Computação Orientada a Objetos

Padrões de Projeto
Template e Iterator

Slides baseados em:
-E. Gamma and R. Helm and R. Johnson and J. Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
-E. Freeman and E. Freeman. Padrões de Projetos. Use a cabeça. Alta Books Editora. 2009.
- Slides Prof. Christian Danniel Paz Trillo
- Slides Profa. Patrícia R. Oliveira

Profa. Karina Valdivia Delgado
EACH-USP

PADRÃO TEMPLATE METHOD

- Padrão Comportamental.
- Objetivo:
  - Definir 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.

PADRÃO TEMPLATE METHOD EXEMPLO

- Motivação: Considere o seguinte:
  - Receita de Café
    - Ferver um pouco de água
    - Misturar o café na água fervente
    - Servir o café na xícara
    - Acrescentar açúcar e leite
  - Receita de Chá
    - Ferver um pouco de água
    - Colocaro chá em infusão na água fervente
    - Despejar o chá na xícara
    - Acrescentar limão
  - Você foi escolhido para implementar esse sistema

PADRÃO TEMPLATE METHOD EXEMPLO

- Poderiamos criar duas classes Coffe e Tea com dois métodos diferentes prepareRecipe()
- Problemas:
  - existe duplicação de código
  - agregar mais uma nova bebida resulta em mais código duplicado

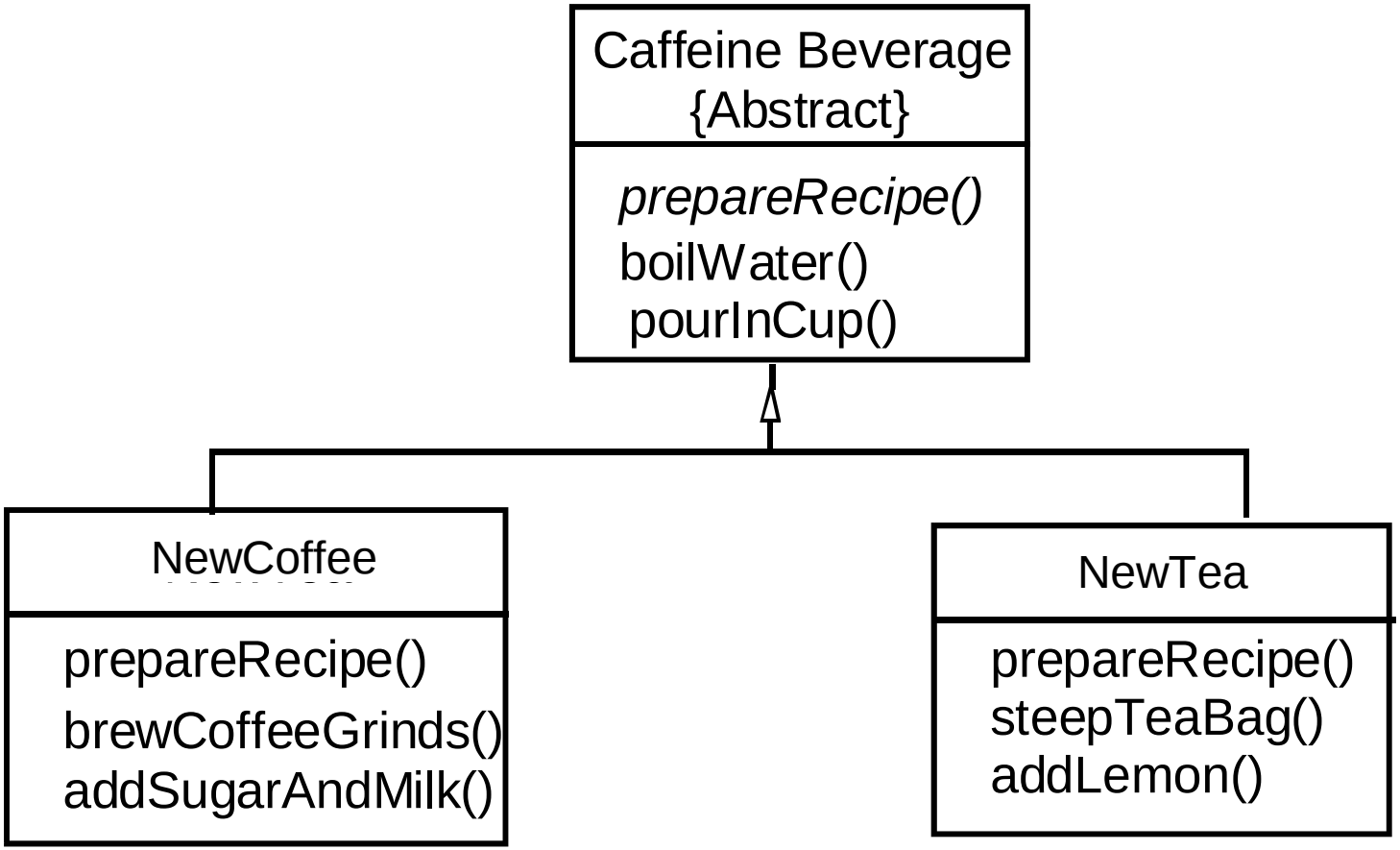
PADRÃO TEMPLATE METHOD

- Receita de Café
  - Ferver um pouco de água
  - Misturar o café na água fervente
  - Servir o café na xícara
  - Acrescentar açúcar e leite
- Receita de Chá
  - Ferver um pouco de água
  - Colocaro chá em infusão na água fervente
  - Despejar o chá na xícara
  - Acrescentar limão

