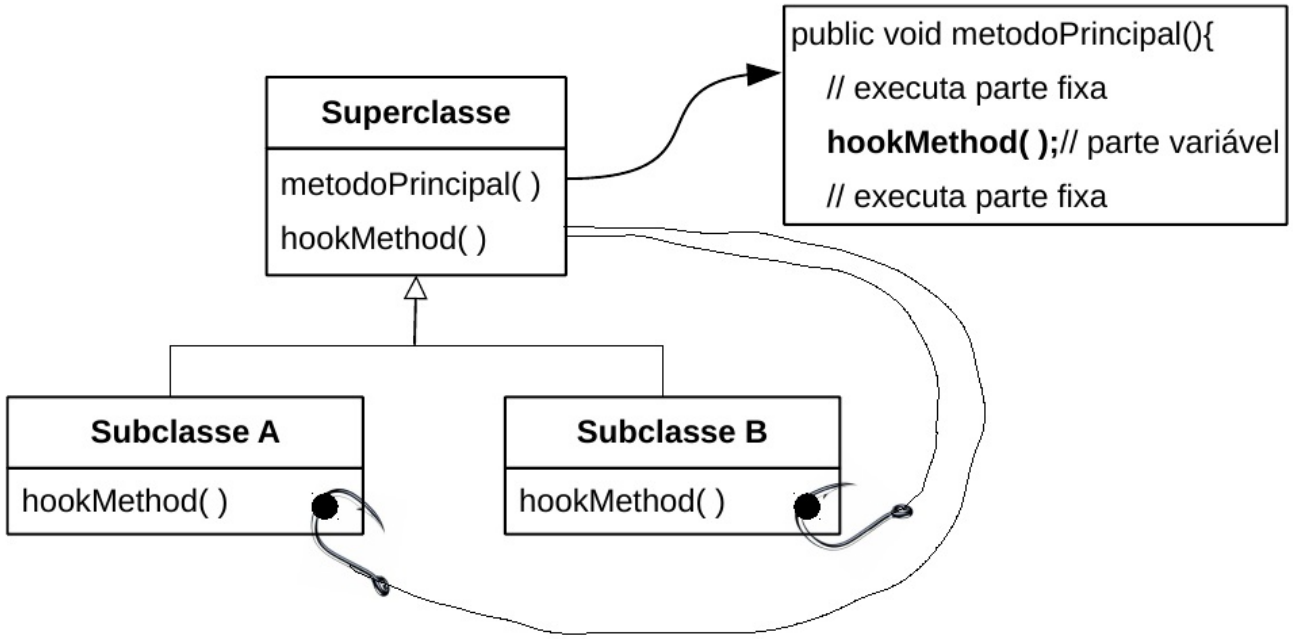
Template Method

Prof. Luiz Merschmann

Padrão de Projeto *Template Method*

Descrição do problema

- O padrão *Template Method* define em um método a estrutura ("esqueleto") de um algoritmo que possui algumas partes fixas e outras variáveis. As partes variáveis são representadas como *hook methods* que podem ser implementados nas subclasses.



Padrão de Projeto *Template Method*

Para espantar o sono, que tal implementarmos um sistema que prepare um café?

I 

COFFEE!

Tudo bem... Então nosso sistema também será capaz de preparar um chá!

