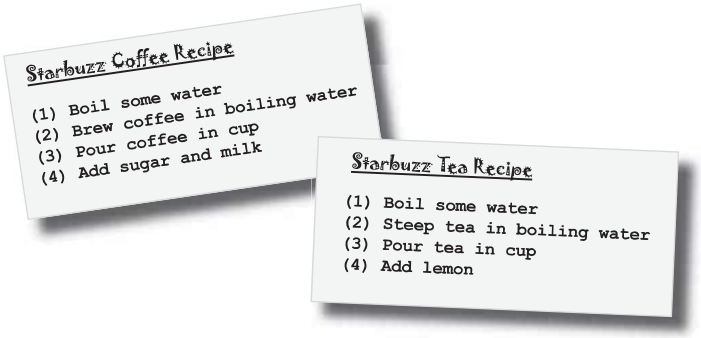
# Taking the design further...

**So what else do Coffee and Tea have in common? Let's start with the recipes.**




**Notice that both recipes follow the same algorithm:**

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.

**So, can we find a way to abstract prepareRecipe() too? Yes, let's find out...**




Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

# Abstracting prepareRecipe()

### Let's step through abstracting prepareRecipe() from each subclass (that is, the Coffee and Tea classes)...

1. The first problem we have is that Coffee uses brewCoffeeGrinds() and addSugarAndMilk() methods while Tea uses steepTeaBag() and addLemon() methods.


Let's think through this: steeping and brewing aren't so different; they're pretty analogous. So let's make a new method name, say, brew(), and we'll use the same name whether we're brewing coffee or steeping tea.

Likewise, adding sugar and milk is pretty much the same as adding a lemon: both are adding condiments to the beverage. Let's also make up a new method name, addCondiments(), to handle this. So, our new prepareRecipe() method will look like this:

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

2. Now we have a new prepareRecipe() method, but we need to fit it into the code. To do this we are going to start with the CaffeineBeverage superclass:




Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

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).

3 Finally we need to deal with the Coffee and Tea classes. They now rely on CaffeineBeverage to handle the recipe, so they just need to handle brewing and condiments:

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

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

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

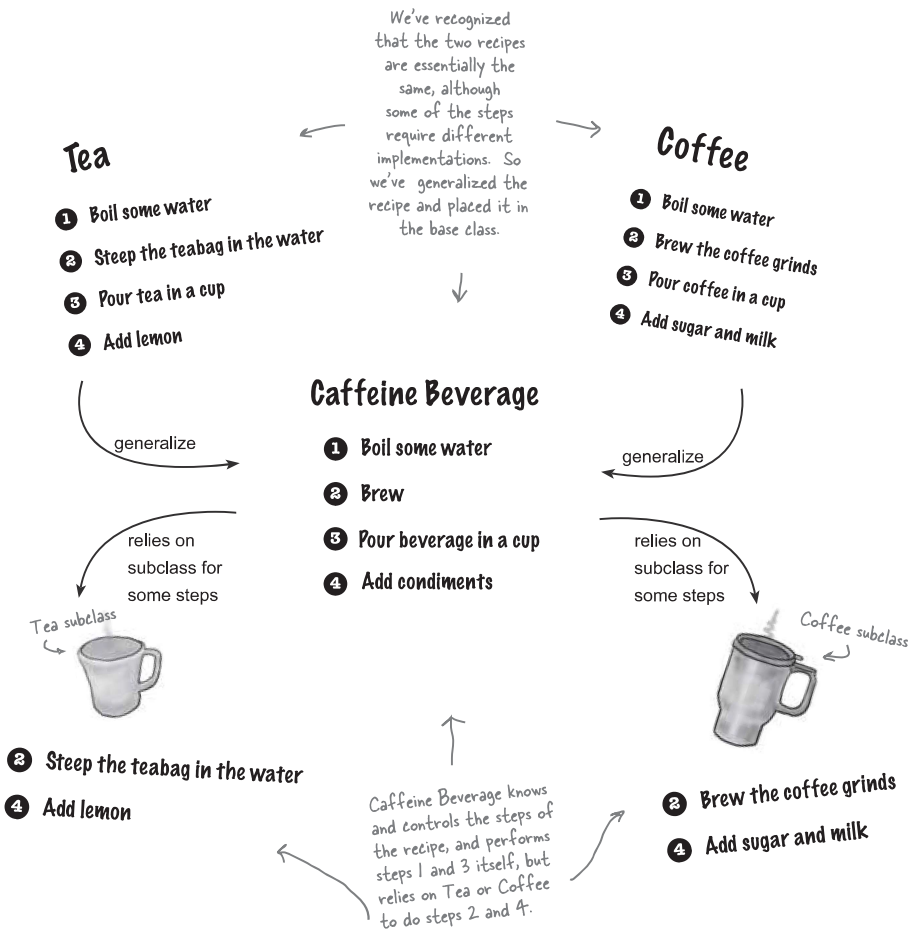
Same for Coffee, except Coffee deals with coffee, and sugar and milk instead of tea bags and lemon.

```
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");
    }
}
```




Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

## What have we done?





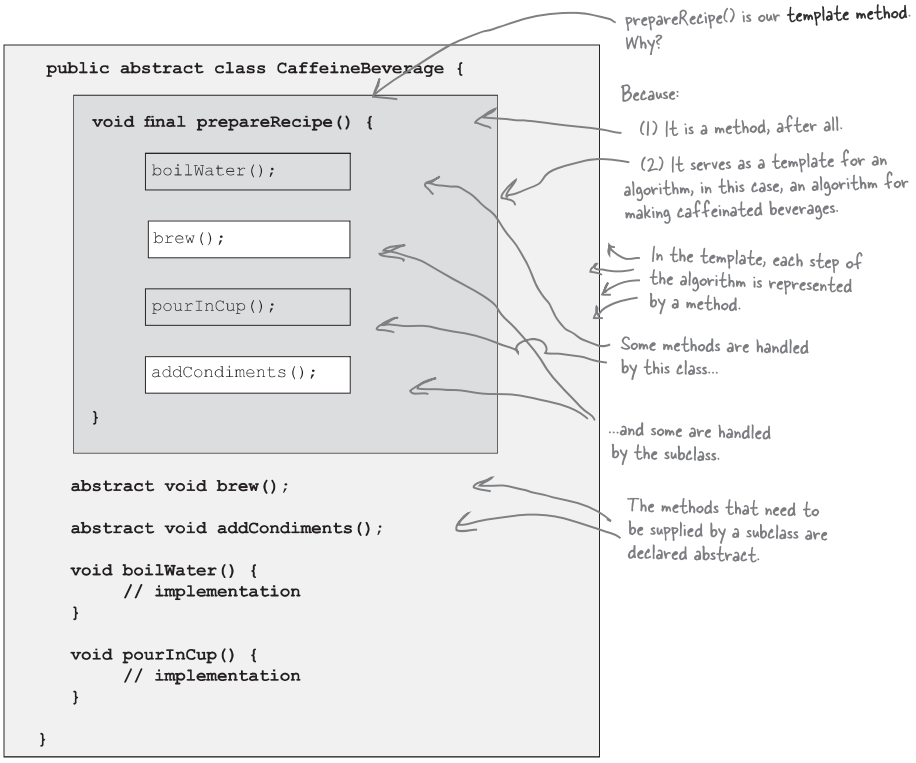


Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

*meet the template method pattern*

## Meet the Template Method

We've basically just implemented the Template Method Pattern. What's that? Let's look at the structure of the CaffeineBeverage class; it contains the actual "template method:"




**The Template Method defines the steps of an algorithm and allows subclasses to provide the implementation for one or more steps.**

# Let's make some tea...


**Let's step through making a tea and trace through how the template method works. You'll see that the template method controls the algorithm; at certain points in the algorithm, it lets the subclass supply the implementation of the steps...**

1. Okay, first we need a Tea object...

```
Tea myTea = new Tea();
```

2. Then we call the template method:

```
myTea.prepareRecipe();
```

which follows the algorithm for making caffeine beverages...

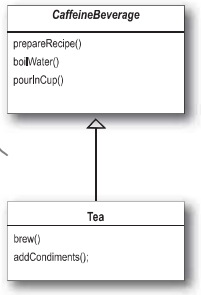
```
boilWater();
brew();
pourInCup();
addCondiments();
```

The prepareRecipe() method controls the algorithm, no one can change this, and it counts on subclasses to provide some or all of the implementation.

3. First we boil water:

```
boilWater();
```

which happens in CaffeineBeverage.




4. Next we need to brew the tea, which only the subclass knows how to do:

```
brew();
```

5. Now we pour the tea in the cup; this is the same for all beverages so it happens in CaffeineBeverage:

```
pourInCup();
```

6. Finally, we add the condiments, which are specific to each beverage, so the subclass implements this:

```
addCondiments();
```




Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

## What did the Template Method get us?

**Underpowered Tea & Coffee implementation**

**New, hip CaffeineBeverage powered by Template Method**

| | |
|---|---|
| Coffee and Tea are running the show; they control the algorithm. | The CaffeineBeverage class runs the show; it has the algorithm, and protects it. |
| Code is duplicated across Coffee and Tea. | The CaffeineBeverage class maximizes reuse among the subclasses. |
| Code changes to the algorithm require opening the subclasses and making multiple changes. | The algorithm lives in one place and code changes only need to be made there. |
| Classes are organized in a structure that requires a lot of work to add a new caffeine beverage. | The Template Method version provides a framework that other caffeine beverages can be plugged into. New caffeine beverages only need to implement a couple of methods. |
| Knowledge of the algorithm and how to implement it is distributed over many classes. | The CaffeineBeverage class concentrates knowledge about the algorithm and relies on subclasses to provide complete implementations. |




Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25