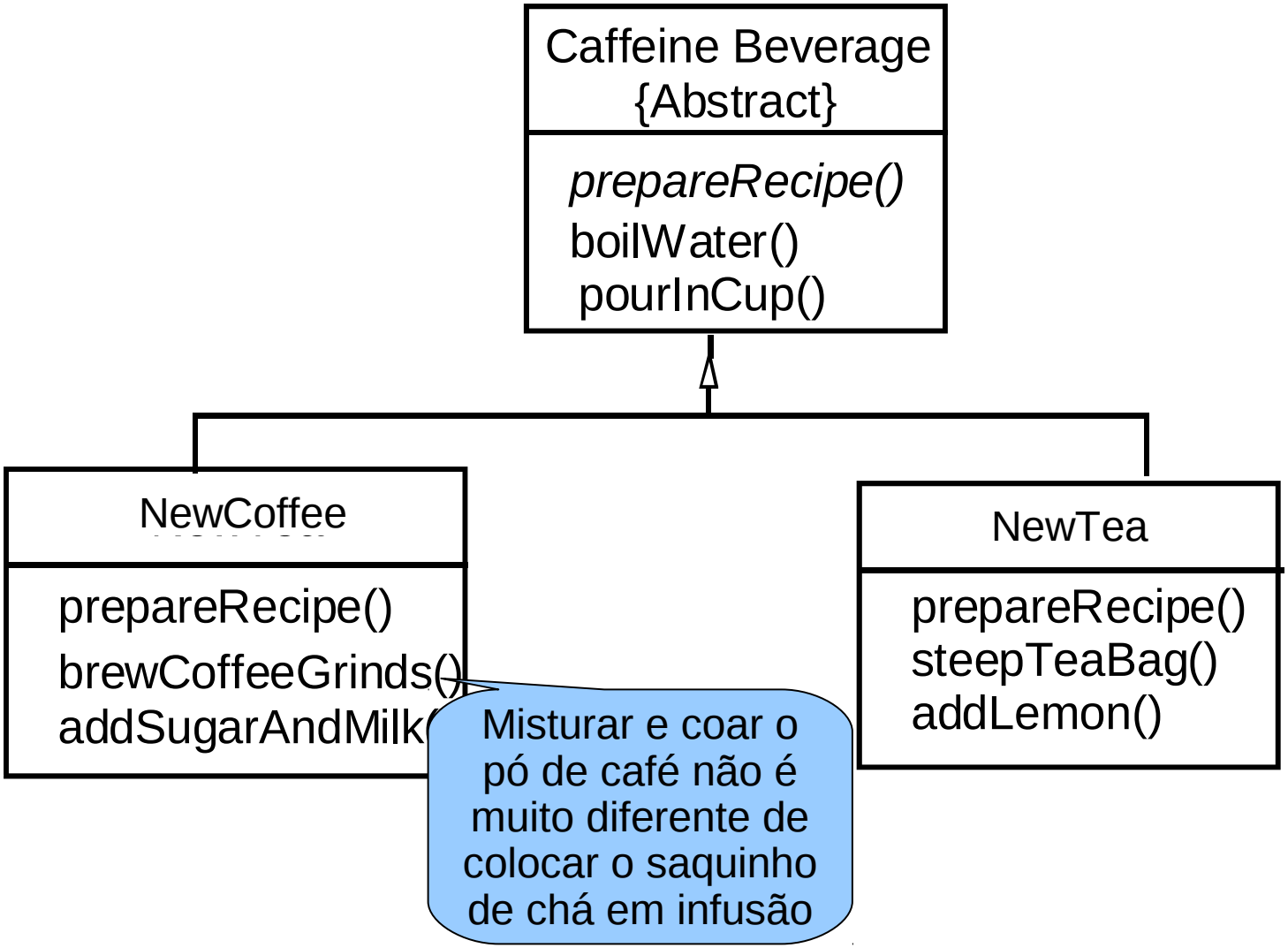
PADRÃO TEMPLATE METHOD - Motivação

- Poderiamos abstrair os pontos em comum usando uma classe abstrata e duas subclasses
- As subclasses herdam o algoritmo genérico
- Alguns métodos no algoritmo são concretos (métodos que executam a mesma ação para todas as subclasses)
- Outros métodos no algoritmo executam ações específicas da subclasse