RECEITA DO CAFÉ

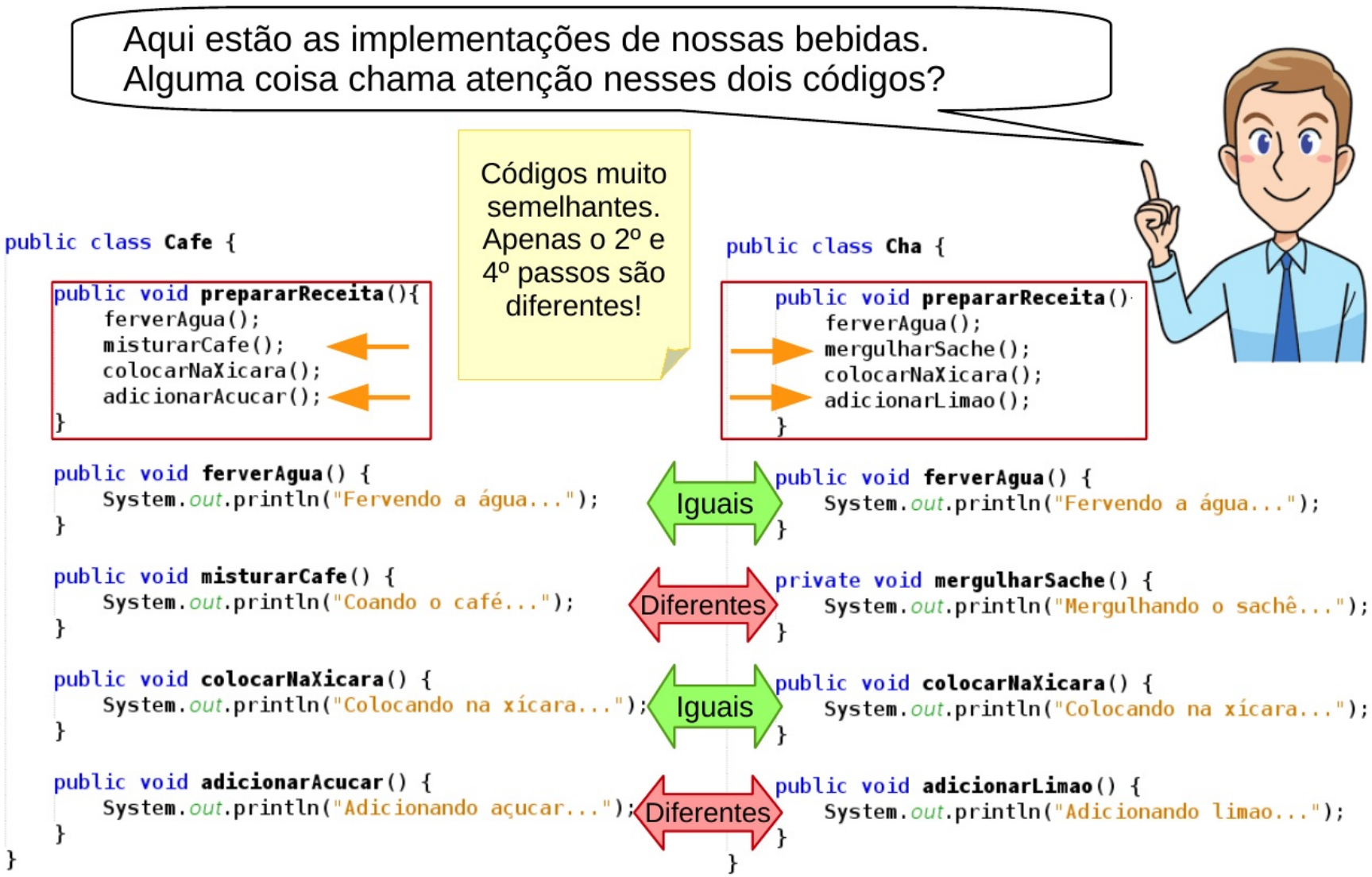
(1) Ferva um pouco de água
(2) Misture o café na água fervente
(3) Coloque em uma xícara
(4) Adicione açúcar

