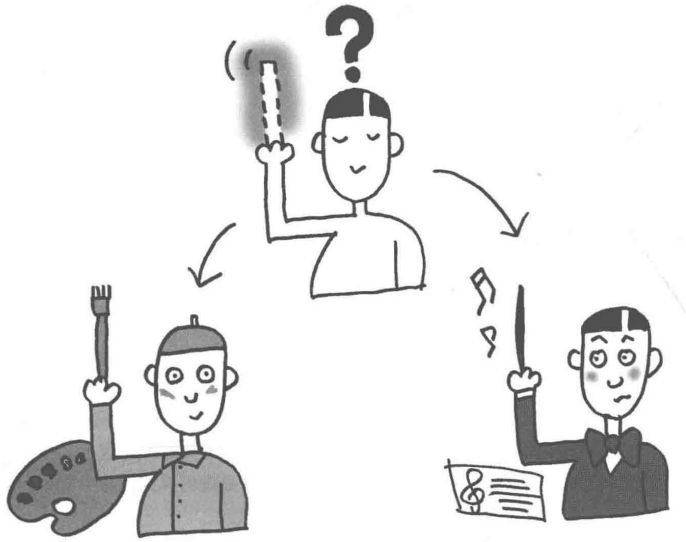
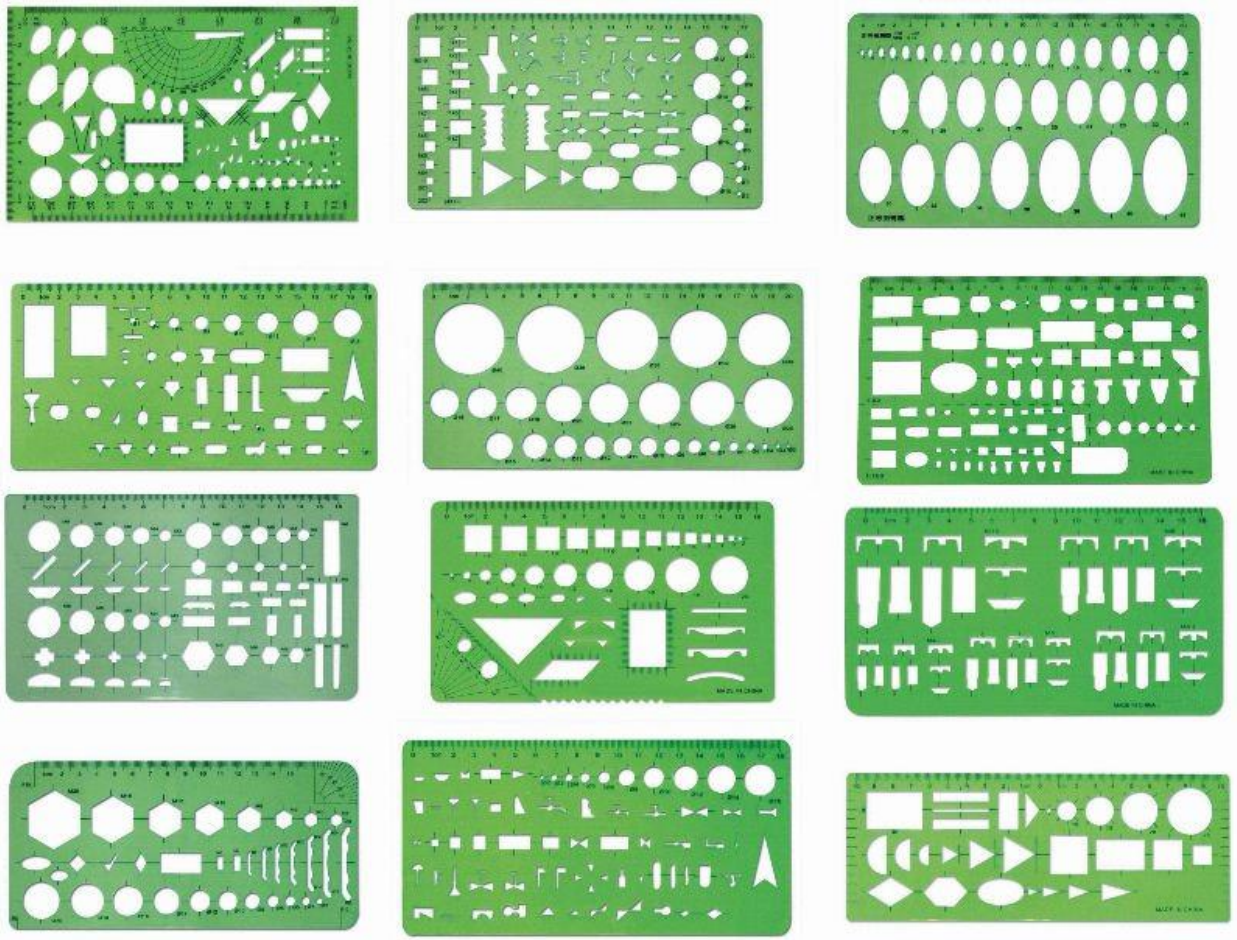
Template Method (模板方法, Behavioral Pattern)



