Design patterns. Behavioural software design pattern

Template method pattern

1. Design pattern description

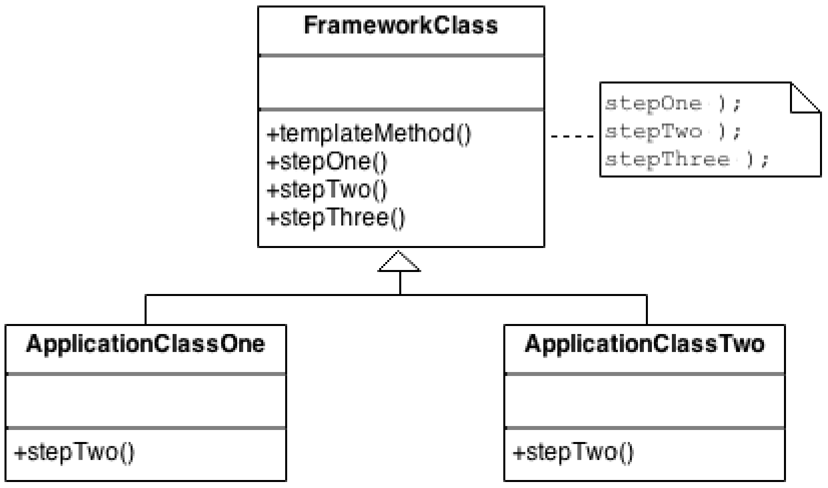
The aim of the Template Method is to define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Problems that the Template method pattern solves

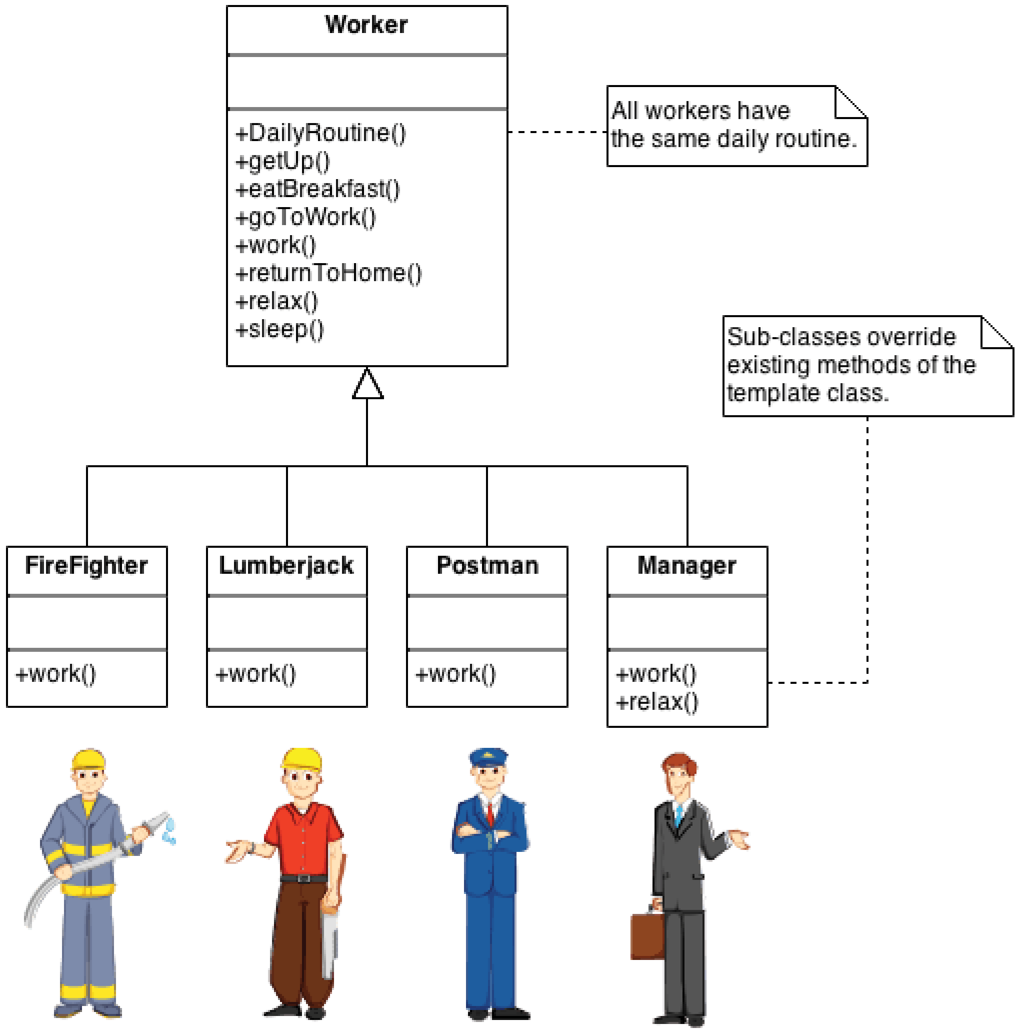
- How can the invariant parts of a behaviour be implemented once so that subclasses can implement the variant parts?
- How can subclasses redefine certain parts of a behaviour without changing the behaviour's structure?