RECEITA DO CHÁ

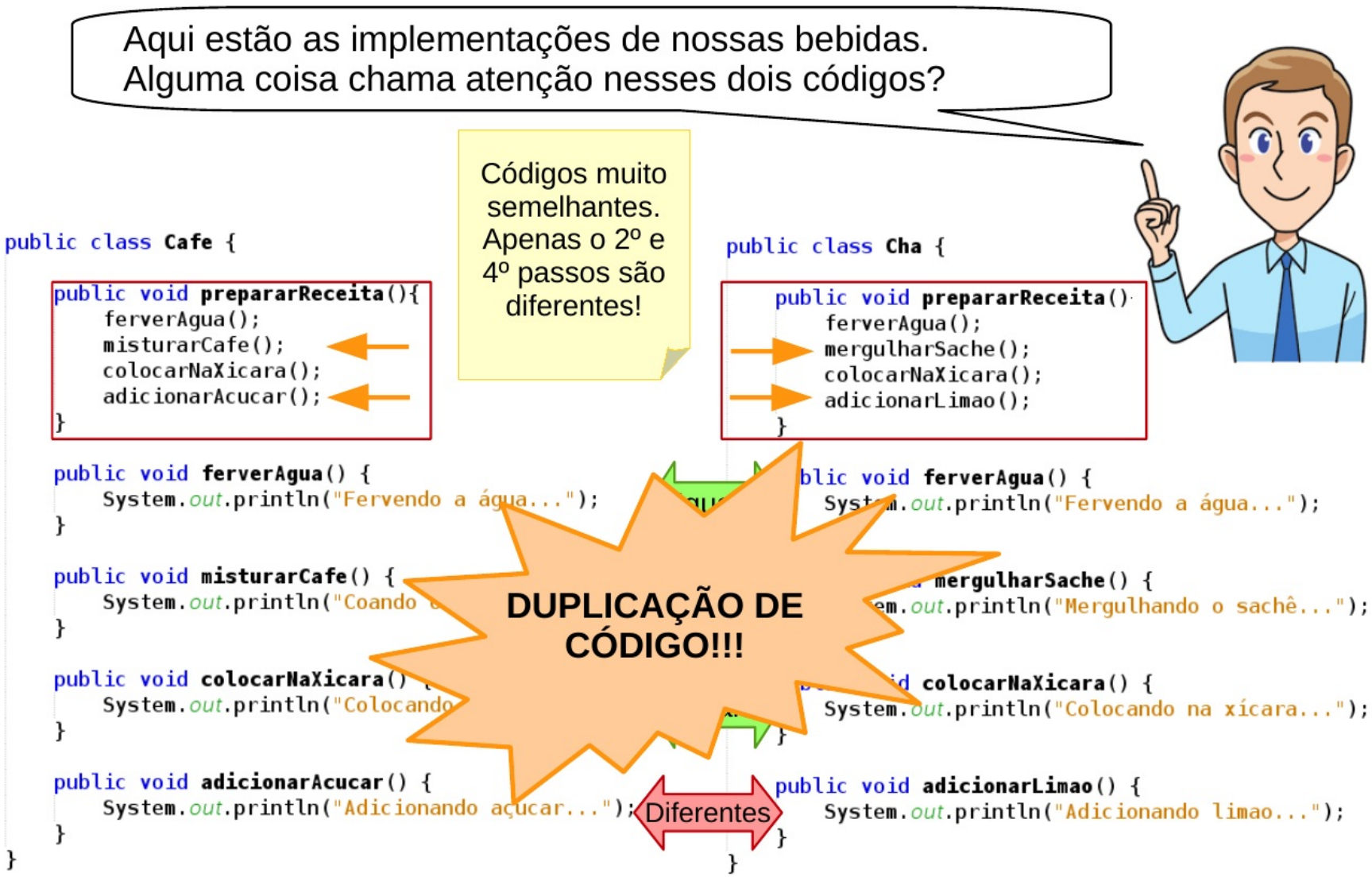
(1) Ferva um pouco de água
(2) Mergulhe o chá na água fervente
(3) Coloque em uma xícara
(4) Adicione limão

