Theory: Template method

11 minutes 0 / 4 problems solved

§1. Design problem

Template Method is a behavioral pattern that describes the common algorithm while subclasses implement steps of this algorithm. This pattern lets the subclasses implement the steps of the algorithm without changing that algorithm's skeleton.

As an example of the **Template Method**, we will consider the working process. The algorithm contains three steps: go to work, work, go home. It's a very true-to-life example indeed.

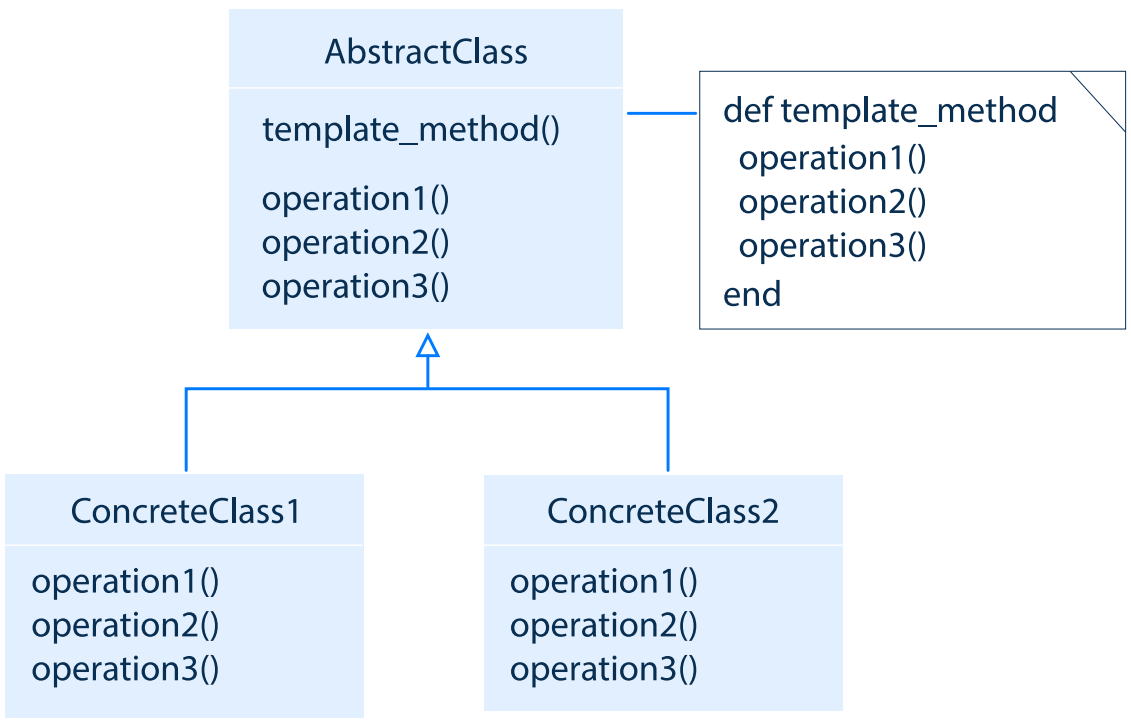
Suppose all workers go to work and go home in an absolutely identical manner. The working routine, however, is different and depends on the worker's qualification. We can choose any profession, but the algorithm remains the same.

§2. Template method pattern

An abstract base class implements standard algorithm steps and can provide a default implementation for custom steps. Specific subclasses provide concrete implementation for each of these steps.

Template Method has the following components:

- **Abstract Class** describes primitive operations and the template method itself which calls primitive operations;
- **Concrete Class** implements the primitive operations.



This pattern is actively used in JDK.

§3. Practice example

To demonstrate how the Template Method works, let's create an abstract class `Worker` that describes the working routine. Then, let's add the template method:

Current topic:

Template method

Topic depends on:

✓ The concept of patterns

✗ Abstract class

Table of contents:

```
public abstract class Worker {

    public void work() {
        goToWork();

        workingProcess();

        goHome();
    }



    public void goToWork() {

        System.out.println("= I'm going to work sadly =");

    }



    public void goHome() {

        System.out.println("= I'm going home happy =");

    }



    public abstract void workingProcess();

}
```

The common algorithm of actions is already determined. Now, we will create two concrete classes: `Programmer` and `Actor`:

```

public class Programmer extends Worker {

    @Override
    public void workingProcess() {
        System.out.println(" > I'm a programmer");
        System.out.println(" > I drink coffee");
        System.out.println(" > I write code");
        System.out.println(" > I drink coffee again");

        System.out.println(" > I write code again");

    }

}



public class Actor extends Worker {



    @Override

    public void workingProcess() {

        System.out.println(" > I'm an actor");

        System.out.println(" > I read a scenario");

        System.out.println(" > I get used to role ");

        System.out.println(" > I play a role");

   }

}
```

In the `Demo` class we create programmer and actor instances and call the template method :

```
public class TemplateMethodDemo {
    public static void main(String[] args) {
        Worker programmer = new Programmer();
        Worker actor = new Actor();
        programmer.work();
        actor.work();
    }
}
```

§4. Conclusion

Template Method is applicable in the following cases:

- When the behavior of an algorithm can vary, you let subclasses implement the behavior through overriding;
- When you want to avoid code duplication, implementing variations of the algorithm in subclasses;
- When you want to be sure that subclassing is allowed.

Report a typo

263 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (9) Hints (0) Useful links (1) Show discussion