What solution does the Template method design pattern describe?

- The key idea in this pattern is to control subclassing. Subclasses do no longer control how the behaviour of a parent class is redefined. Instead, **a parent class controls how subclasses redefine it**. This is also referred to as inversion of control. "This refers to how a parent class calls the operations of a subclass and not the other way around." [GoF, p327].
- Inversion of control is a common feature of frameworks. When using a library (reusable classes), we call the code we want to reuse. When using a framework (reusable application), we write subclasses and implement the variant code the framework calls.
- Template methods are a fundamental technique for code reuse to implement the common (invariant) parts of a behaviour and to eliminate code duplication, by factoring out invariant behaviour among classes and localizing (generalizing) it in a common class.



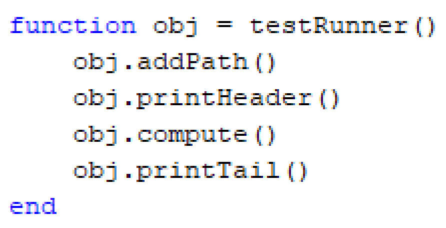
2. Design pattern example



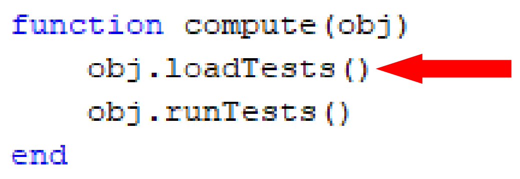
```
worker1 = Postman();
worker1.doDailyRoutine();

worker2 = Manager();
worker2.doDailyRoutine();


classdef Worker < handle
    properties
    end
    methods
        function doDailyRoutine(obj)
            fprintf("#########\n")
            obj.getUp();
            obj.eatBreakfast();
            obj.goToWork();
            obj.work();
            obj.returnToHome();
            obj.relax();
            obj.sleep();
            fprintf("#########\n\n")
        end
    end
    methods (Static)
        function getUp()
            fprintf("Getting up \n")
        end
        function eatBreakfast()
            fprintf("Eating breakfast \n")
        end
        function goToWork()
            fprintf("Going to work \n")
        end
        function returnToHome()
            fprintf("Returning to home \n")
        end
        function relax()
            fprintf("Relaxing 1 hour\n")
        end
        function sleep()
            fprintf("Sleeping\n")
        end
    end
    methods (Abstract,Static)
        work()
    end
end


classdef Postman < Worker
    properties
    end
    methods (Static)
        function work()
            fprintf("<strong>Delivering letters </strong> \n")
        end
    end
end


classdef Manager < Worker
    properties
    end
    methods (Static)
        function work()
            fprintf("<strong>Managing  </strong> \n")
        end
        function relax()
            fprintf("<strong>Relaxing for 2 hours  </strong> \n")
        end
    end
end
```

3. Existing pattern in SWAN?

Yes!

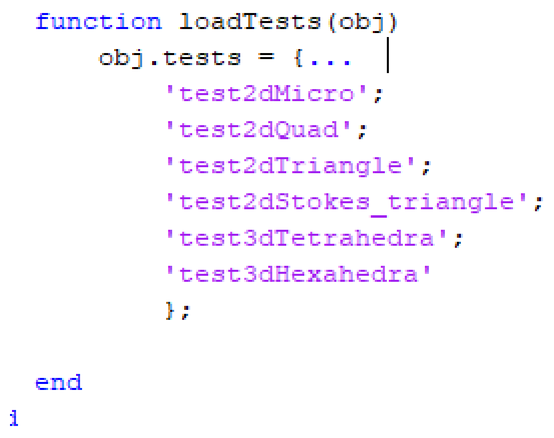


```
function obj = testRunner()
    obj.addPath()
    obj.printHeader()
    obj.compute()
    obj.printTail()
end
```



```
function compute(obj)
    obj.loadTests()
    obj.runTests()
end
```

FemTests:



```
 function loadTests(obj)
     obj.tests = {...
         'test2dMicro';
         'test2dQuad';
         'test2dTriangle';
         'test2dStokes_triangle';
         'test3dTetrahedra';
         'test3dHexahedra'
         };

 end
i
```