The Template Method

CS342 Fall 2016

Defend against Change

- Our Design Patterns will always follow these rules:
  - Separate those elements that change from those that stay the same
  - Program to an Interface not and Implementation
  - Favor Composition over Inheritance
    - except when they don’t...

Suppose....

- Suppose we are writing a program to make a hot beverage.
- We need to write a class for the beverage, and the steps involved in making the beverage.

Making Coffee and Tea

```
class Coffee
    def prepareRecipe()
        boilWater()
        brewCoffeeGrinds()
        pourInCup()
        addSugarAndMilk()
    end
    #…
end
```

```
class Tea
    def prepareRecipe()
        boilWater()
        steepTeaBag()
        pourInCup()
        addLemon()
    end
    #…
end
```

Abstract the commonality (1)

- Is there code duplication in the two classes?
  - Which two methods are the same?
- Need to abstract the commonality into a base class

Classwork: Hot Beverage

Abstract the commonality (2)

```
CaffeineBeverage: prepareRecipe()
    def prepare()
        raise NoMethodError
    end
    def boilWater() #shared
    end
    def pourInCup() #shared
    end
```

```
Coffee
    prepare()
    brewCoffee()
    addSugar()
```

```
Tea
    prepare()
    steepTeaBags()
    addLemon()
```

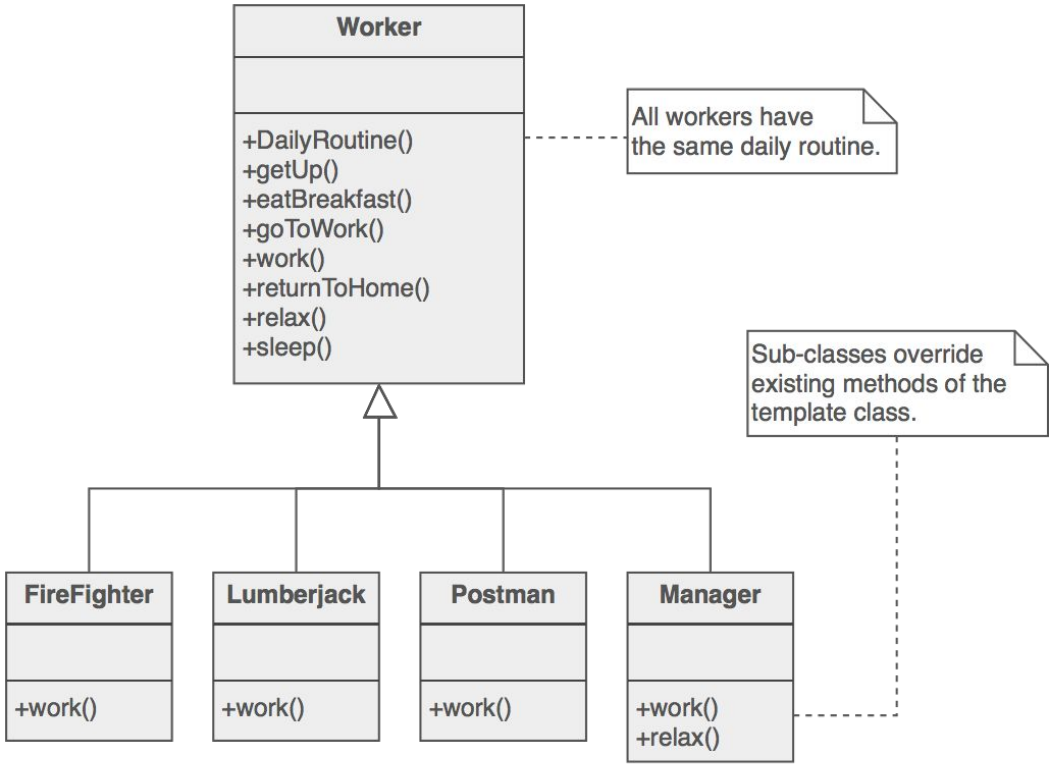
Abstract the commonality (3)

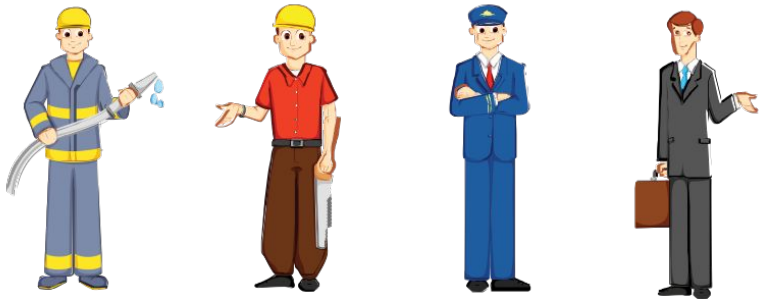
- Coffee
  - brewCoffee()
  - addSugar()
- Tea
  - steepTeaBags()
  - addLemon()

- Is there still code duplication? Where?
  - prepare() method
- We could abstract further, putting prepare in a Beverage class
  - How?