UFLA
UNIVERSIDADE FEDERAL DE LAVRAS

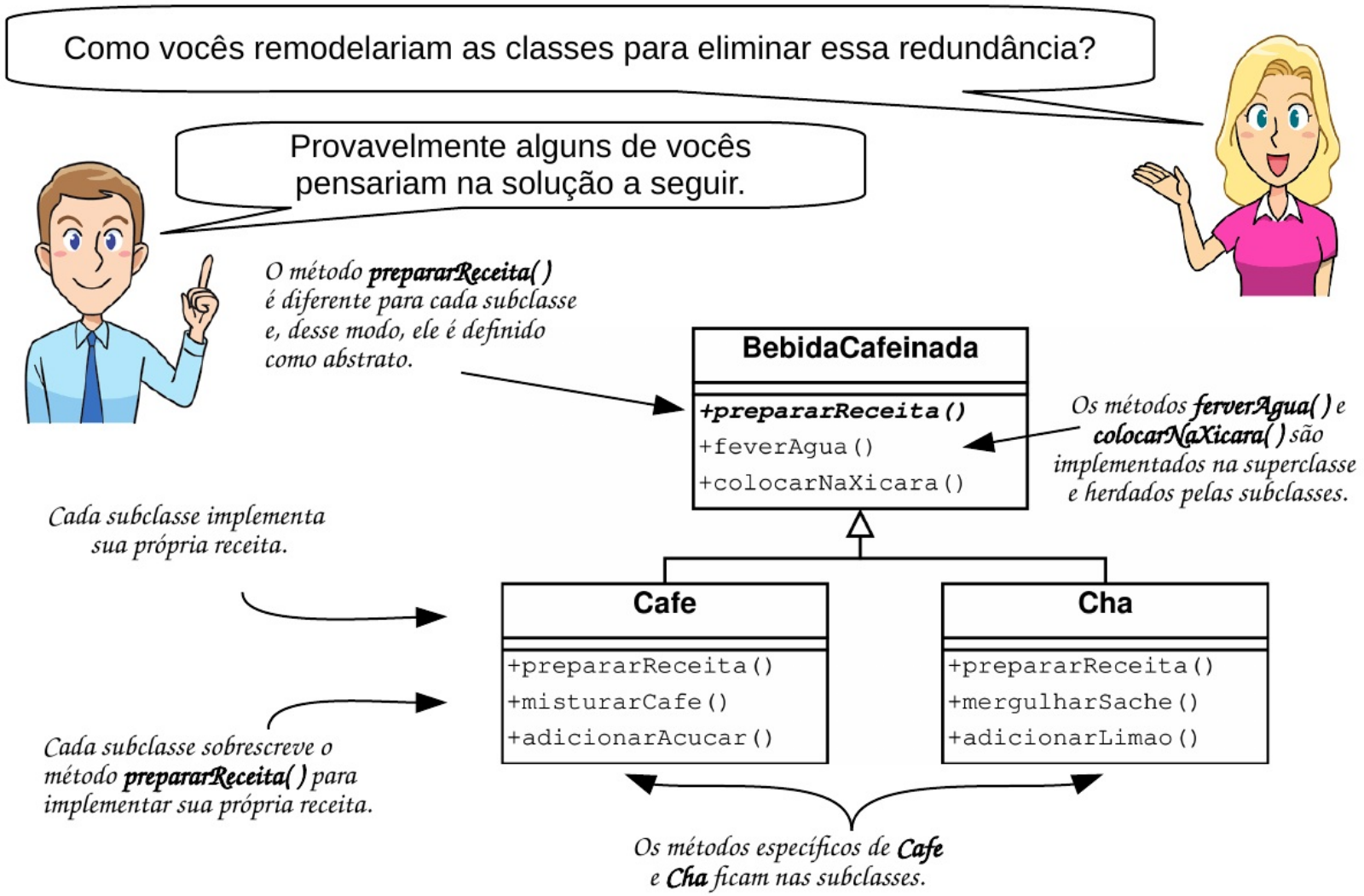
Padrão de Projeto *Template Method*



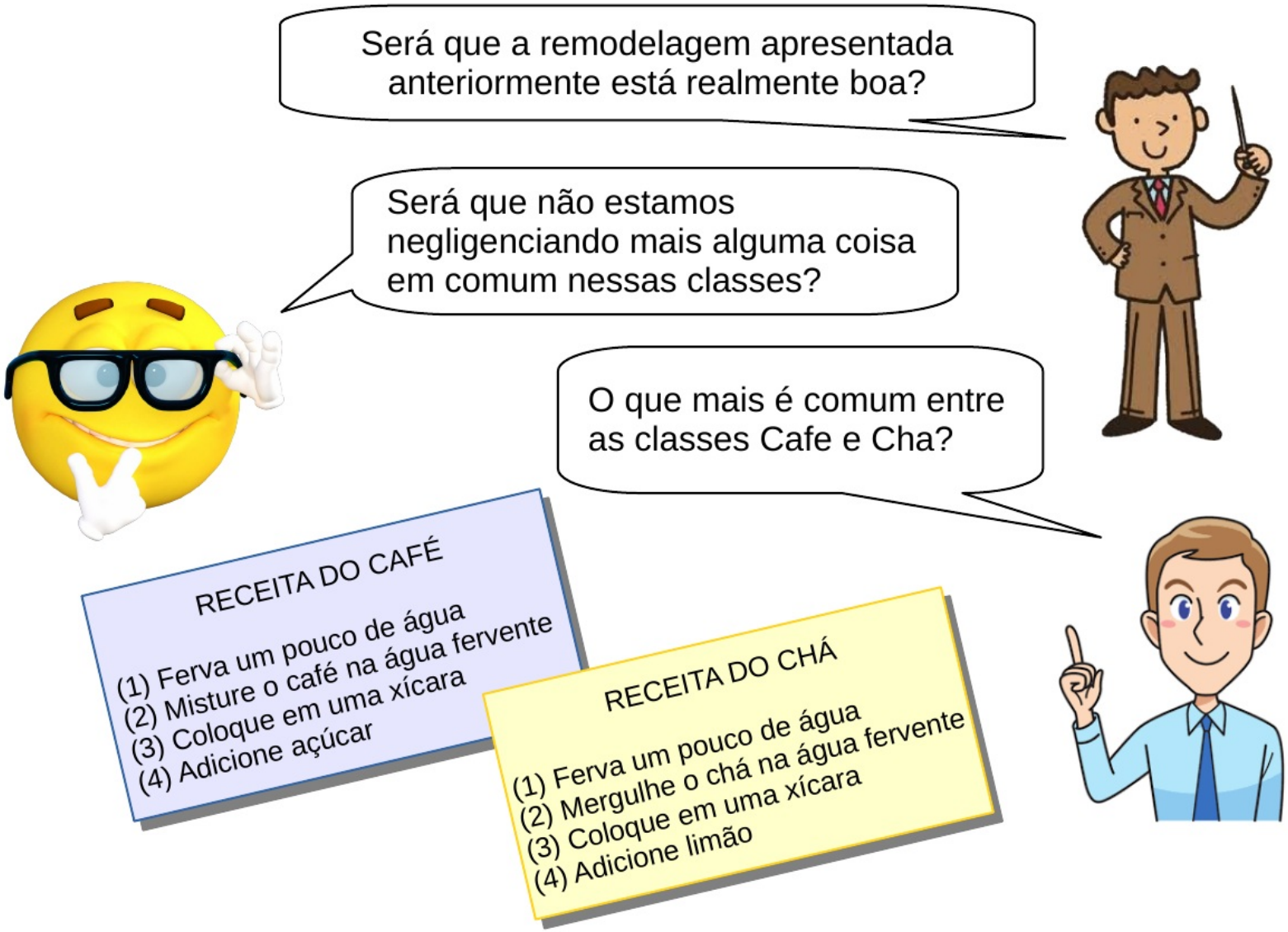
Padrão de Projeto *Template Method*



Padrão de Projeto *Template Method*



Padrão de Projeto *Template Method*



Padrão de Projeto *Template Method*

Note que as duas receitas seguem o mesmo algoritmo!

Esses métodos eram iguais e já estavam implementados na superclasse.

(1) Ferva um pouco de água

(2) Use água fervente para extrair a bebida

(3) Coloque a bebida em uma xícara

(4) Adicione os condimentos

Portanto, acabamos de encontrar uma maneira de abstrair o método prepararReceita( ).

Qual a importância disso para nossa implementação?

Já esses outros dois métodos não estavam implementados na superclasse. No entanto, eles são iguais, sendo apenas aplicados a diferentes bebidas.

Padrão de Projeto *Template Method*

Agora podemos incluir a implementação do método prepararReceita( ) na superclasse BebidaCafeinada.

```
public abstract class BebidaCafeinada {

    public final void prepararReceita(){
        ferverAgua();
        misturarComAguaFervente();
        colocarNaXicara();
        adicionarCondimentos();
    }

    public abstract void misturarComAguaFervente();

    public abstract void adicionarCondimentos();

    public void ferverAgua() {
        System.out.println("Fervendo a água...");
    }

    public void colocarNaXicara() {
        System.out.println("Colocando na xícara...");
    }
}
```

Essa classe é abstrata.

Agora o mesmo método **PrepararReceita( )** será usado para chá e café. Ele é declarado como **final** para evitar que as subclasses possam alterar a receita.

Como a implementação desses métodos é específica para café e chá, eles são declarados como abstratos e, portanto, implementados nas subclasses.

Assim como havíamos feito na remodelagem anterior, esses dois métodos continuam sendo implementados nesta classe.

