Informatics 122 Software Design II

Lecture 9 Template Design Pattern

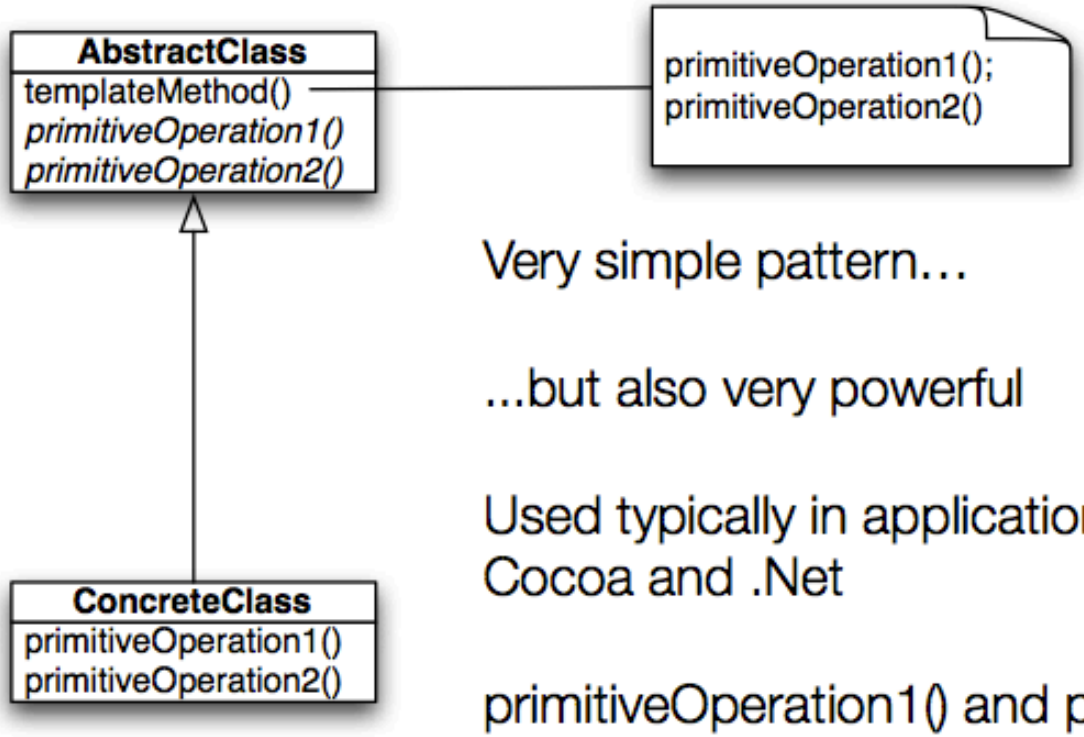
Emily Navarro

Portions of the slides in this lecture are adapted from http://www.cs.colorado.edu/~kena/classes/5448/f12/lectures/

Template Method: Definition

- The Template Method pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure
  - Template Method defines the steps of an algorithm and allows subclasses to provide the implementation for one or more steps
    - Makes the algorithm abstract
      - Each step of the algorithm is represented by a method
    - Encapsulates the details of most steps
      - Steps (methods) handled by subclasses are declared abstract
      - Shared steps (concrete methods) are placed in the same class that has the template method, allowing for code re-use among the various subclasses

Template Method: Structure



Very simple pattern…

...but also very powerful

Used typically in application frameworks, e.g. Cocoa and .Net

primitiveOperation1() and primitiveOperation2() are sometimes referred to as hook methods as they allow subclasses to hook their behavior into the service provided by AbstractClass

Example: Tea and Coffee

- Consider an example in which we have recipes for making tea and coffee at a coffee shop
  - Coffee
    - Boil water
    - Brew coffee in boiling water
    - Pour coffee in cup
    - Add sugar and milk
  - Tea
    - Boil water
    - Steep tea in boiling water
    - Pour tea in cup
    - Add lemon

