Instituto de computação uff

Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Padrão Template Method

Documento baseado no material preparado pelo Prof. Luiz André (http://www.ic.uff.br/~lapaesleme/)

Padrões de Projeto

O Padrão Template Method

Como encapsular algoritmos

Considere o preparo de duas receitas

Receita de Café:

(1) Ferver um pouco de água
(2) Colocar o café na água fervendo
(3) Servir o café em uma chícara
(4) Adicionar açucar e leite

Receita de Chá:

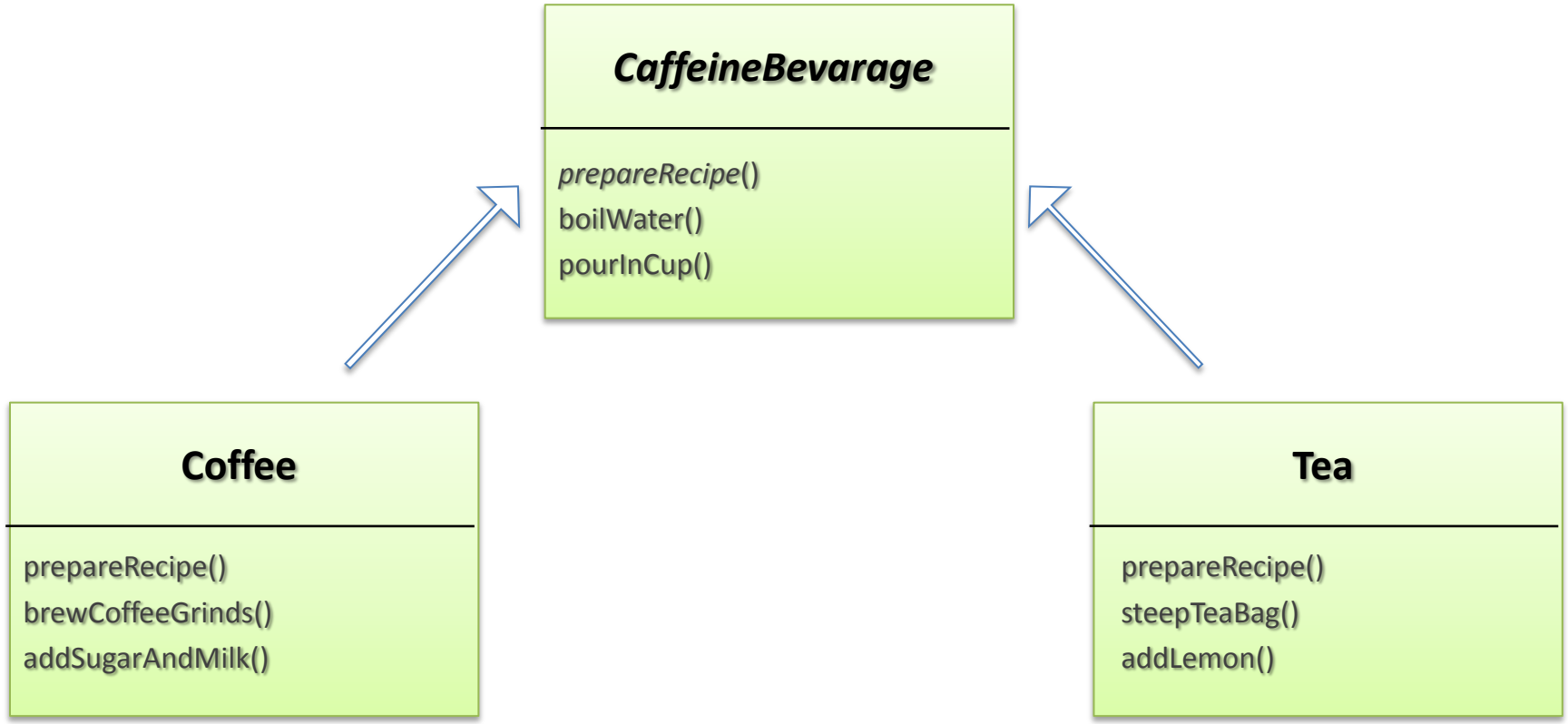
(1) Ferver um pouco de água
(2) Submergir o chá na água fervendo
(3) Servir o chá em um chícara
(4) Adicionar limão