Padrão de Projeto *Template Method*

Por fim, veja como ficam as classes Cafe e Cha...

```
public class Cafe extends BebidaCafeinada{

    @Override
    public void misturarComAguaFervente() {
        System.out.println("Coando o café...");
    }

    @Override
    public void adicionarCondimentos() {
        System.out.println("Adicionando açucar...");
    }
}
```

Assim como havíamos feito na remodelagem anterior, Cafe e Cha ***herdam*** *de BebidaCafeinada.*

Agora ***Cafe*** *precisa implementar os 2 métodos abstratos da classe* ***BebidaCafeinada****.*

```
public class Cha extends BebidaCafeinada{

    @Override
    public void misturarComAguaFervente() {
        System.out.println("Mergulhando o sachê...");
    }

    @Override
    public void adicionarCondimentos() {
        System.out.println("Adicionando limao...");
    }
}
```

Do mesmo modo, ***Cha*** *precisa implementar os 2 métodos abstratos da classe* ***BebidaCafeinada****.*

Padrão de Projeto *Template Method*

Analisando a estrutura da classe BebidaCafeinada ...

```
public abstract class BebidaCafeinada {

    public final void prepararReceita(){
        ferverAgua();
        misturarComAguaFervente();
        colocarNaXicara();
        adicionarCondimentos();
    }

    public abstract void misturarComAguaFervente();

    public abstract void adicionarCondimentos();

    public void ferverAgua() {
        System.out.println("Fervendo a água...");
    }

    public void colocarNaXicara() {
        System.out.println("Colocando na xícara...");
    }
}
```

Temos aqui nosso ***template method****. Ele serve como um template para um algoritmo.*

Nesse template, cada etapa do algoritmo é representada por um método.

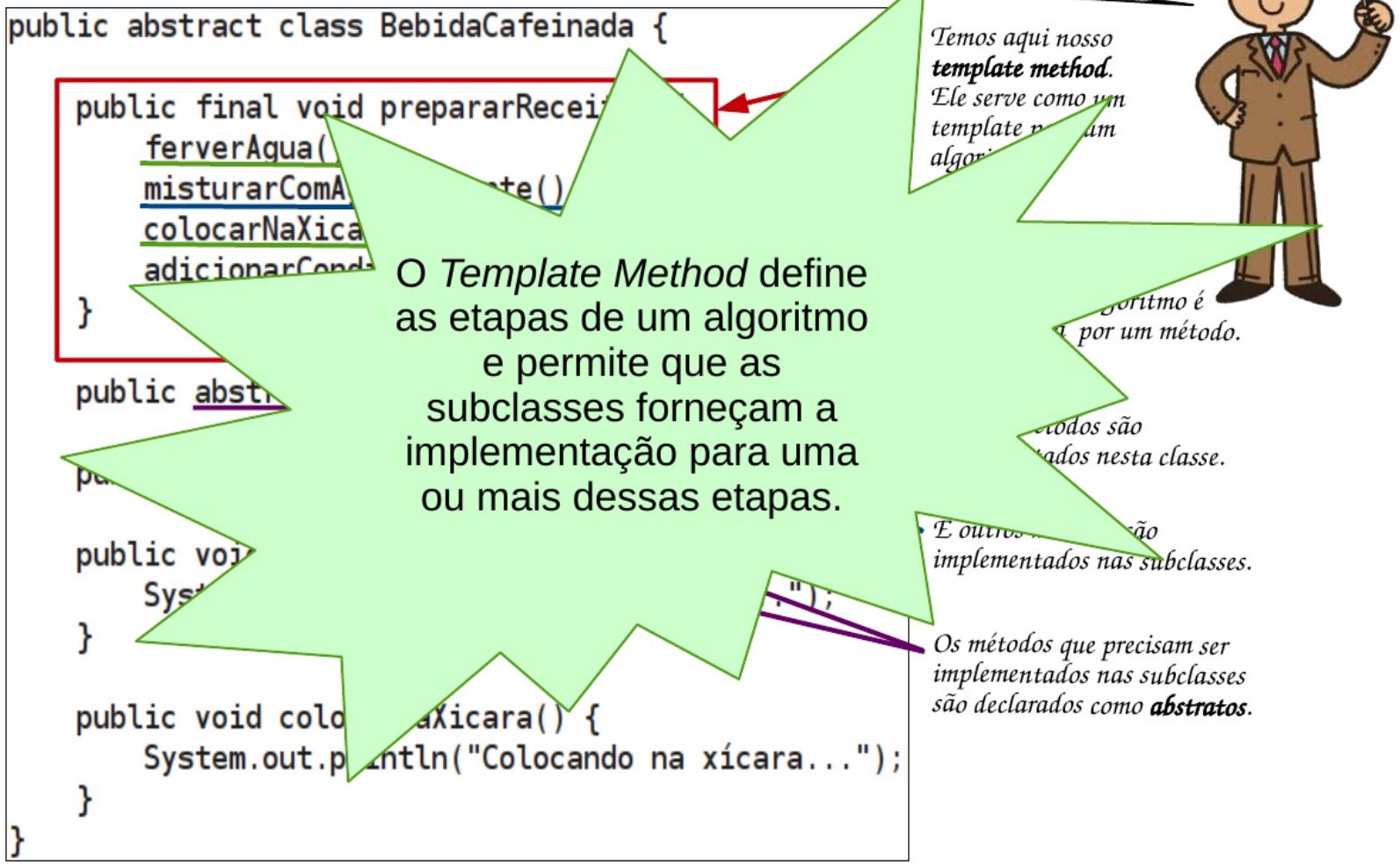
Alguns métodos são implementados nesta classe.