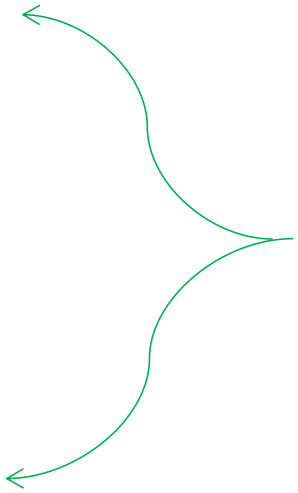
Kai SHI

Template



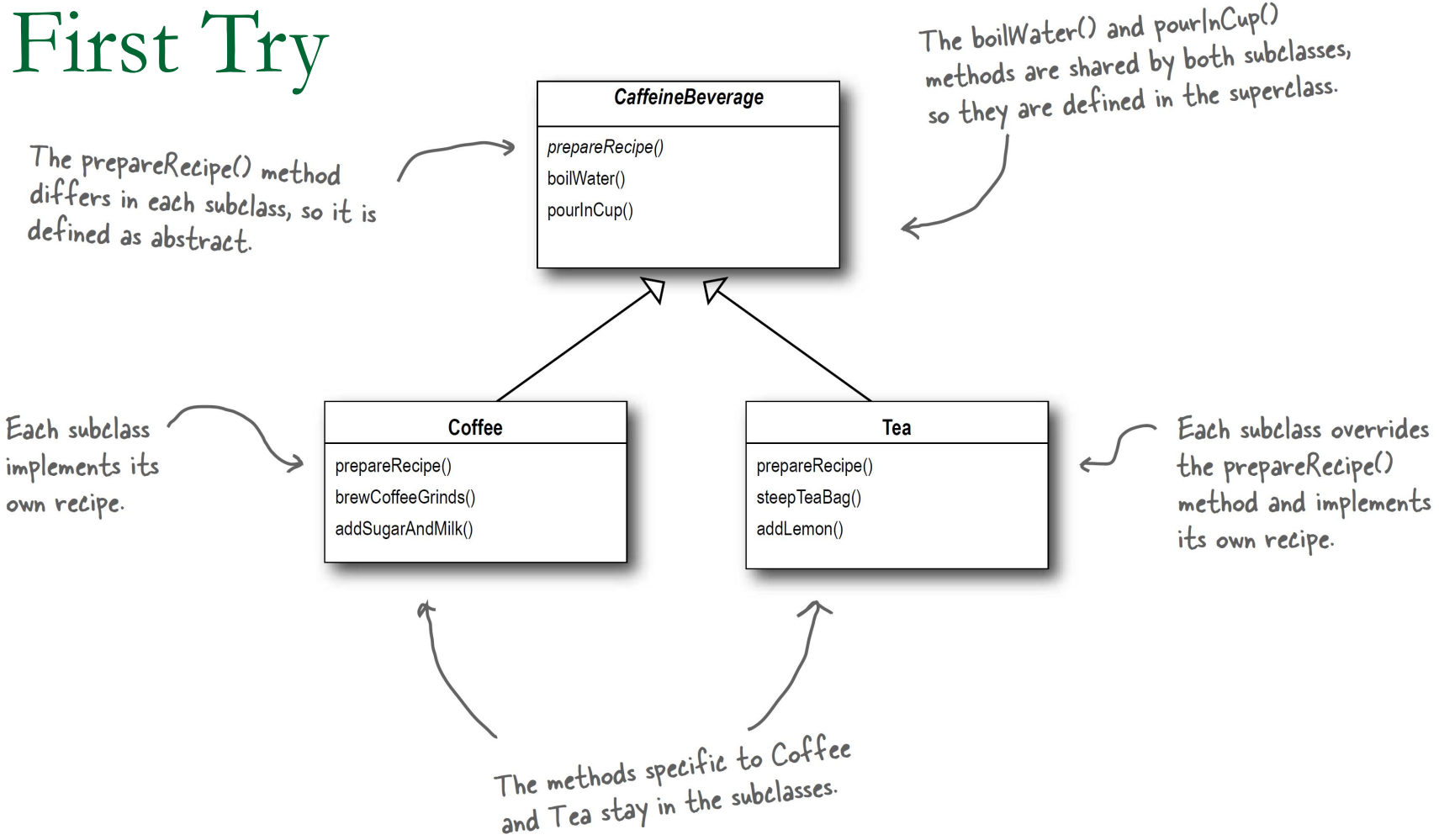
Template Method: Encapsulating Algorithms

- Make coffee
  - (1) Boil some water
  - (2) Brew coffee in boiling water
  - (3) Pour coffee in cup
  - (4) Add sugar and milk
- Make tea
  - (1) Boil some water
  - (2) Steep tea in boiling water
  - (3) Pour tea in cup
  - (4) Add lemon
- Draw Class Diagram Now



The procedures (i.e., the algorithms to make beverage) are similar.