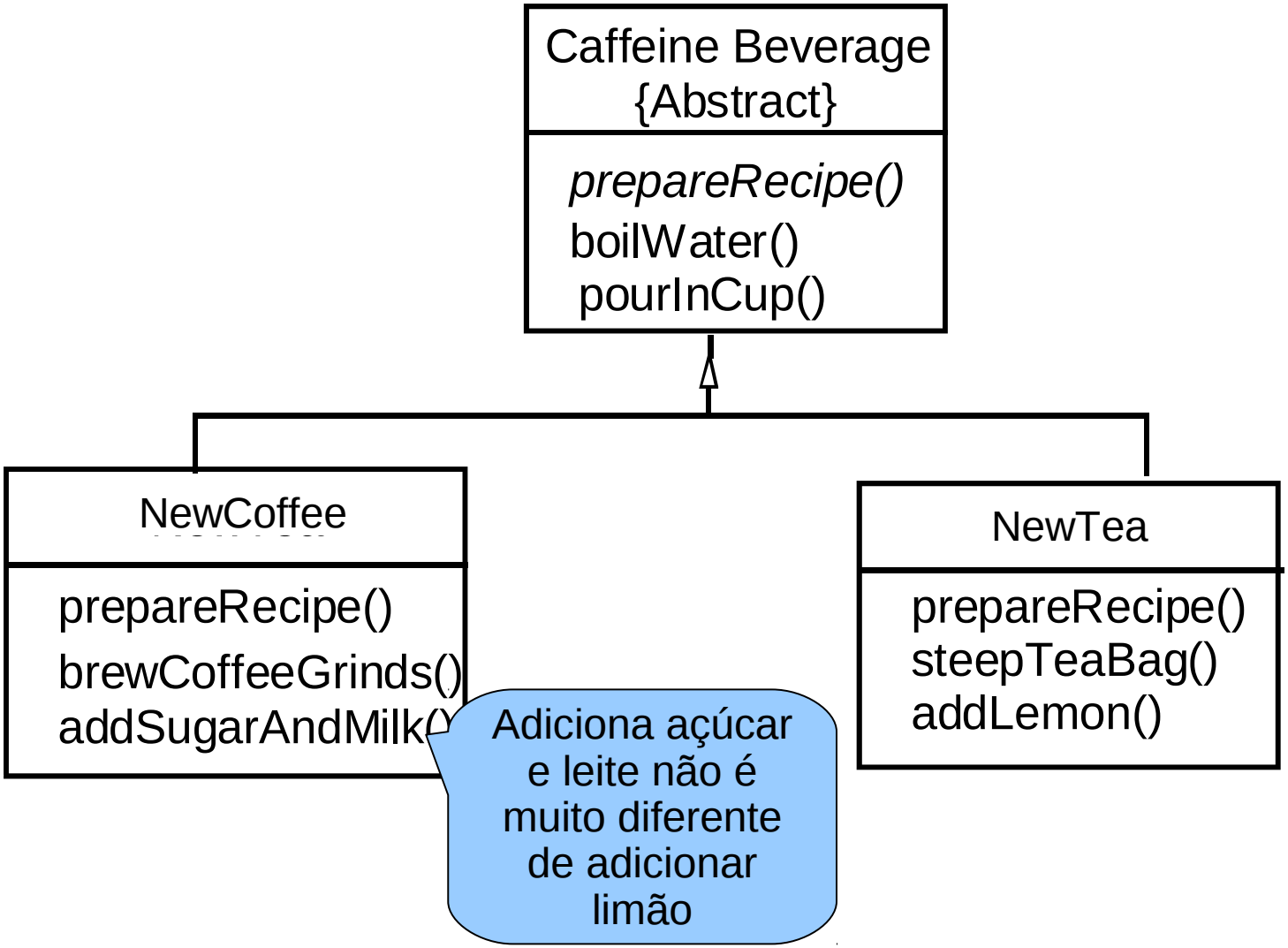
F EMPLATE METHOD - Motivação



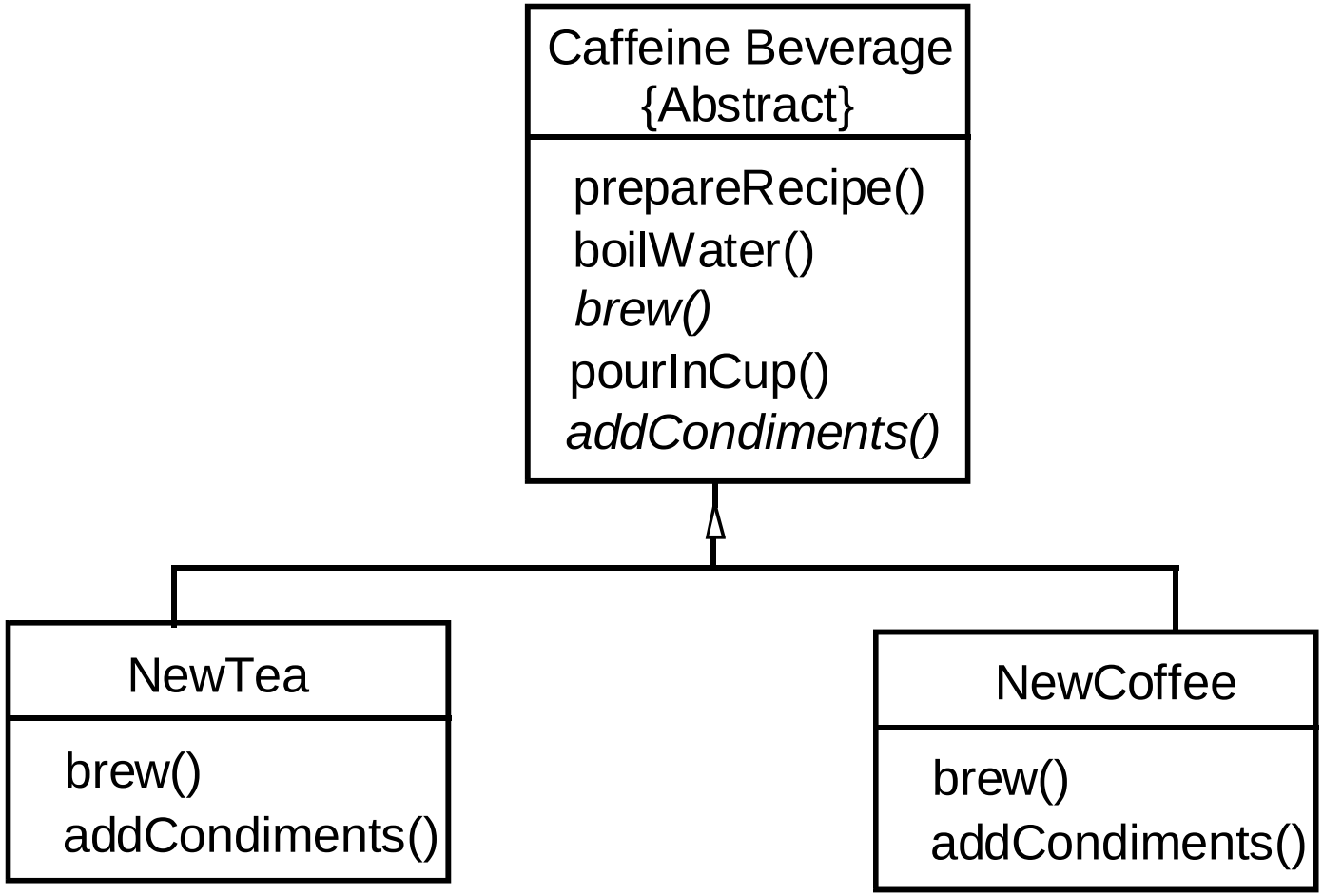
F EMPLATE METHOD - Motivação



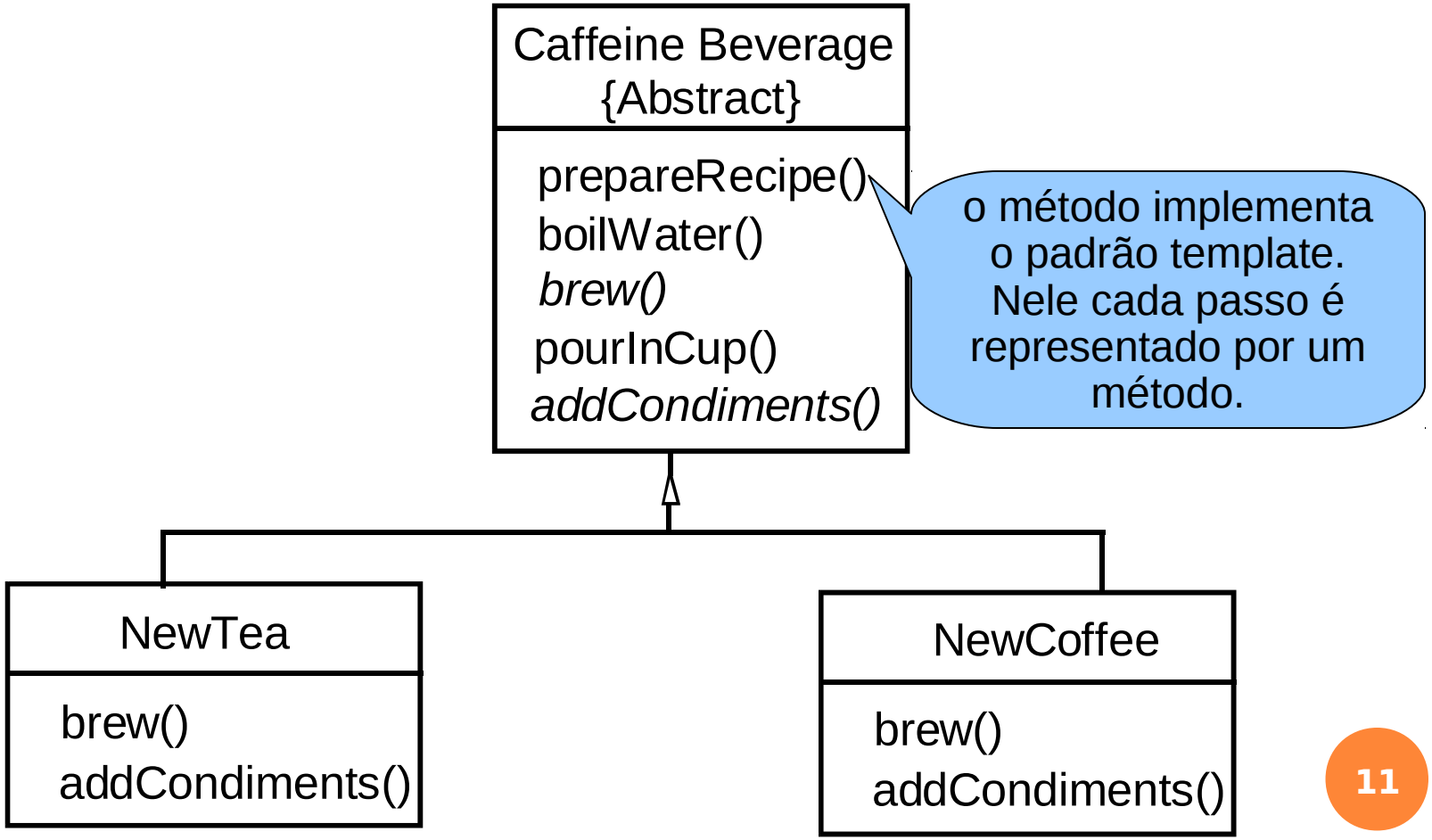
F EMPLATE METHOD - Motivação



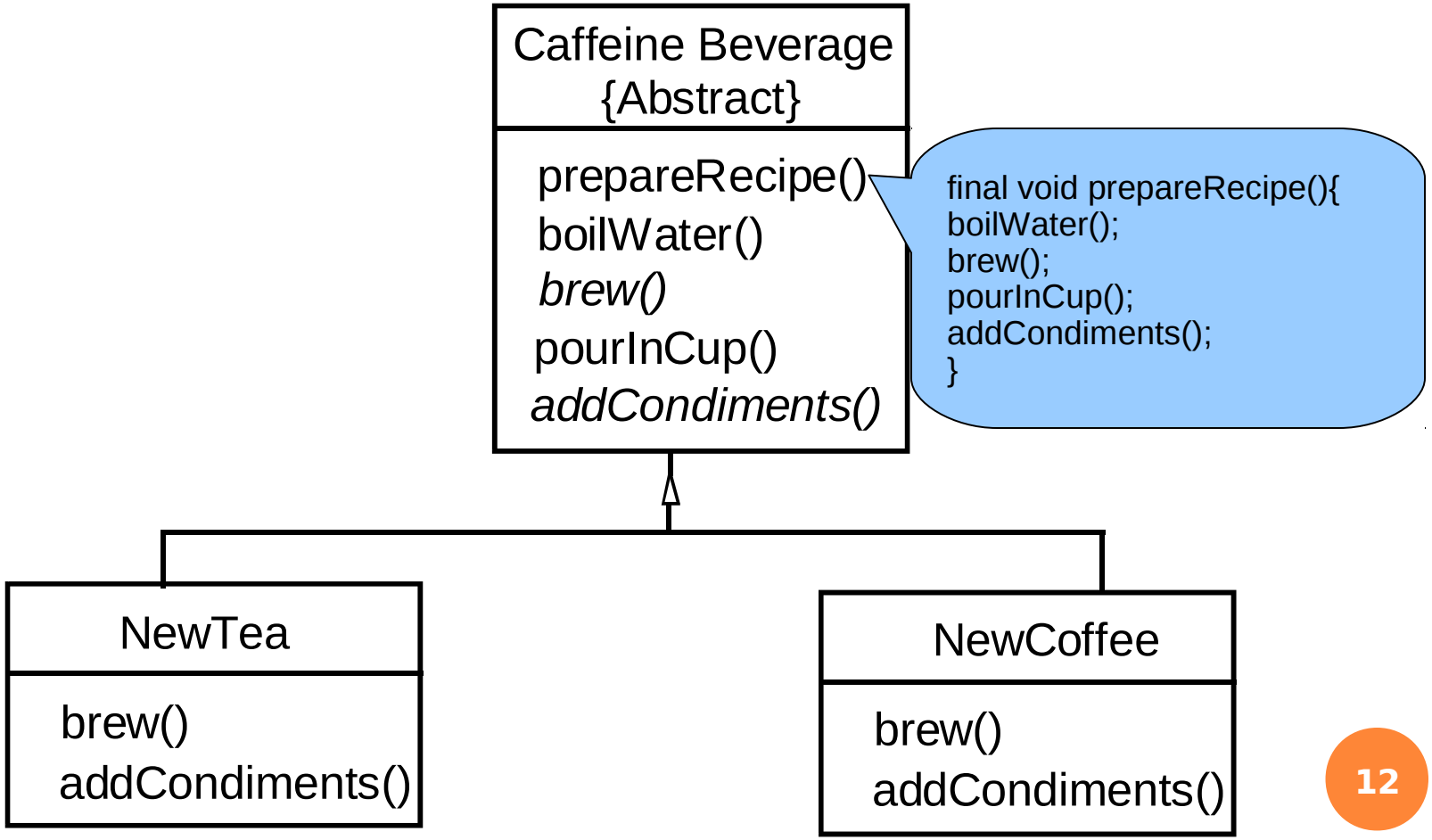
PADRÃO TEMPLATE METHOD - Motivação



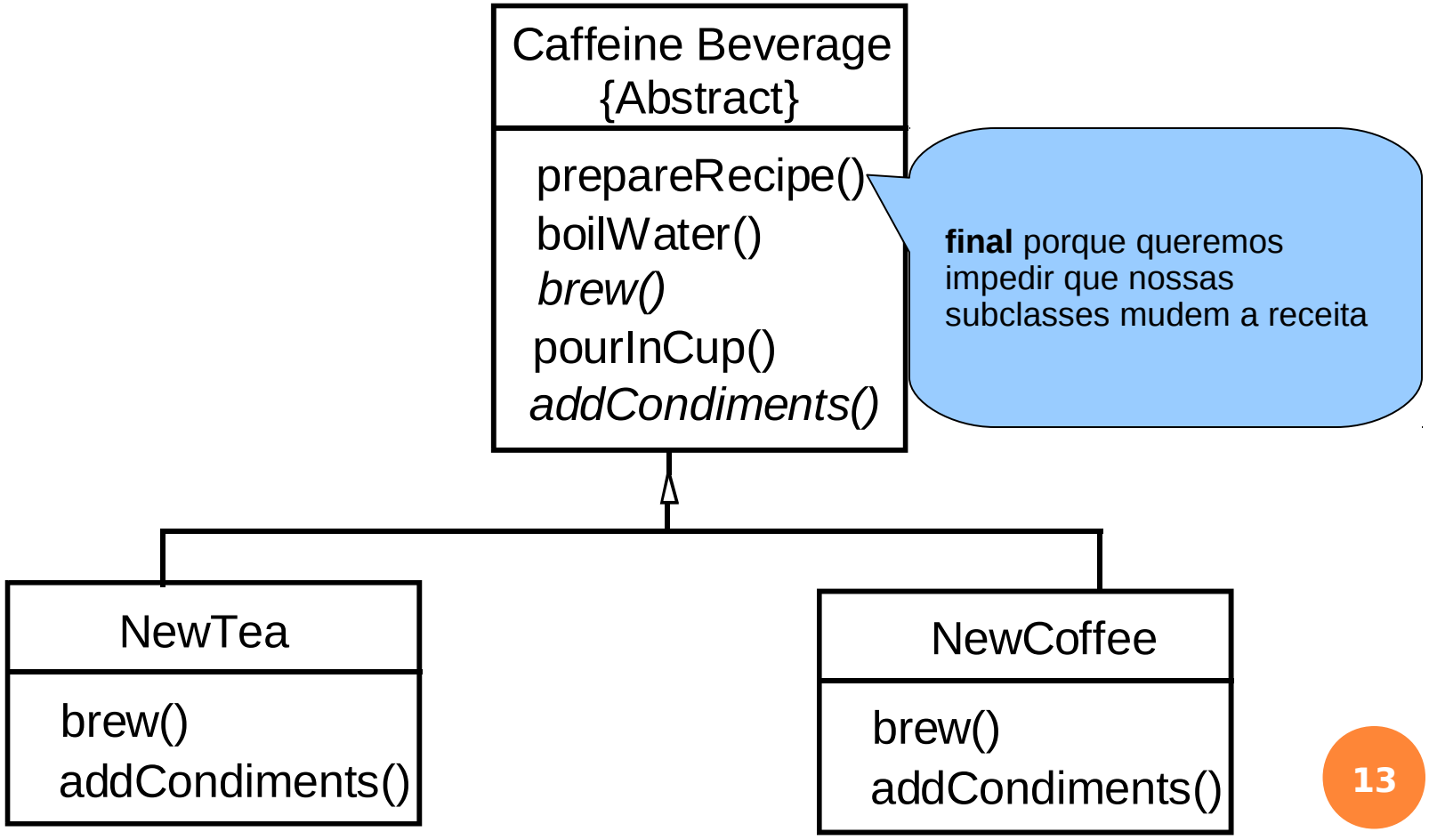
