Template Pattern

CSCI-4448 - Boese

University of Colorado **Boulder**

Objectives

- Problem
- Definition
- Why
- Examples
- How
- Comparisons

Problem: StarBuzz

Problem: StarBuzz

- Coffee Recipe
  - –Boil some water
  - –Brew coffee in boiling water
  - –Pour coffee in cup
  - –Add sugar and milk

- Tea Recipe
  - –Boil some water
  - –Steep tea in boiling water
  - –Pour tea in cup
  - –Add lemon

- Suppose you are required to implement a system to maintain this
- Don't want duplicate code
- Adding a new beverage would result in further duplication.
- Knowledge of the algorithm and implementation is distributed over classes.

Problems with the Solution

Coffee

prepareCoffee( )
boilWater( )
brew( )
pourInCup( )
addSugarMilk( )

Tea

prepareTea( )
boilWater( )
steep( )
pourInCup( )
addLemon( )

- **Code is duplicated** across the classes – code changes would have to be made in more than one place.
- **Adding** a new beverage would result in **further duplication.**
- **Knowledge of the algorithm and implementation is distributed** over classes.

Problem: StarBuzz

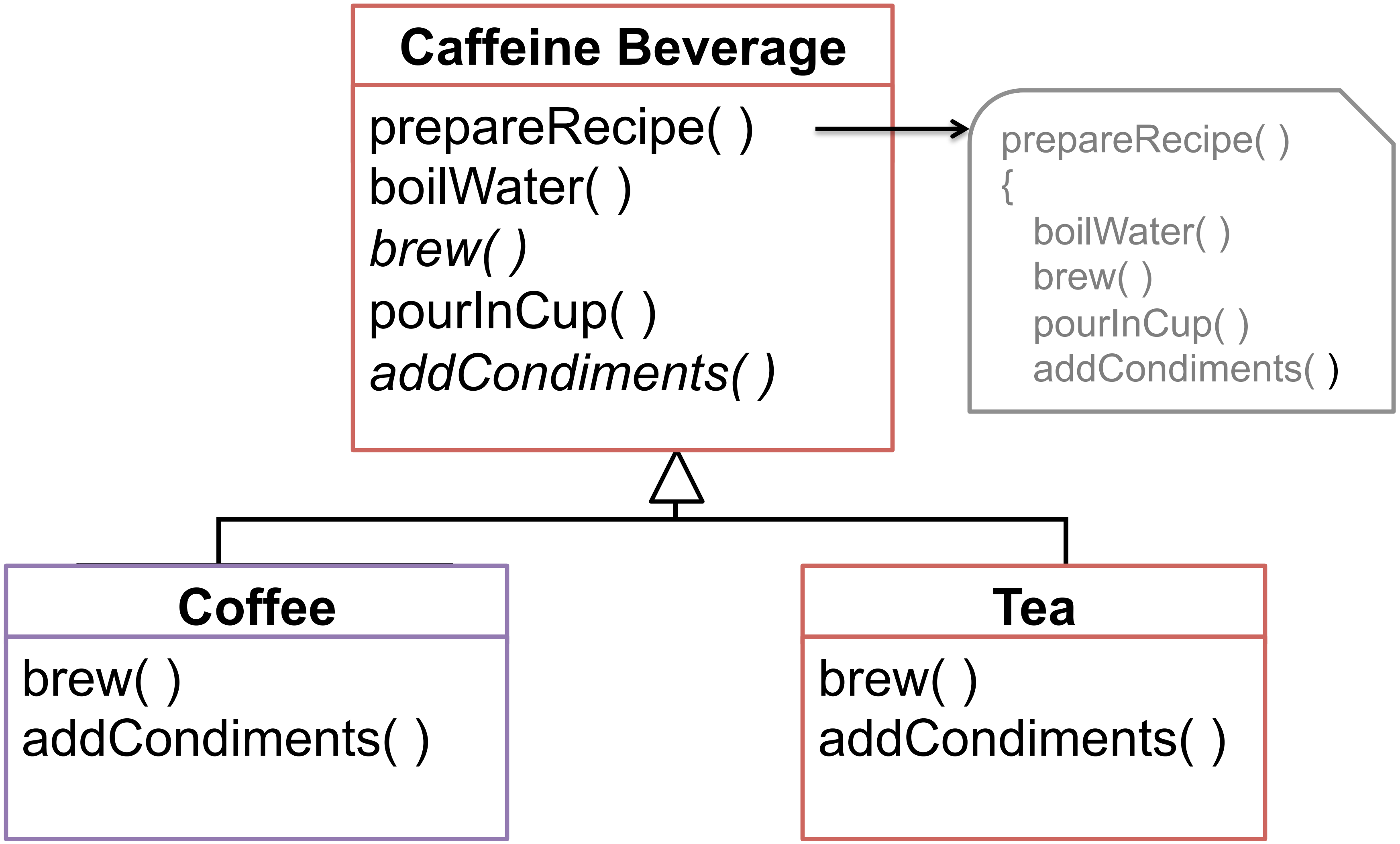
- Coffee Recipe
  - Boil some water
  - Brew coffee in boiling water
  - Pour coffee in cup
  - Add sugar and milk