First Try

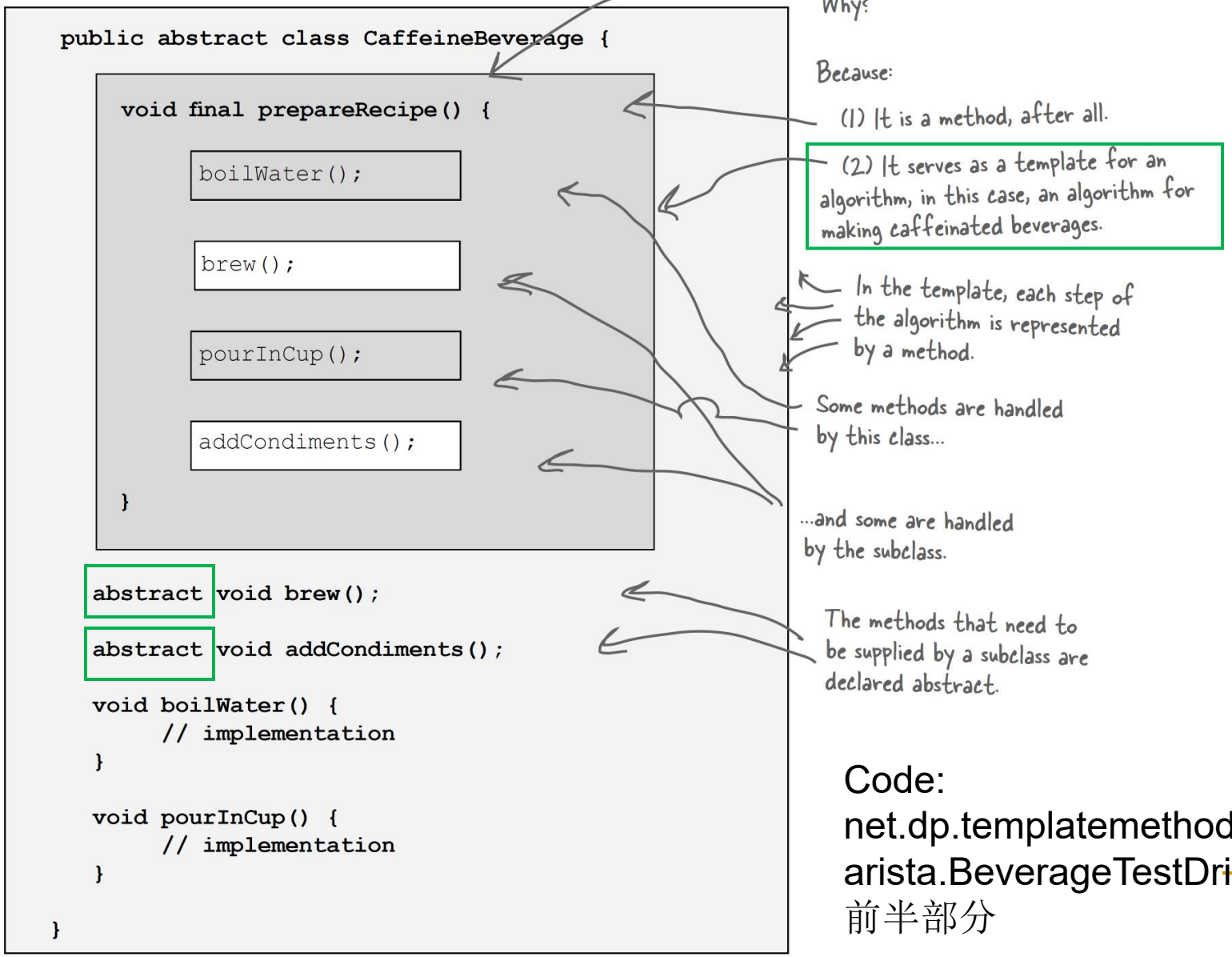


- It looks OK. But, how to make sure the procedure is correct?

- Draw Class Diagram Now


We've recognized that the two recipes are essentially the same, although some of the steps require different implementations. So we've generalized the recipe and placed it in the base class.
Tea
1 Boil some water
2 Steep the teabag in the water
3 Pour tea in a cup
4 Add lemon
Coffee
1 Boil some water
2 Brew the coffee grinds
3 Pour coffee in a cup
4 Add sugar and milk
Caffeine Beverage
1 Boil some water
2 Brew (冲泡)
3 Pour beverage in a cup
4 Add condiments
generalize
generalize
relies on subclass for some steps
relies on subclass for some steps
Tea subclass
Coffee subclass
2 Steep the teabag in the water
4 Add lemon
2 Brew the coffee grinds
4 Add sugar and milk
Caffeine Beverage knows and controls the steps of the recipe, and performs steps 1 and 3 itself, but relies on Tea or Coffee to do steps 2 and 4.

Template Method



Code:
net.dp.templatemethod.barista.BeverageTestDrive
前半部分

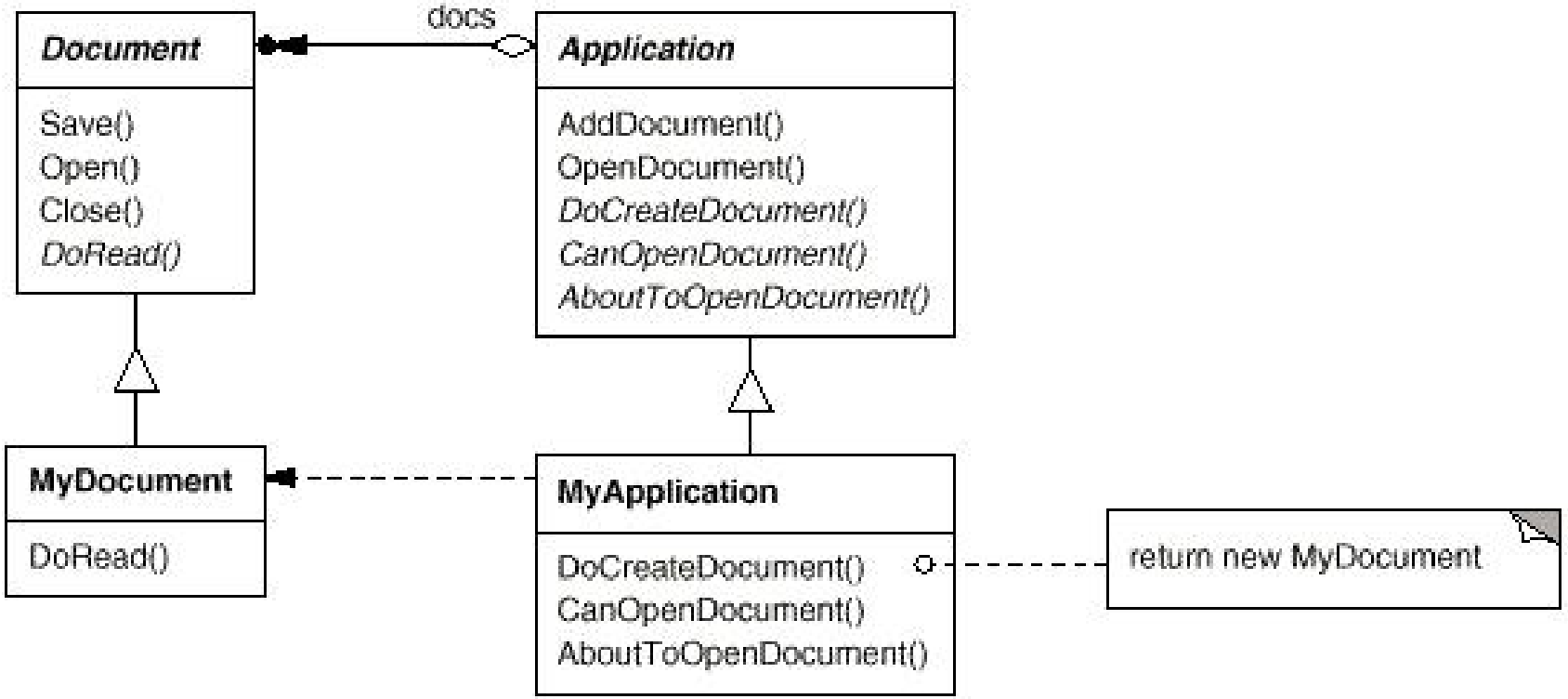
Template Method

- Intent
  - Define the skeleton of an algorithm in an operation, deferring some steps to **lets subclasses redefine** certain steps of an algorithm without changing the algorithm's structure.
  - (准备一个抽象类，将**部分逻辑**以具体方法以及具体构造子的形式实现，然后声明一些抽象方法来迫使子类实现**剩余**的逻辑。不同的子类可以以不同的方式实现这些抽象方法，从而对剩余的逻辑有不同的实现。)

Motivation (1/2)