PADRÃO TEMPLATE METHOD - Motivação



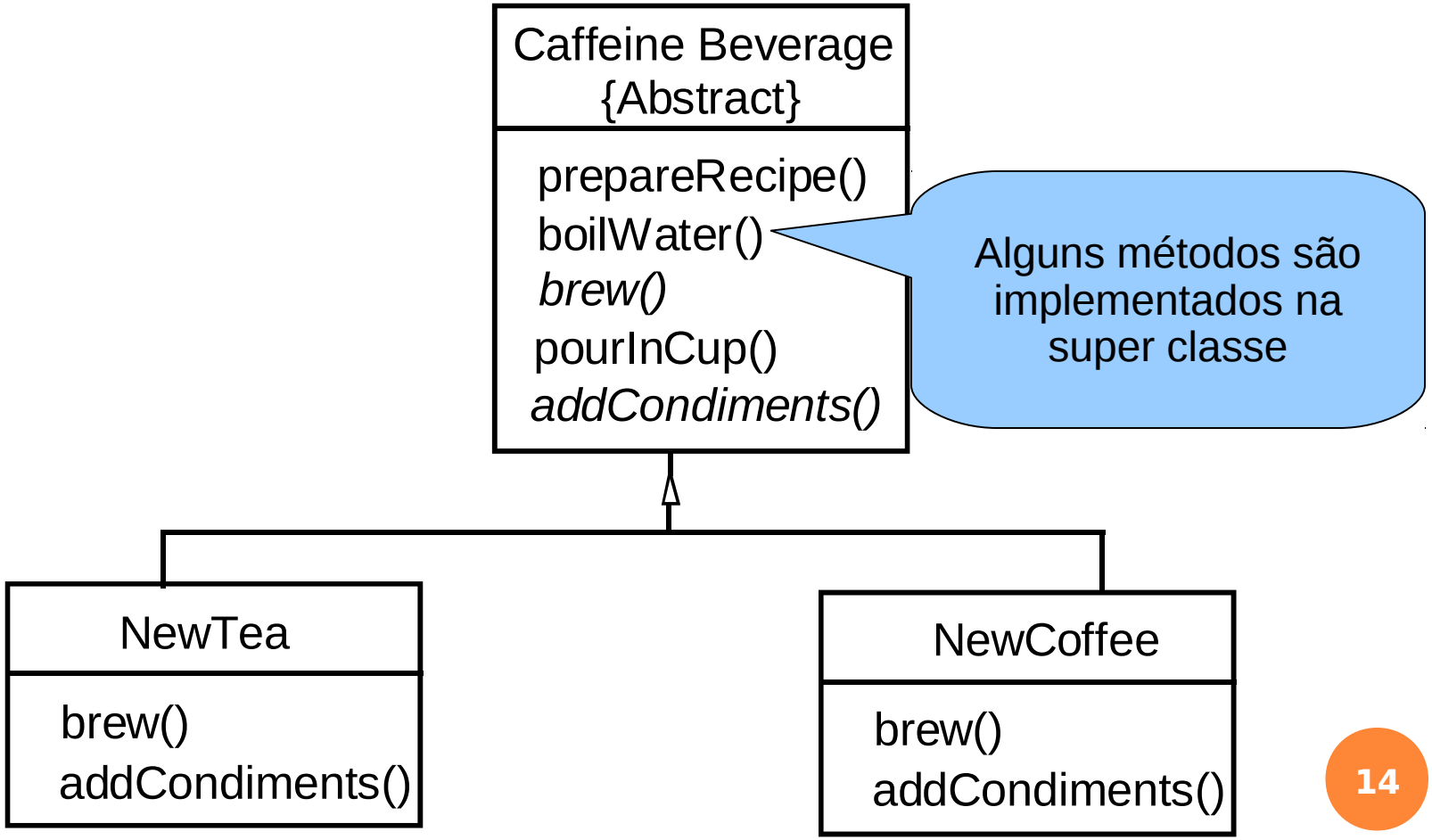
PADRÃO TEMPLATE METHOD - Motivação



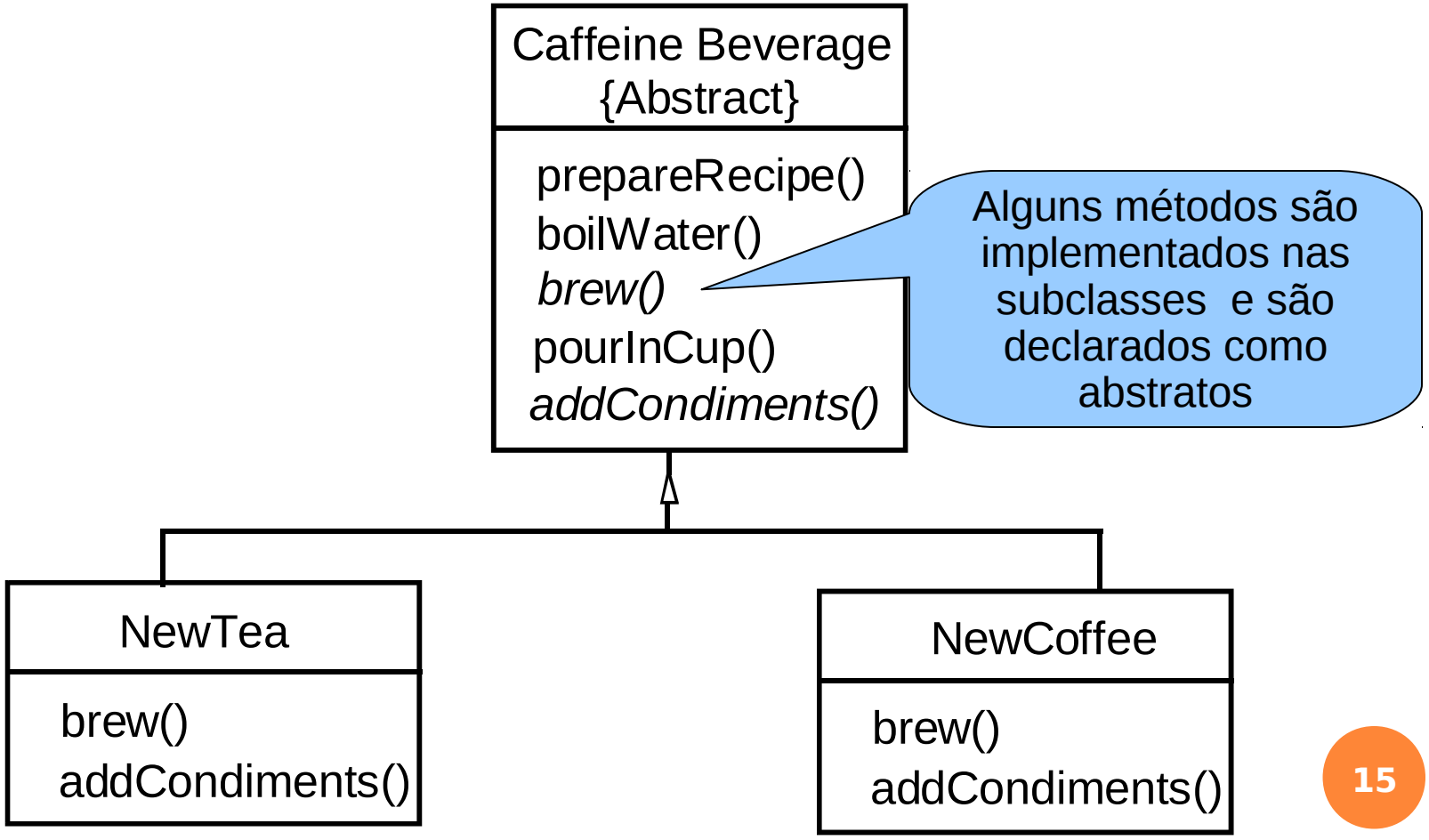
PADRÃO TEMPLATE METHOD - Motivação



PADRÃO TEMPLATE METHOD - Motivação



PADRÃO TEMPLATE METHOD - Motivação



PADRÃO TEMPLATE METHOD - Motivação

- A classe CaffeineBeverage é dona do algoritmo, ela concentra todos os conhecimentos sobre o algoritmo, dependendo das subclasses apenas para o fornecimento das implementações detalhadas.
- Mudanças de código só precisam ser feitas em um único lugar, uma vez que o algoritmo reside em um único lugar.
- Outras bebidas pode ser facilmente adicionadas.

PADRÃO TEMPLATE METHOD - Motivação

- Outro exemplo:
  - Considere um método que adiciona um objeto numa lista serializada.
  - Não importando qual o método de serialização, o processo é o mesmo:
    - Desserializa a lista.
    - Busca se o objeto já está na lista.
    - Se não está, o adiciona à lista.
    - Serializa a lista.

PADRÃO TEMPLATE METHOD - Motivação

- Exemplo:

