")) {
            return true;
        } else {
            return false;
        }
    }
}
```

Here's where you override the hook and provide your own functionality.

Get the user's input on the condiment decision and return true or false. depending on the input.

```
private String getUserInput() {
    String answer = null;

    System.out.print("Would you like milk and sugar with your coffee (y/n)? ");

    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    try {
        answer = in.readLine();
    } catch (IOException ioe) {
        System.err.println("IO error trying to read your answer");
    }
    if (answer == null) {
        return "no";
    }
    return answer;
}
}
```

This code asks the user if he'd like milk and sugar and gets his input from the command line.

Cuando usar hooks

Los métodos que es necesario que las subclases implementen se utilizan métodos abstractos.

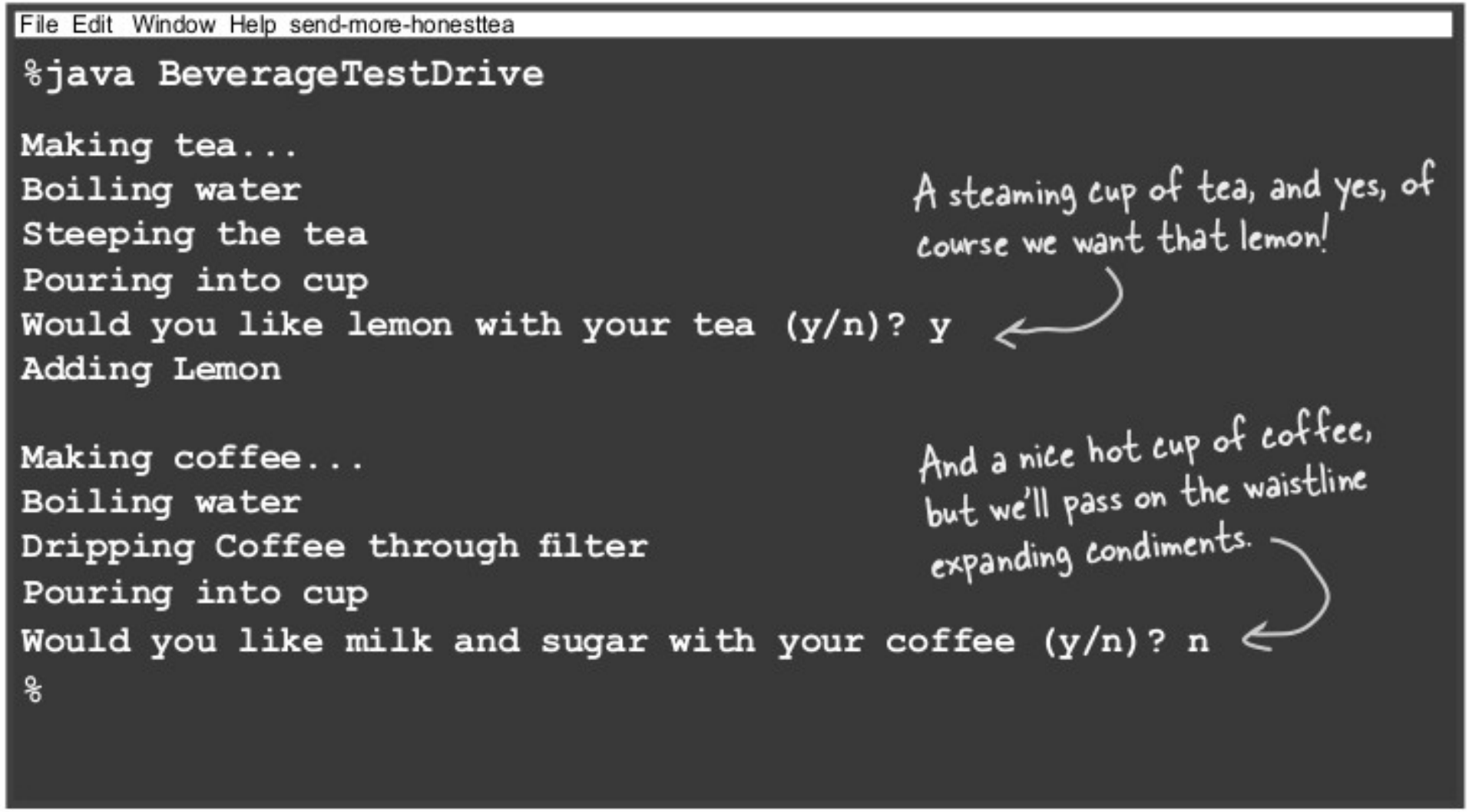
Cuando un método tiene una implementación general y no se requiere que todas las subclases lo implementen de una manera distinta se utiliza un método hook.

Bueno, siempre si quiero la modificación en el té

```
import java.io.*;

public class TeaWithHook extends CaffeineBeverageWithHook {

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

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

    public boolean customerWantsCondiments() {

        String answer = getUserInput();

        if (answer.toLowerCase().startsWith("y")) {
            return true;
        } else {
            return false;
        }
    }
```

```
    private String getUserInput() {
        // get the user's response
        String answer = null;

        System.out.print("Would you like lemon with your tea (y/n)? ");

        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        try {
            answer = in.readLine();
        } catch (IOException ioe) {
            System.err.println("IO error trying to read your answer");
        }
        if (answer == null) {
            return "no";
        }
        return answer;
    }
}
```

```
public class BeverageTestDrive {
    public static void main(String[] args) {

        TeaWithHook teaHook = new TeaWithHook();
        CoffeeWithHook coffeeHook = new CoffeeWithHook();

        System.out.println("\nMaking tea...");
        teaHook.prepareRecipe();

        System.out.println("\nMaking coffee...");
        coffeeHook.prepareRecipe();
    }
}
```


File Edit Window Help send-more-honesttea
%java BeverageTestDrive
Making tea...
Boiling water
Steeping the tea
Pouring into cup
Would you like lemon with your tea (y/n)? y
Adding Lemon
Making coffee...
Boiling water
Dripping Coffee through filter
Pouring into cup
Would you like milk and sugar with your coffee (y/n)? n
%
A steaming cup of tea, and yes, of course we want that lemon!
And a nice hot cup of coffee, but we'll pass on the waistline expanding condiments.