Código

```
public class Tea {
  void prepareRecipe() {
    boilWater();
    steepTeaBag();
    pourInCup();
    addLemon();
  }
  public void boilWater() {
    System.out.println("Boiling water");
  }
  public void steepTeaBag() {
    System.out.println("Steeping the tea");
  }
  public void addLemon() {
    System.out.println("Adding Lemon");
  }
  public void pourInCup() {
    System.out.println("Pouring into cup");
  }
}
```

```
public class Coffee {
  void prepareRecipe() {
    boilWater();
    brewCoffeeGrinds();
    pourInCup();
    addSugarAndMilk();
  }
  public void boilWater() {
    System.out.println("Boiling water");
  }
  public void brewCoffeeGrinds(){
    System.out.println("Dripping Coffee through filter");
  }
  public void pourInCup() {
    System.out.println("Pouring into cup");
  }
  public void addSugarAndMilk() {
    System.out.println("Adding Sugar and Milk");
  }
}
```

Uma primeira solução



Problemas e limitações??

- Método prepareRecipe() é declarado como abstrato
- Ele é completamente definido apenas nas subclasses
- Entretanto existe muito em comum entre o prepareRecipe() para Tea e Coffee
- Não estamos fazendo uma abstração que permita um bom reuso

Problemas e limitações??

- Método prepareRecipe() é declarado como abstrato
- Ele é completamente definido apenas nas subclasses
- Entretanto existe muito em comum entre o prepareRecipe() para Tea e Coffee
- Não estamos fazendo uma abstração que permita um bom reuso

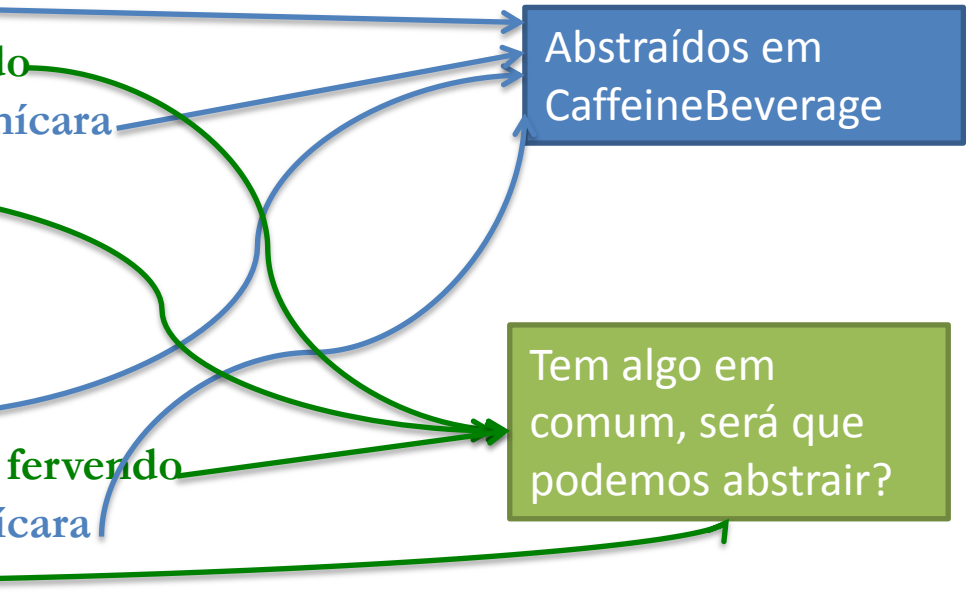
O que podemos notar se olharmos as duas receitas com cuidado?