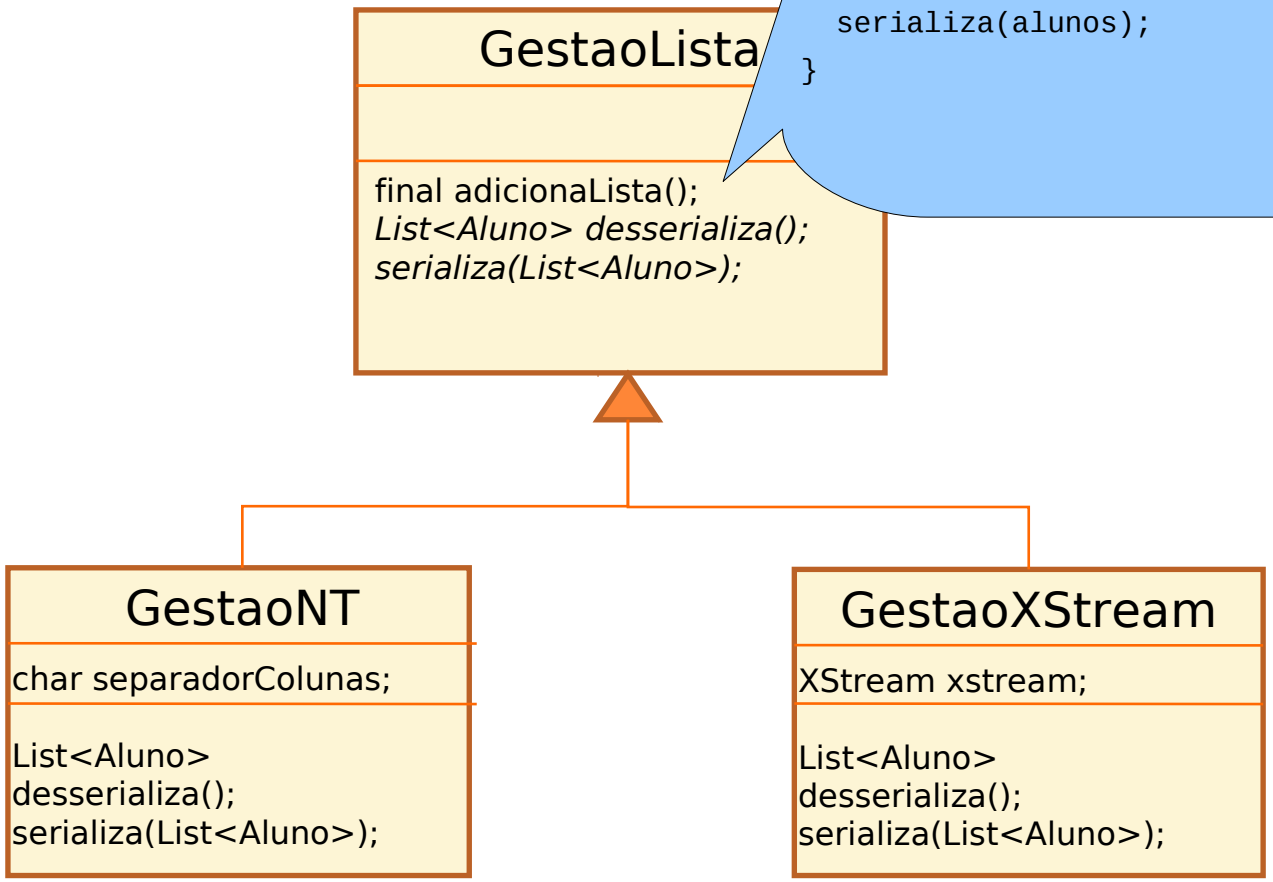
```
List<Aluno> alunos = desserializar();
if (alunos.contains(novoAluno.id)==false)
   alunos.add(novoAluno)
serializar(alunos);
```

- Se tivéssemos vários métodos para serializar:
  - XStream.
  - Arquivos de texto normais.
  - Java Serialization.

PADRÃO TEMPLATE METHOD - Motivação

- Exemplo

```
final adicionaLista()
  List<Aluno> alunos = desserializa();
  if (alunos.contains(novoAluno.id)==false)
       alunos.add(novoAluno)
  serializa(alunos);
}
```