E outros métodos são implementados nas subclasses.

Os métodos que precisam ser implementados nas subclasses são declarados como ***abstratos****.*

Padrão de Projeto *Template Method*

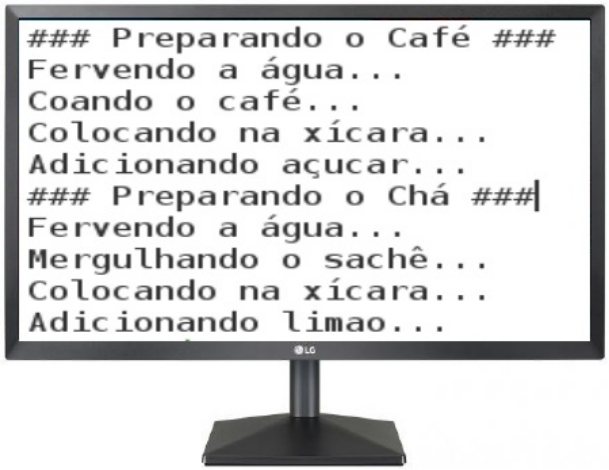
Analisando a estrutura da classe BebidaCafeinada ...



Padrão de Projeto *Template Method*

Afinal, como iremos preparar o café e o chá?

```
public class TemplateMetodo1 {

    public static void main(String[] args) {

        BebidaCafeinada cafe = new Cafe();
        BebidaCafeinada cha = new Cha();

        System.out.println("### Preparando o Café ###");
        cafe.prepararReceita();

        System.out.println("### Preparando o Chá ###");
        cha.prepararReceita();
    }

}
```



Padrão de Projeto *Template Method*

Quais as consequências do uso desse padrão de projeto?

SEM Uso do Padrão de Projeto	*COM Uso do Padrão de Projeto*
	
As classes Cafe e Cha controlam o algoritmo.	*A classe BebidaCafeinada controla o algoritmo e o protege.*
Código duplicado nas classes Cafe e Cha.	*A classe BebidaCafeinada possibilita a reutilização de código.*
Alterações no código do algoritmo de preparação das bebidas requerem mudanças em múltiplas classes.	*Como o algoritmo de preparação das bebidas está implementado em uma única classe, as mudanças ocorrerão apenas nela.*
As classes estão organizadas de modo que muito trabalho é necessário para inclusão de novas bebidas cafeinadas.	*A utilização do padrão de projeto fornece um framework que facilita a inclusão de novas bebidas cafeinadas.*
O conhecimento do algoritmo e sua implementação é distribuída entre múltiplas classes.	*O conhecimento do algoritmo e sua implementação é concentrada em uma única classe, a qual faz uso de subclasses para completar a implementação.*

UFLA
UNIVERSIDADE FEDERAL DE LAVRAS

Estrutura do Padrão de Projeto *Template Method*