Receita de Café:

(1) Ferver um pouco de água
(2) Colocar o café na água fervendo
(3) Servir o café(bebida) em uma chícara
(4) Adicionar açucar e leite

Receita de Chá:

(1) Ferver um pouco de água
(2) Fazer a infusão do chá na água fervendo
(3) Servir o chá(bebida) em um chícara
(4) Adicionar limão



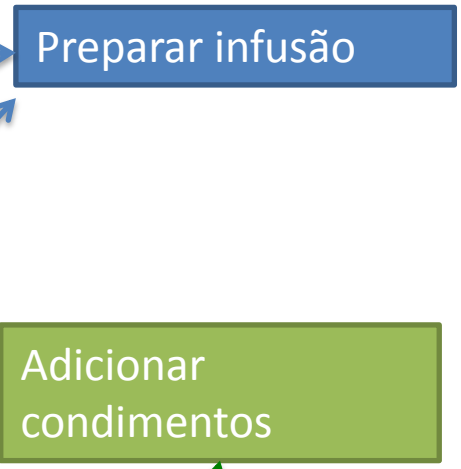
O que podemos notar se olharmos as duas receitas com cuidado?

Receita de Café:

(1) Ferver um pouco de água
(2) Colocar o café na água fervendo
(3) Servir o café(bebida) em uma chícara
(4) Adicionar açucar e leite

Receita de Chá:

(1) Ferver um pouco de água
(2) Fazer a infusão do chá na água fervendo
(3) Servir o chá(bebida) em um chícara
(4) Adicionar limão



Receita de Bebida Genérica

(1) Ferver um pouco de água
(2) Preparar infusão
(3) Servir a bebida em uma chícara
(4) Adicionar codimento

Padrões de Projeto

O Padrão Template Method

```
public abstract class CaffeineBeverage {
        final void prepareRecipe() {
                boilWater();
                brew();
                pourInCup();
                addCondiments();
        }
        abstract void brew();
        abstract void addCondiments();

        void boilWater() {
                System.out.println("Boiling water");
        }
        void pourInCup() {
                System.out.println("Pouring into cup");
        }
}
```

Não queremos que a receita (algoritmo) mude

Devem ser implementados nas classes filhas

Padrões de Projeto

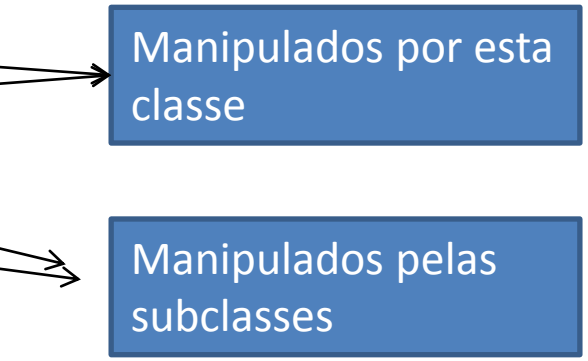
O Padrão Template Method

```
public class Tea extends CaffeineBeverage {
        public void brew() {
                System.out.println("Steeping the tea");
        }
        public void addCondiments() {
                System.out.println("Adding Lemon");
        }
}

public class Coffee extends CaffeineBeverage {
        public void brew() {
                System.out.println("Dripping Coffee through filter");
        }
        public void addCondiments() {
                System.out.println("Adding Sugar and Milk");
        }
}
```

Padrões de Projeto

O Padrão Template Method

```
public abstract class CaffeineBeverage {
    void final prepareRecipe() {
        boilWater();
        brew();
        pourInCup();
        addCondiments();
    }
    abstract void brew();
    abstract void addCondiments();

    void boilWater() {
    // implementation
    }
    void pourInCup() {
    // implementation
    }
}
```