- Tea Recipe
  - Boil some water
  - Steep tea in boiling water
  - Pour tea in cup
  - Add lemon

- What is the same?

- What varies?

Abstracting Prepare Recipe



More General Approach

- Both subclasses **inherit a general algorithm**.
  - The *prepareRecipe( )* method implements the template pattern.
  - Each step in algorithm is represented by a method.

```
prepareRecipe( )
{
   boilWater( )
   brew( )
   pourInCup( )
   addCondiments( )
}
```

- Some methods in the algorithm are **concrete**, i.e. methods that perform the same actions for all subclasses.
- Other methods in the algorithm are **abstract**, i.e. methods that perform class-specific actions.

Advantages of the New Approach

- **A single class protects and controls the algorithm**, namely, CaffeineBeverage.
- The **superclass facilitates reuse** of methods.
- **Code changes will occur in only one place**.
- Other beverages can be **easily added**.

This is the Template Pattern

- The template pattern defines the steps of an algorithm and allows the subclasses to implement one or more of the steps.

Definition

Definition

“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 algorithms structure.”

-Gang of Four

Definition

- **Name** “Template”
- **Intent**
  - Define the skeleton of an algorithm in an operation, deferring some steps to subclasses.

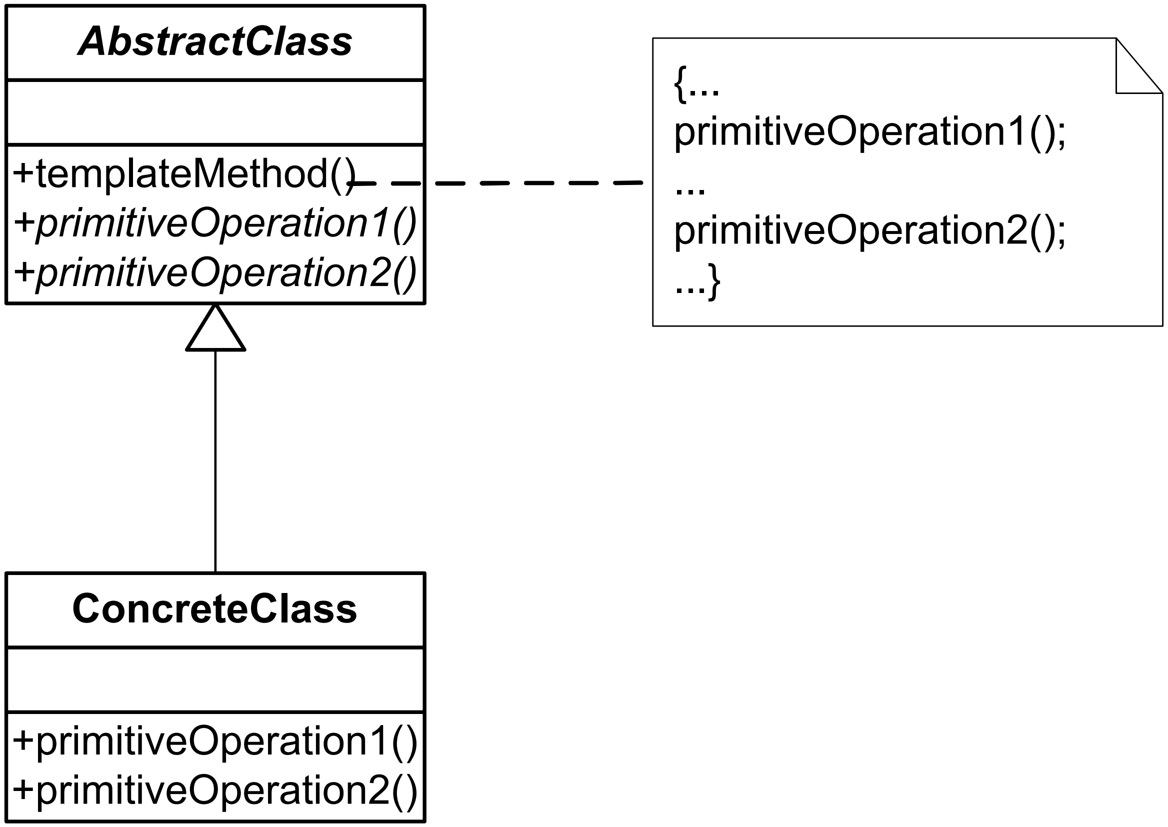
Why

Why use Template Pattern?

- To implement invariant aspects of an algorithm once and let subclasses define variant parts
- To localize common behaviour in a class to increase code reuse
- To control subclass extensions
- Lets you enforce overriding rules

- **Known Uses**
  - Just about all object-oriented systems (especially frameworks)

Template Design Pattern Structure



Hooks

Using Hooks

- We want to minimize the number of abstract methods used.
- Thus, the steps of the algorithm should not be too granular.
- However, less granularity means less flexibility.
- **Hooks** are methods which can be overridden by subclasses, however this is ***optional*** (as opposed to **abstract methods** which are ***required*** to be overridden)
- Example: Suppose the customer is given an option as to whether they would like condiments or not.

Examples of Using Hooks in the Java API

- `JFrame` hooks
  - `paint()`
- Applet hooks
  - `init()`
  - `repaint()`
  - `start()`
  - `stop()`
  - `destroy()`
  - `paint()`

Java Sorting

Sorting Using the Template Pattern

- Java's **`Arrays`** class provides a template method for sorting.
- The sort is a merge sort and requires a method ***compareTo( )***.