Coffee Implementation

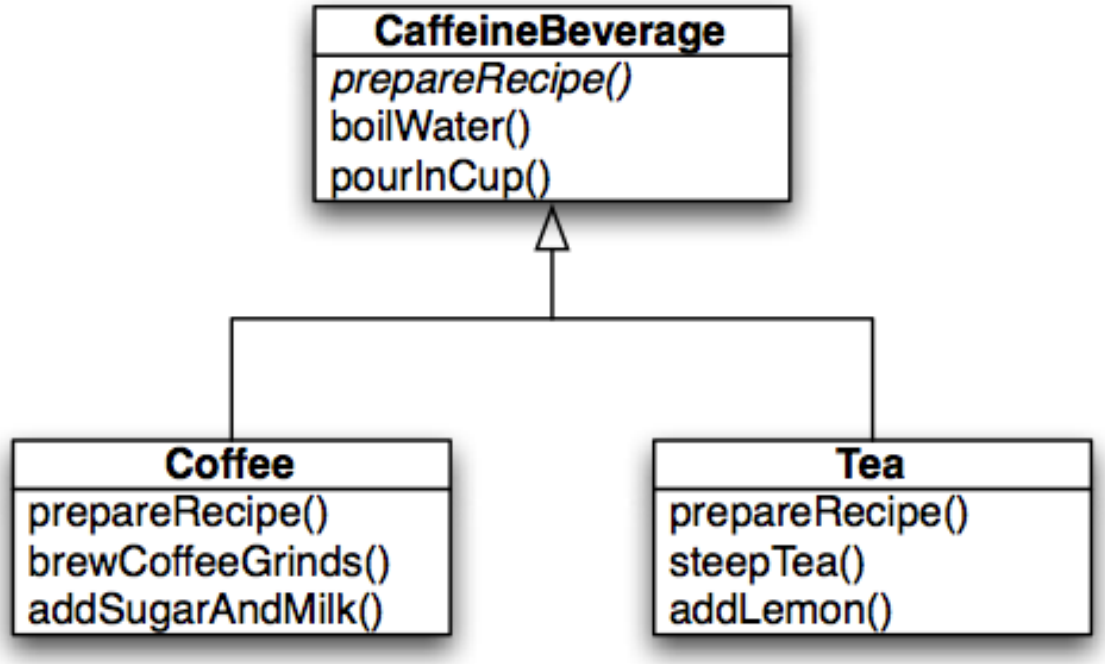
```
1  public class Coffee {
2
3      void prepareRecipe() {
4          boilWater();
5          brewCoffeeGrinds();
6          pourInCup();
7          addSugarAndMilk();
8      }
9
10     public void boilWater() {
11         System.out.println("Boiling water");
12     }
13
14     public void brewCoffeeGrinds() {
15         System.out.println("Dripping Coffee through filter");
16     }
17
18     public void pourInCup() {
19         System.out.println("Pouring into cup");
20     }
21
22     public void addSugarAndMilk() {
23         System.out.println("Adding Sugar and Milk");
24     }
25 }
26
```

Tea Implementation

```
1  public class Tea {
2
3      void prepareRecipe() {
4          boilWater();
5          steepTeaBag();
6          pourInCup();
7          addLemon();
8      }
9
10     public void boilWater() {
11         System.out.println("Boiling water");
12     }
13
14     public void steepTeaBag() {
15         System.out.println("Steeping the tea");
16     }
17
18     public void addLemon() {
19         System.out.println("Adding Lemon");
20     }
21
22     public void pourInCup() {
23         System.out.println("Pouring into cup");
24     }
25 }
26
```

Code Duplication!

- We have code duplication occurring in these two classes
  - boilWater() and pourInCup() are exactly the same
- Let's get rid of the duplication



Similar Algorithms

- The structure of the algorithms in prepareRecipe() is similar for Tea and Coffee
  - We can improve our code further by making the code in prepareRecipe() more abstract
    - brewCoffeeGrinds() and steepTea() -> brew()
    - addSugarAndMilk() and addLemon() -> addCondiments()
- Now all we need to do is specify this structure in CaffeineBeverage.prepareRecipe() and make sure we do it in such a way so that subclasses can't change the structure
  - By using the word "final" (see next slide)

Caffeine Beverage

```
1  public abstract class CaffeineBeverage {
2
3      final void prepareRecipe() {
4          boilWater();
5          brew();
6          pourInCup();
7          addCondiments();
8      }
9
10     abstract void brew();
11
12     abstract void addCondiments();
13
14     void boilWater() {
15         System.out.println("Boiling water");
16     }
17
18     void pourInCup() {
19         System.out.println("Pouring into cup");
20     }
21 }
22
```

Note: use of final keyword for prepareRecipe()

brew() and addCondiments() are abstract and must be supplied by subclasses

boilWater() and pourInCup() are specified and shared across all subclasses

Coffee and Tea Implementations

```
1  public class Coffee extends CaffeineBeverage {
2      public void brew() {
3          System.out.println("Dripping Coffee through filter");
4      }
5      public void addCondiments() {
6          System.out.println("Adding Sugar and Milk");
7      }
8  }
9
10 public class Tea extends CaffeineBeverage {
11     public void brew() {
12         System.out.println("Steeping the tea");
13     }
14     public void addCondiments() {
15         System.out.println("Adding Lemon");
16     }
17 }
18
```

Nice and simple!

What have we done?

- Took two separate classes with separate but similar algorithms
- Noticed duplication and eliminated it by adding a superclass
- Made steps of algorithm more abstract and specified its structure in the superclass
  - Thereby eliminating another "implicit" duplication between the two classes
- Revised subclasses to implement the abstract (unspecified) portions of the algorithm… in a way that made sense for them