- Consider an application framework that provides Application (responsible for opening existing document) and Document classes (represents the loaded document).
- Applications built with the framework can subclass Application and Document to suit specific needs. E.g., DrawDocument, SpreadsheetDocument

Motivation (2/2)



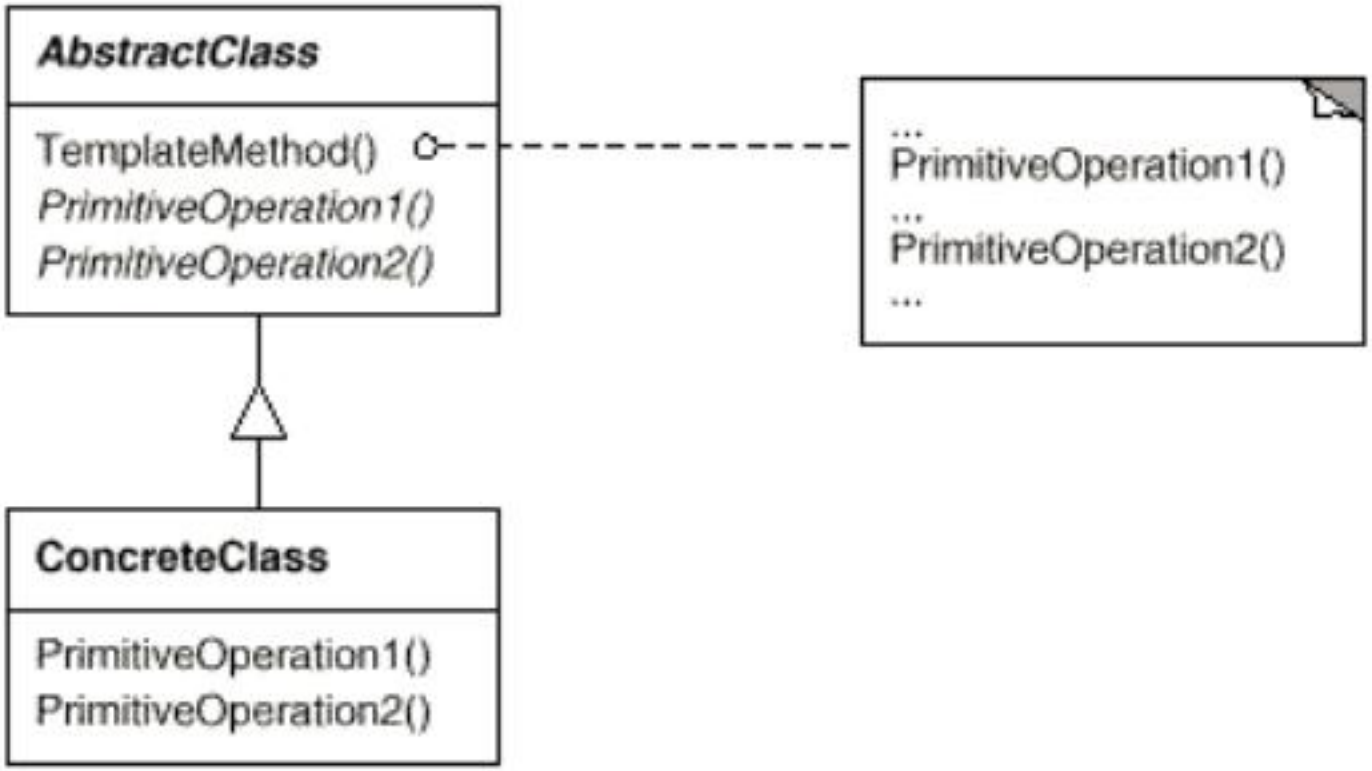
Applicability: The Template Method pattern should be used

- To implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary.
- Refactoring to generalize.
  - When common behavior among subclasses should be factored and localized in a common class to avoid code duplication.
- Control subclasses extensions.
  - You can define a template method that calls "hook" operations at specific points, thereby permitting extensions only at those points.

Design Principle (mentioned in Strategy)

- Separating what changes from what stays the same

Structure


AbstractClass
TemplateMethod()
PrimitiveOperation1()
PrimitiveOperation2()
...
PrimitiveOperation1()
...
PrimitiveOperation2()
...
ConcreteClass
PrimitiveOperation1()
PrimitiveOperation2()

Here we have our abstract class; it is declared abstract and meant to be subclassed by classes that provide implementations of the operations.

Here's the template method. It's declared final to prevent subclasses from reworking the sequence of steps in the algorithm.

```
abstract class AbstractClass {

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

    abstract void primitiveOperation1();

    abstract void primitiveOperation2();

    void concreteOperation() {
        // implementation here
    }
}
```

The template method defines the sequence of steps, each represented by a method.

In this example, two of the primitive operations must be implemented by concrete subclasses.

We also have a concrete operation defined in the abstract class. More about these kinds of methods in a bit...

Participants

- AbstractClass
  - defines abstract primitive operations that concrete subclasses define to implement steps of an algorithm.
  - implements a template method defining the skeleton of an algorithm. The template method calls primitive operations as well as operations defined in AbstractClass or those of other objects.
- ConcreteClass
  - implements the primitive operations to carry out subclass-specific steps of the algorithm.

Consequences

- Template methods are a fundamental technique for code reuse.
- Template methods are particularly important in class libraries, because they are the means for factoring out (分解) common behavior in library classes.

“Hook”: Control subclasses extensions

- You can define a template method that calls "hook" operations at specific points, thereby permitting extensions only at those points.

```
public abstract class CaffeineBeverageWithHook {

    final 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;
    }
}
```

We've added a little conditional statement that bases its success on a concrete method, customerWantsCondiments(). If the customer WANTS condiments, only then do we call addCondiments().

Here we've defined a method with a (mostly) empty default implementation. This method just returns true and does nothing else.

This is a **hook** because the subclass can override this method, but doesn't have to.

```
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;
        }
    }

    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;
    }
}
```

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.

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

Test

```
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();
    }
}
```

Create a tea.