Padrões de Projeto

O Padrão Template Method

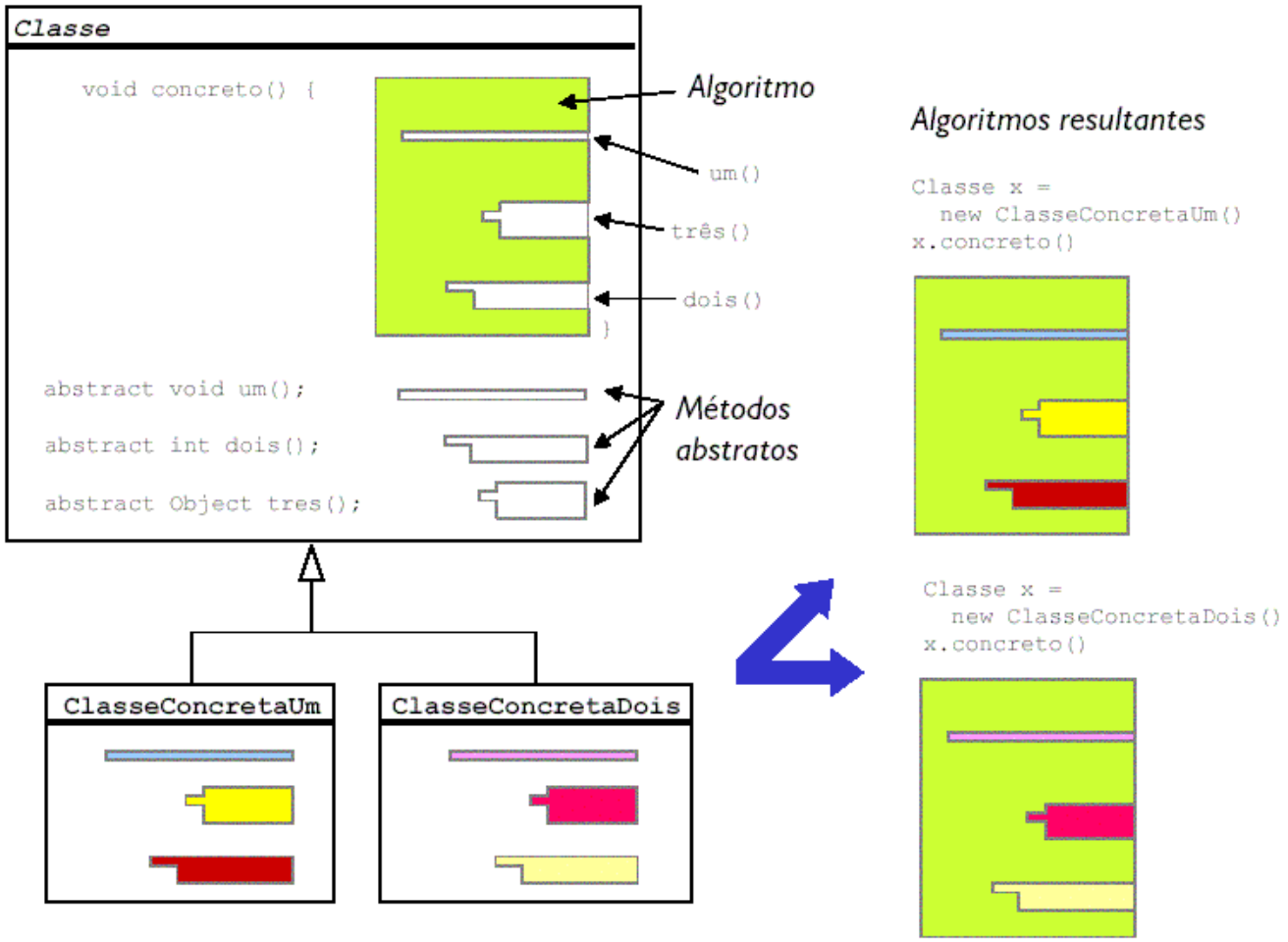
O padrão Template Method define os passos de um algoritmo e permiteque as subclasses forneçam a implementação de um ou mais passos

Definição

O Padrão Template Method define o esqueleto de um algoritmo dentro de um método, transferindo alguns de seus passos para as subclasses. O Template Method permite que as subclasses redefinam certos passos de um algoritmo sem alterar a estrutura do próprio algoritmo.