Template Method

- Defines the steps of an algorithm
  - allows subclasses to provide the implementation for one or more steps





CaffeineBeverage

- Should the entire CaffeineBeverage class be abstract?
  - how would you make an abstract class in Ruby?
    - make the constructor, initialize(), raise an exception, which forces it to be overridden
- What methods of the Beverage class should be abstract?
  - How do you decide?
    - Any methods that can’t be generalized at all should be abstract

CaffeineBeverage class

```
class CaffeineBeverage
    def prepareRecipe()
        boilWater()
        brew()
        pourCup()
        addCondiments()
    end
    def brew()
        raise NoMethodError
    end
    def addCondiments()
        raise NoMethodError
    end
    ###
end
```

Tea class

```
class Tea < CaffeineBeverage
     def brew()
        ...
     end
     def addCondiments()
        ...
     end
end
```

Template Method Pattern: usage

- When you have a pattern of steps that produces a different (but similar) output, convert similar operations to a template.
  - Convert from many specialized operations to a generalized operation.
- Refactor common behavior to simplify and generalize the code.
  - brewGrounds() and steepTea() to just brew()

Template Method Pattern: definition

- Defines the skeleton of an algorithm in a method, deferring some steps to subclasses so they can redefine certain steps of an algorithm without changing the algorithm’s structure.

The Hook

- Problem:
  - What if a customer wants tea or coffee without condiments?
  - What if the user (driver code programmer) wants to add a step?
- The pattern provides a hook
  - A way to extend the algorithm
    - extend, not change

Hook Methods

- A hook method is a Non-Abstract method defined in the super class that should be overridden in the concrete classes if required
- Generally contain nothing
  - ```
    def hook()
    end
    ```
    - If the customer wants no condiments, hook gets called, and nothing happens


    ```

Hook Methods

- Different than abstract methods
  - ```
    def abstractMethod()
        raise NoMethodError
    end
    ```
- Hook methods may contain default code
  - ```
    def hook()
        defaultAction()
    end
    ```

Template Method Pattern: hook

```
class TemplateClass
   def templateMethod()
      baseOperation()
      subOperation()
      hook()
   end
end
```

Hook adds extensibility

```
def hook()
   additionalOperation1()
   additionalOperation2()
   additionalOperation3()
end
```

Now your Template Method looks like...

```
class TemplateClass
   def templateMethod()
      baseOperation()
      subOperation()
      hook()
   end
end
```

```
additionalOperation1()
additionalOperation2()
additionalOperation3()
```

Template Method Pattern: hook (2)

```
class TemplateClass
   def templateMethod()
      baseOperation()
      baseOperation()
      subOperation()
      if(hookNeeded())
         hook()
      end
   end
```

CaffieneBeverage

```
class CaffieneBeverageClass
   def prepare()
      boilWater()
      brew()
      pourCup()
      if(condimentsWanted())
         addCondiments()
      end
   end
```

Hook adds extensibility

```
def addCondiments()
   addHoney()
   addLemon()
   if(isOver21())
      addWhiskey()
   end
end
```

Template Method Methods

- The Template Method has 3 kinds of methods
  - Base common methods
    - These are the shared methods that all subclasses will use
  - Abstract methods that must be overridden
    - The specific functionality that is unique to the subclass
  - Hooks
    - The additional functionality that cannot be predicted

Template Method Pattern (1)

- You can vary behavior using a simple kind of inheritance
- The idea is that most of an algorithm or procedure is fixed; the detailed behavior depends on calls to specific operations
- Template methods are a fundamental technique for DRY code
  - Do Not Repeat Yourself
- They are particularly important in class libraries, although its use is often hidden

Hollywood Principle

- "Hollywood principle"
  - "Don't call us, we'll call you”
- Question?
  - How is the “Hollywood Principle” different from typical inheritance?
- Clients of Tea/Coffee use the superclass abstraction.
  - reduces dependency between backend code and the subclasses

Template Method Problems

- What design principle does the template method violate?
  - Favor composition over inheritance
- What if the steps change? What has to change?
  - The base class has to change, and possibly the subclasses have to be refactored
- How flexible is a change during runtime?
  - Inheritance relationships are decided at compile time

Classwork: Loan Software