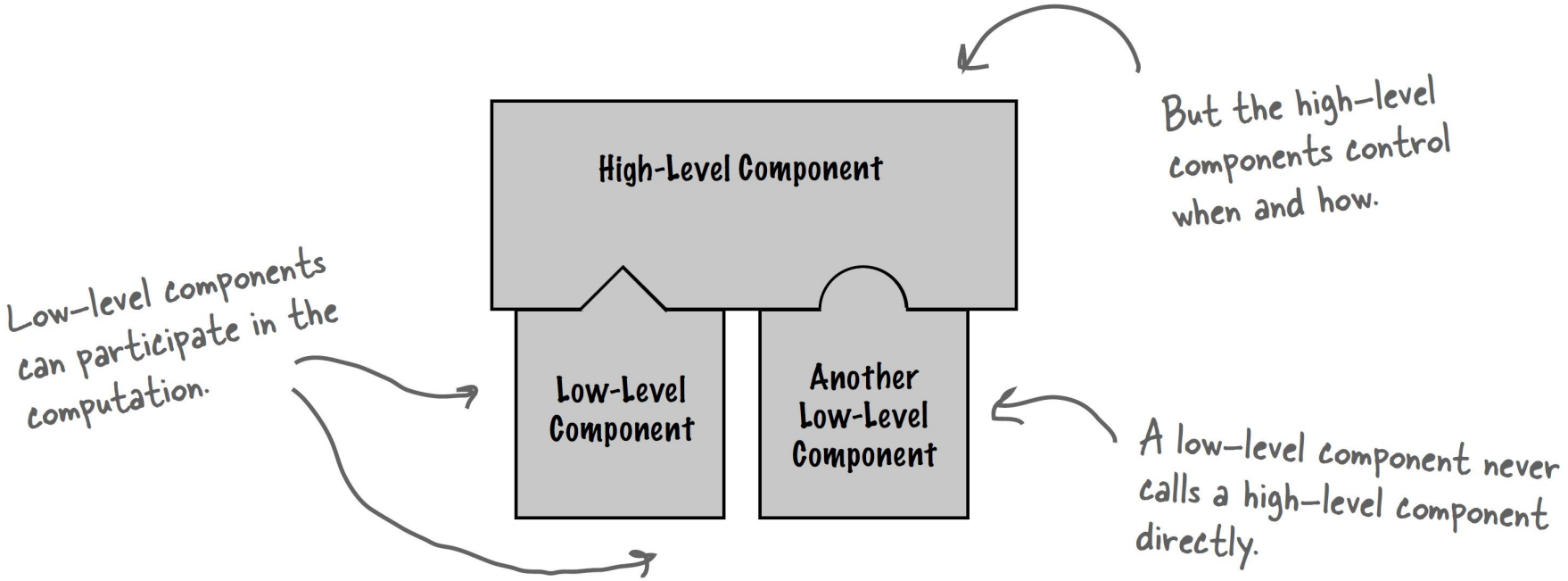
A coffee.

And call prepareRecipe() on both!

Code: net.dp.templatemethod.barista.BeverageTestDrive
后半部分

Dependence Inversion Principle (Hollywood Principle)

- Don't call us, we'll call you.



Implementation 1: Naming conventions

- Identify the operations that should be overridden by adding a prefix to their names.
- E.g., the prefixes template method names with "do-": "doCreateDocument()", "doRead()".

Implementation 2: Using access control

- In Java
  - The primitive operations can be declared as protected and abstract method;
  - The template method can be declared as final method.

Implementation 3: Minimizing primitive operations

- An important goal in designing template methods is to minimize the number of primitive operations that a subclass must override to flesh out the algorithm.
- The more operations that need overriding, the more tedious things get for clients.

Example in Practice: compareTo()

Comparable: compareTo()

- In order to sort objects in a container in Java, you must make the sorted objects implements interface Comparable, and override compareTo().
- It is not a standard template method, but its idea belongs to template method.