Padrões de Projeto

O padrão Template Method

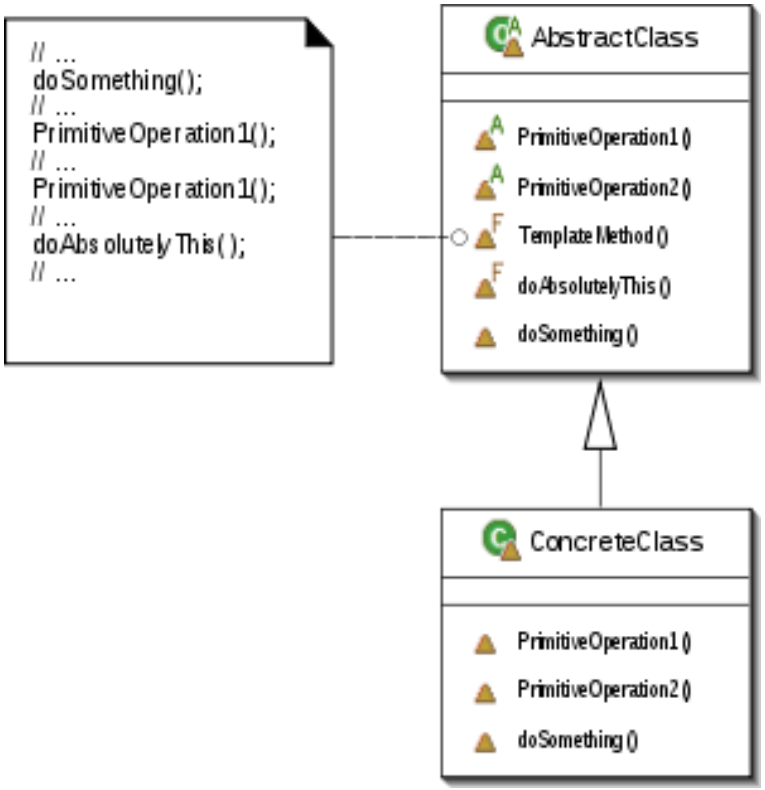


- AbstractClass
  - Define operações primitivas abstratas que implementam passos de um algoritmo.
  - Implementa um método template que define o esqueleto de um algoritmo. O método template chama as operações abstratas.

- ConcreteClass
  - Implementa as operações primitivas para realizar passos específicos do algoritmo

O padrão Template Method - participantes

- Diagrama de classes



Padrões de Projeto

O padrão Template Method – Código de referência

```
abstract class AbstractClass {
    final void templateMethod() {
        primitiveOperation1();
        primitiveOperation2();
        concreteOperation();
    }
    abstract void primitiveOperation1();
    abstract void primitiveOperation2();
    void concreteOperation() {
        // implementation here
    }
}
```

Padrões de Projeto

O padrão Template Method – o hook

- Como explorar os métodos concretos em um template method
- O uso do hook ou gancho
- O gancho é um método concreto com uma implementação vazia que permite que as subclasses interajam com o algoritmo da classe abstrata

Padrões de Projeto

O padrão Template Method – Hooks

```
abstract class AbstractClass {
    final void templateMethod() {
        primitiveOperation1();
        primitiveOperation2();
        concreteOperation();
        hook();
    }
    abstract void primitiveOperation1();
    abstract void primitiveOperation2();
    void concreteOperation() {
        // implementation here
    }
    void hook(){};
}
```

As subclasses são livres para sobreescrever o hook mas podem não fazê-lo se desejarem

- Os ganchos fornecem a s subclasses a possibilidade de se pendurarem nos algoritmos em vários pontos se desejarem ou ignorarem o gancho