```
public static void sort(Object a[ ]) {
      Object aux[ ] = (Object a[ ])a.clone);
      mergeSort(aux,a,0,a.length,0);
 }
private static void mergeSort(Object src[ ], Object dest[ ], int low,int high, int off)
{
   for(int i=low; i < high; ++i)
   {
     for(int j=i; j < low; &&
      ((Comparable)dest[j-1).compareTo((Comparable)dest[j])>0;j--)
      {
          swap(dest, j, j-1);
      }
   }
   return;
 }
```

How

Template Pattern

- **Encapsulates an algorithm** by creating a template for it.
- Defines the skeleton of an algorithm as a set of steps.
- Some methods of the algorithm have to be implemented by the subclasses – these are abstract methods in the super class.
- The subclasses can redefine certain steps of the algorithm without changing the algorithm's structure.
- Some steps of the algorithm are concrete methods defined in the super class.

Hollywood Principle

- The Template pattern follows the **Hollywood principle**.
  - Principle: Don’t call us, we will call you.
- Low-level components are activated by high-level components.
- A low-level component never calls a high-level component.
- In the template pattern the abstract class is the high-level component and the concrete classes the low-level components.

How – Force Override Operation Methods

```
public abstract class MyClass
{
 ...
 // A template method!
 public final void templateMethod()
 {
   primitiveOperation1();
   primitiveOperation2();
 }
 public abstract void primitiveOperation1();
 public abstract void primitiveOperation2();
 ...
}
```

How – Default Operation Methods

```
public class MyClass
{
 ...
 // A template method!
 public final void templateMethod()
 {
   ConcreteOperation1();
   ConcreteOperation2();
 }
 public void ConcreteOperation1()
 {
       // Default behavior for Operation 1
 }
 public void ConcreteOperation2()
 {
       // Default behavior for Operation 2
 }
 ...
}
```

Both

```
public abstract class AbstractClass
{
    final void templateMethod()
    {
        primitiveOperation1();
        primitiveOperation2();
        concreteOperation();
    }
    abstract void primitiveOperation1();
    abstract void primitiveOperation2();

    void concreteOperation()
    {
      //Implementation
    }
}
```

Comparisons

Comparisons

- Similar to the strategy pattern.
- The Factory pattern is a specialization of the Template pattern.

In Summary…

- Design Principle: Don't call us we'll call you.
- Template pattern defines steps of an algorithm.
- Subclasses cannot change the algorithm - final
- Facilitates code reuse.