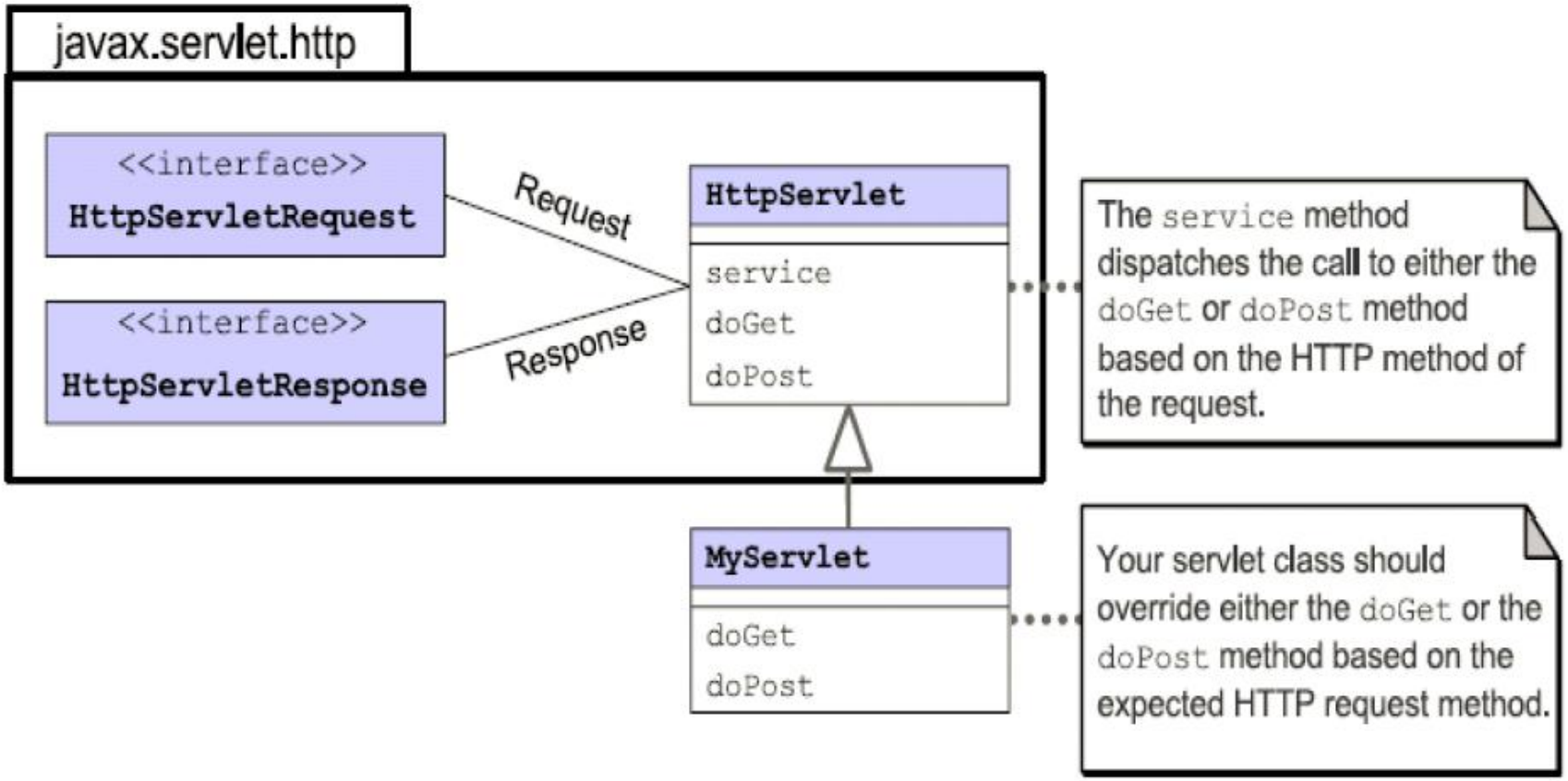
- Code: templatemethod.compare.Test

Example in Practice: HttpServlet in JavaEE

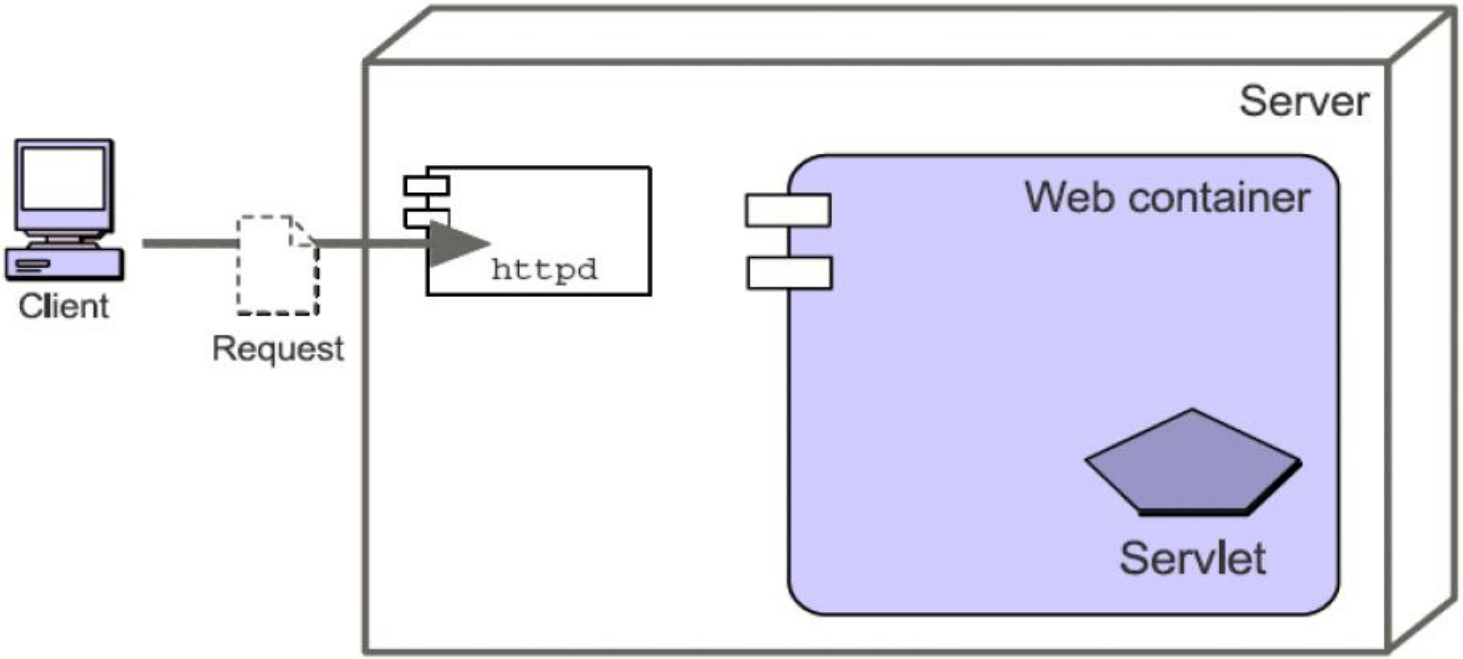
HttpServlet (1/2)

- HttpServlet use Template Method Pattern a lot;
- The subclass of HttpServlet is used to process the http request in different according to it request method (type).