- Vejamos um exemplo de uso

O padrão Template Method – Hooks

```
public abstract class CaffeineBeverageWithHook {
        final void prepareRecipe() {
                boilWater();
                brew();
                pourInCup();
                if (customerWantsCondiments()) {
                addCondiments();
                }
        }
        abstract void brew();
        abstract void addCondiments();
        void boilWater() {
        System.out.println("Boiling water");
        }
        void pourInCup() {
        System.out.println("Pouring into cup");
        }
        boolean customerWantsCondiments() {
        return true;
        }
}
```

Padrões de Projeto

O padrão Template Method – Hooks

```
public class CoffeeWithHook extends
      CaffeineBeverageWithHook {
        public void brew() {
            System.out.println("Dripping Coffee
            through filter");
        }
        public void addCondiments() {
            System.out.println("Adding Sugar and
            Milk");
        }
        public boolean customerWantsCondiments() {
              String answer = getUserInput();
              if (answer.toLowerCase().startsWith("y"))
                  {
                  return true;
              } else {
                  return false;
              }
    }
```

```
private String getUserInput() {
        String answer = null;
        System.out.print("Would you like milk and
        sugar with your coffee (y/n)? ");

        BufferedReader in = new BufferedReader(new
        InputStreamReader(System.in));

        try {
              answer = in.readLine();
        } catch (IOException ioe) {
              System.err.println("IO error trying to read
              your answer");
        }

        if (answer == null) {
              return "no";
        }

        return answer;
        }
}
```

```
public class BeverageTestDrive {
    public static void main(String[] args) {
        TeaWithHook teaHook = new TeaWithHook();

    CoffeeWithHook coffeeHook = new CoffeeWithHook();
    System.out.println("\nMaking tea...");
    teaHook.prepareRecipe();
    System.out.println("\nMaking coffee...");
    coffeeHook.prepareRecipe();
    }
}
```

Padrões de Projeto

O padrão Template Method – Hooks

```
/** * An abstract class that is common to several games in  which players play against the others, but only one is
      * playing at a given time. */

abstract class Game {
      protected int playersCount; abstract void initializeGame();
      abstract void makePlay(int player);
      abstract boolean endOfGame();
      abstract void printWinner(); /* A template method : */
       public final void playOneGame(int playersCount) {
                this.playersCount = playersCount;       initializeGame();
                int j = 0;
                 while (!endOfGame()) {
                   makePlay(j); j = (j + 1) % playersCount;
                }
       printWinner();
       }
}
```

Padrões de Projeto

O padrão Template Method – Hooks

```
//Now we can extend this class in order
 //to implement actual games:

class Monopoly extends Game { /* Implementation of necessary concrete methods */
        void initializeGame() {
         // Initialize players
         // Initialize money
        }
        void makePlay(int player) {
        // Process one turn of player
         }
        boolean endOfGame() {
         // Return true if game is over
        // according to Monopoly rules
        }
        void printWinner() {
         // Display who won
        } /* Specific declarations for the Monopoly game. */ //
... }
```

Padrões de Projeto

O padrão Template Method – Hooks

```
class Chess extends Game { /* Implementation of necessary concrete methods */
        void initializeGame() {
                // Initialize players
                // Put the pieces on the board
        }
        void makePlay(int player) {
                // Process a turn for the player
        }
        boolean endOfGame() {
                // Return true if in Checkmate or
                // Stalemate has been reached
        }
        void printWinner() {
                // Display the winning player
        }
        /* Specific declarations for the chess game. */ //
... }
```

Padrões de Projeto

Bibliografia

- Use a Cabeça ! Padrões de Projetos (design Patterns) - 2ª Ed. Elisabeth Freeman e Eric Freeman. Editora: Alta Books
- Padroes de Projeto – Soluções reutilizáveis de software orientado a objetos. Erich Gamma, Richard Helm, Ralph Johnson. Editora Bookman
- http://en.wikipedia.org/wiki/Template_method_pattern. Acessado em 23-10-2014 20:24h Brasília