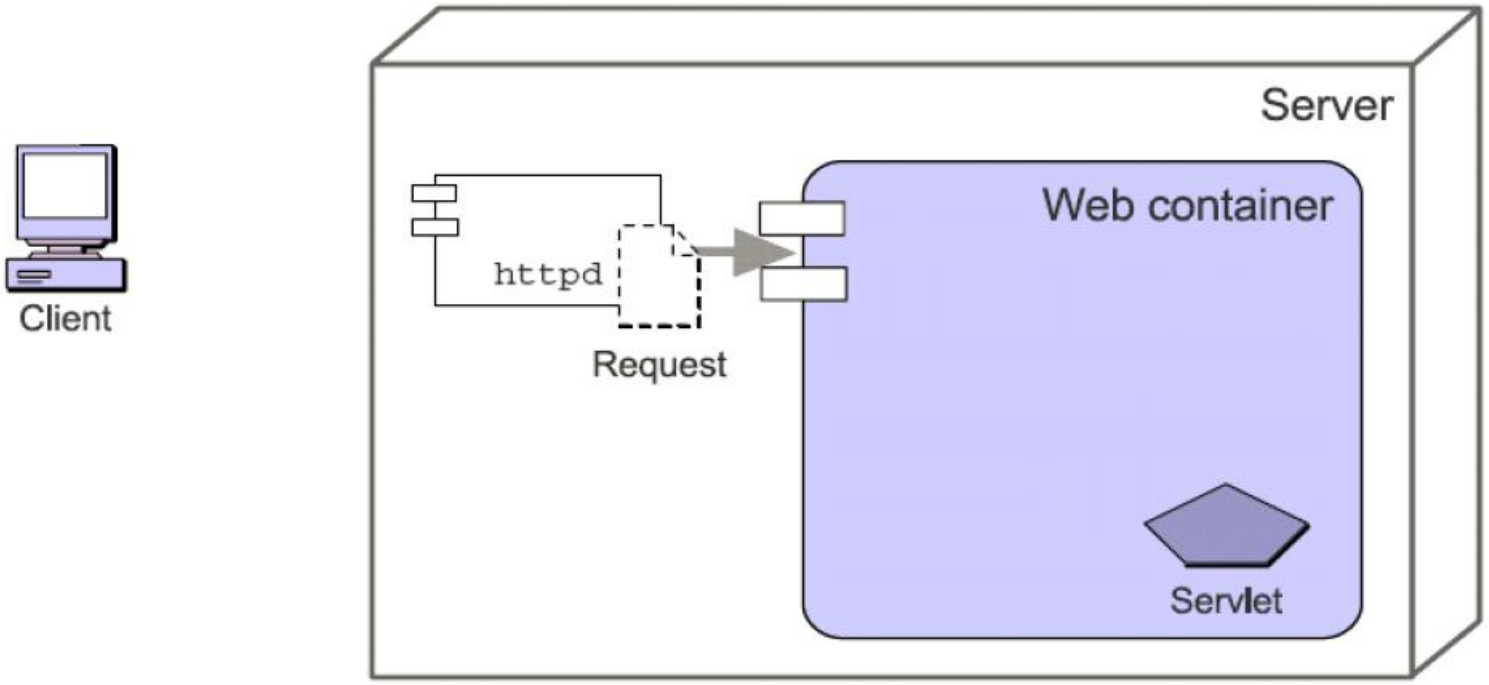
HttpServlet (2/2)



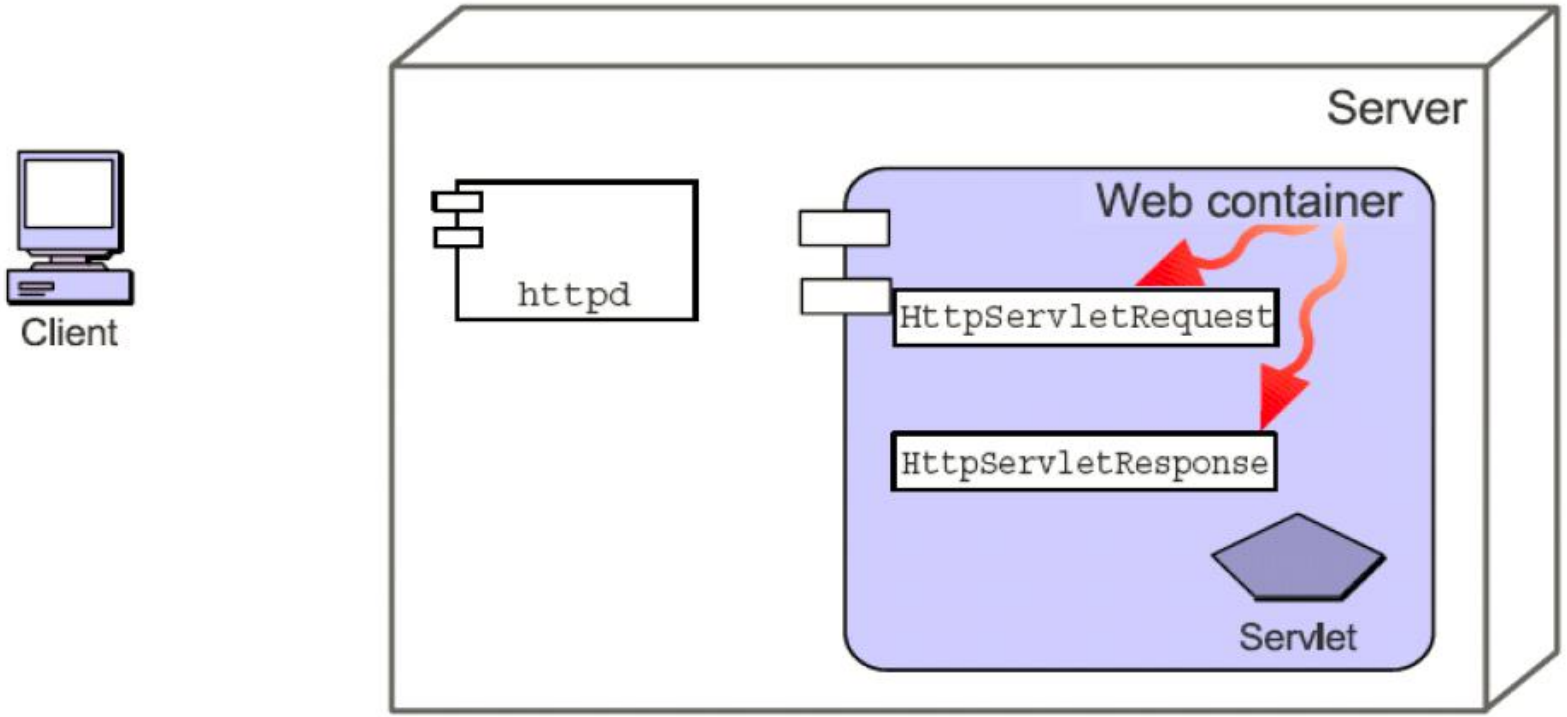
Step 1: Client send the request



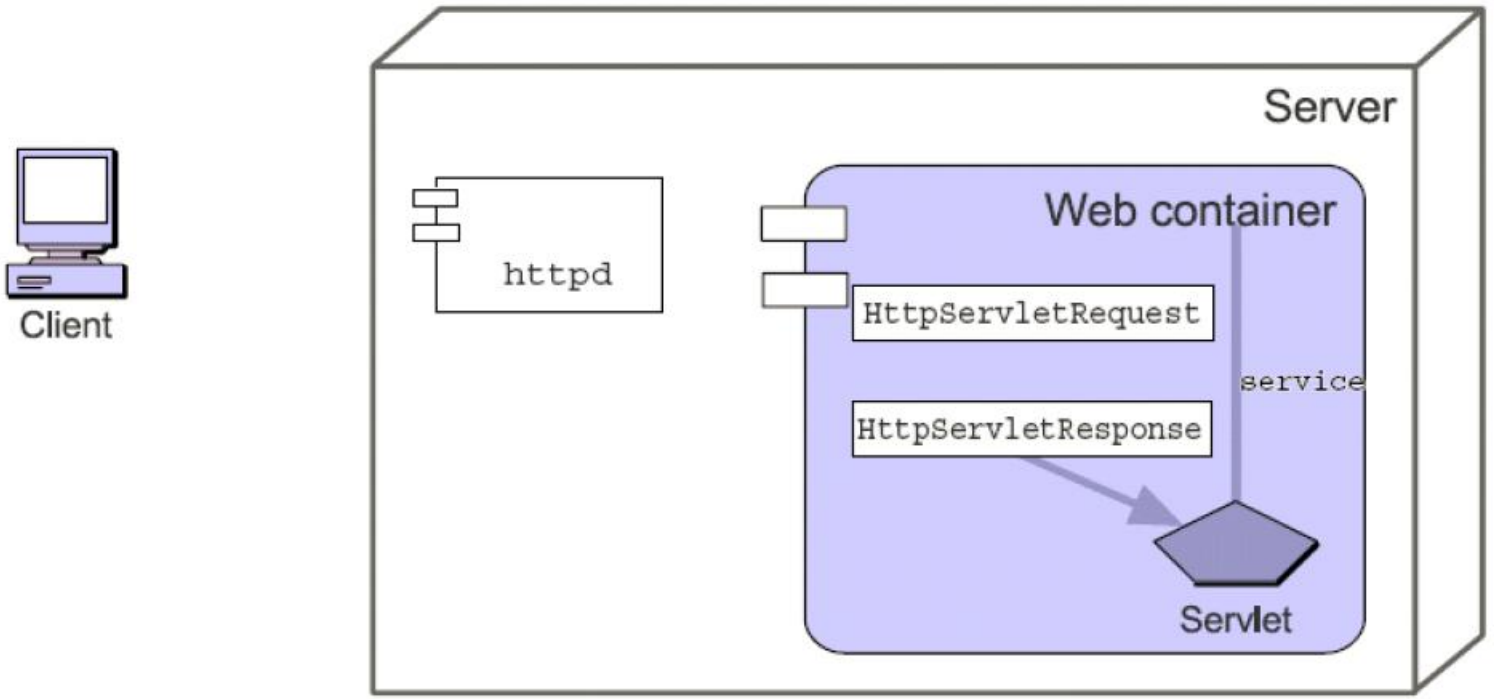
Step 2: Web server send the request to the Web container



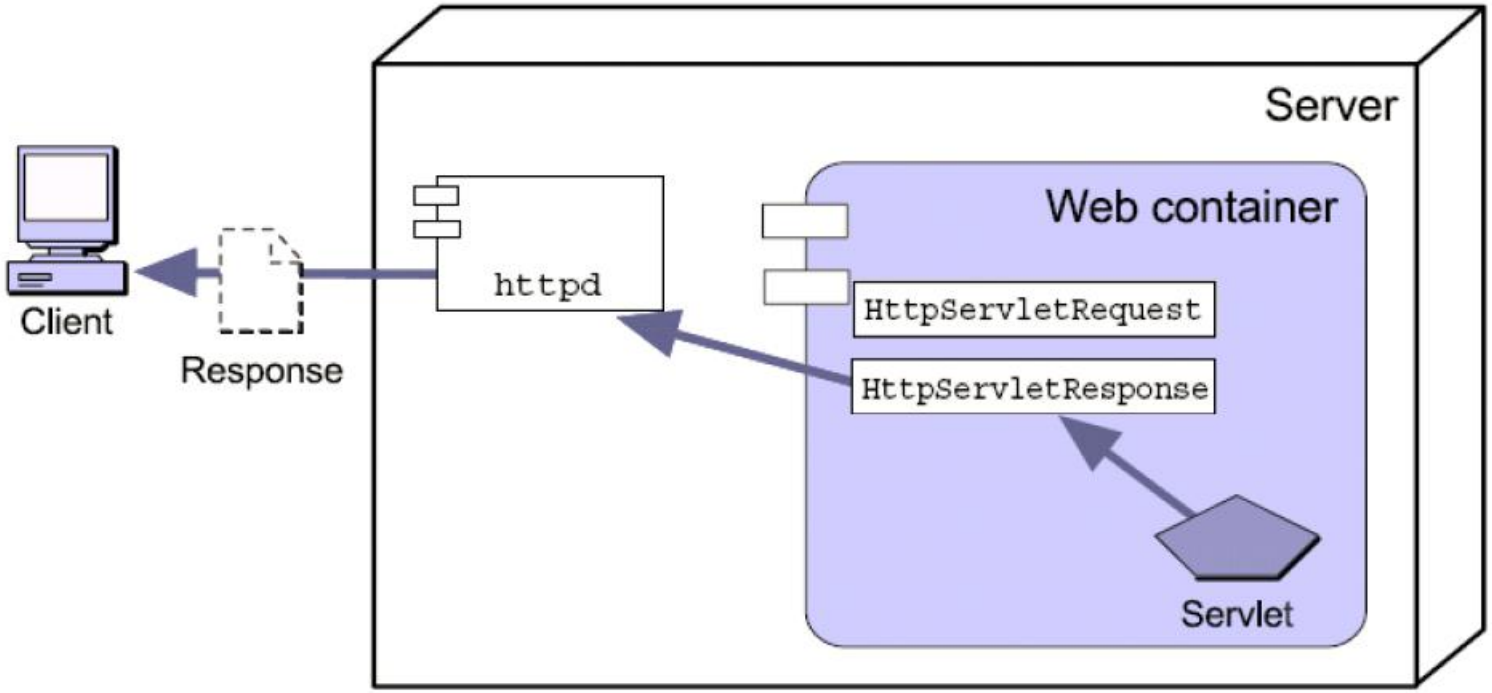
Step 3: Web container initializes the request and response object



Step 4: Web container invokes the Servlet according to the request URL



Step 5: Web container return the response, which is modified by the target Servlet, to the client



doXxx method in HttpServlet

- There are many types of request.
- In service method, the types of request is determined, and corresponding doXxx method is invoked
  - doGet
  - doPost
  - doPut
  - doDelete
  - doOptions
  - doTrace