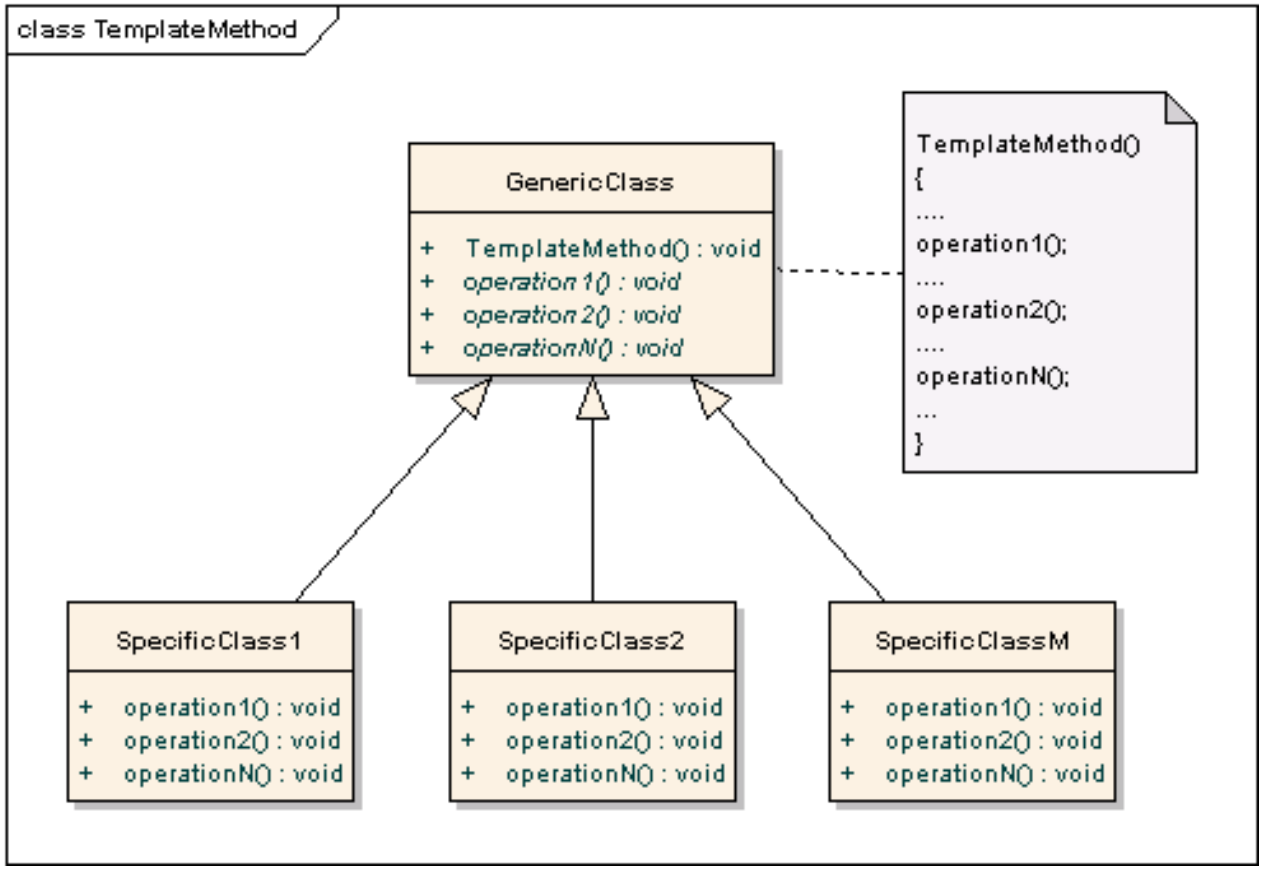


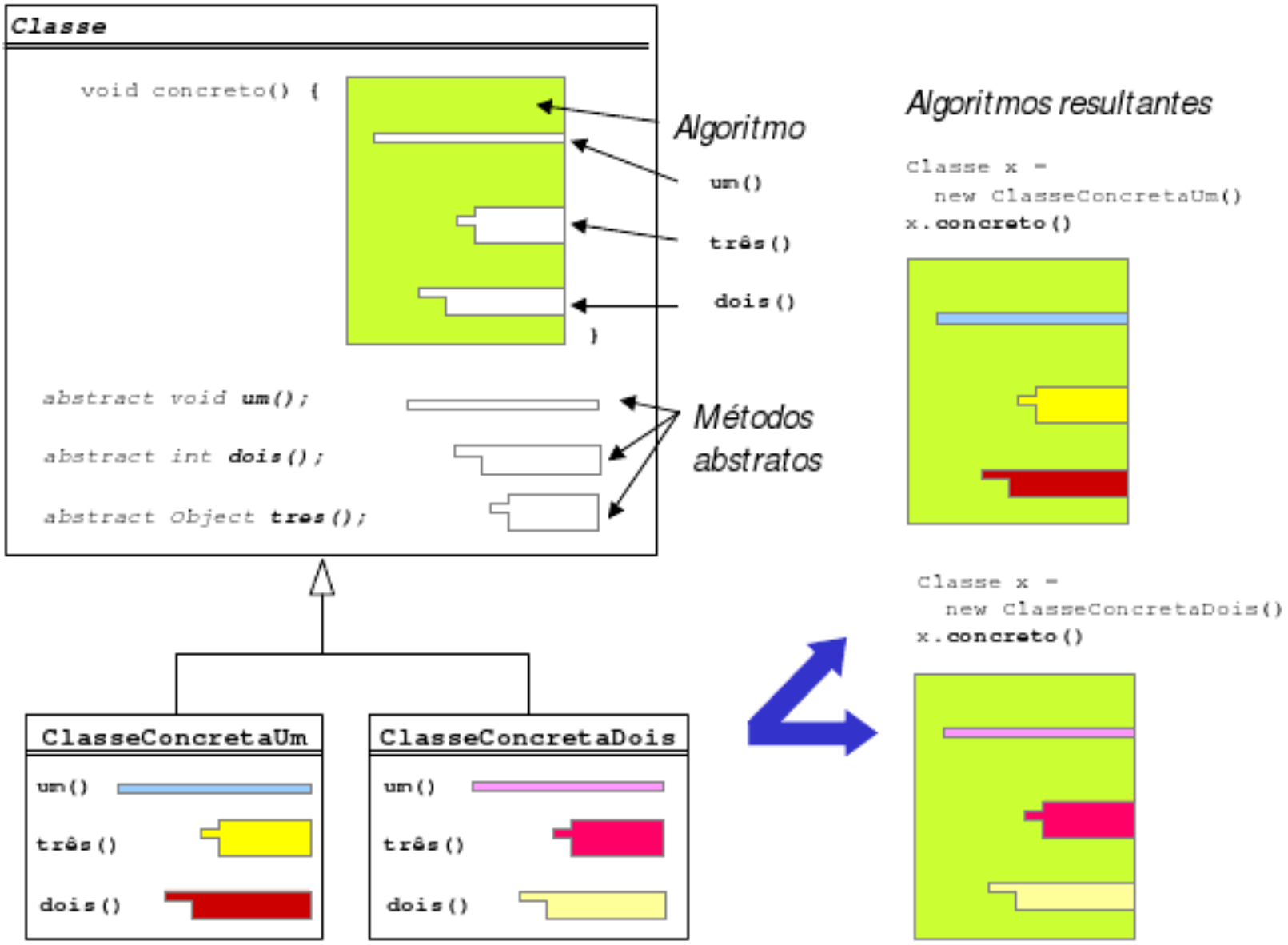
PADRÃO TEMPLATE METHOD

- Aplicabilidade:
  - Implementar as partes invariantes de um algoritmo uma só vez. Deixar para subclasses a implementação do comportamento que pode variar.
  - O comportamento comum entre subclasses deve ser concentrado em uma classe comum para evitar a duplicação de código.
  - Controlar os pontos nos quais as subclasses podem estender o comportamento da classe pai. Estes pontos são chamados métodos Hook (Gancho).

PADRÃO TEMPLATE METHOD

- Estrutura:





- Fonte: http://www.**argonavis**.com.br/cursos/java/j930/J930_05.pdf

PADRÃO TEMPLATE METHOD

- Participantes:
  - GenericClass: Classe abstrata.
    - Define operações primitivas abstratas a serem concretizadas.
    - Implementa (pelo menos) um método “template” que define o esqueleto do algoritmo. Este método deve ser declarado como final (em Java) para evitar que as sub-classes o modifiquem.
  - SpecificClass
    - Implementa as operações primitivas para executarem os passos específicos do algoritmo.
    - Não sobre-escrevem o método template.

PADRÃO TEMPLATE METHOD

- Consequências:
  - Reutilização de Código.
  - Princípio de Hollywood
    - "Don't call us, we'll call you"

    A classe mãe chama as operações de uma subclasse, e não o contrário.
  - Permite estender o comportamento apenas aonde há hooks.

PADRÃO TEMPLATE METHOD

- Implementação:
  - É importante minimizar o número de operações abstratas que devem sofrer override para completar o algoritmo
    - Motivo: simplificação para não chatear o programador
  - O template method deve ser final para evitar que seja sobre-escrito.

PADRÃO TEMPLATE METHOD: USANDO GANCHOS EXEMPLO

- Desejamos minimizar o número de métodos abstratos usados
- Os passos opcionais de um algoritmo sempre podem ser implementados como ganchos
- Ganchos são métodos que não fazem nada, ou executam apenas um comportamento default na classe abstrata, mas podem ser substituídos na subclasse.

Ex: o cliente pode querer ou não um condimento na bebida

PADRÃO TEMPLATE METHOD: USANDO GANCHOS EXEMPLO

```
public abstract class CaffeineBeverage
{
   final void prepareRecipe()
   {
      boilWater();
      brew();
      pourInCup();
      if (customerWantsCondiments(){
          addCondiments();
      }
   }
   abstract void brew();
   abstract void addCondiments();
   void boilWater()
   {
      //Implementation
   }
   void pourInCup()
   {
     //Implementation
   }
   boolean customerWantsCondiments(){ return true;}
}
```

PADRÃO TEMPLATE METHOD: USANDO GANCHOS EXEMPLO

```
public abstract class CaffeineBeverage
{
   final void prepareRecipe()
   {
      boilWater();
      brew();
      pourInCup();
      if (customerWantsCondiments(){
          addCondiments();
      }
   }
   abstract void brew();
   abstract void addCondiments();
   void boilWater()
   {
      //Implementation
   }
   void pourInCup()
   {
     //Implementation
   }
   boolean customerWantsCondiments(){ return true;}
}
```

Esse método é um gancho, as subclasses podem alterar esse método

PADRÃO TEMPLATE METHOD: RESUMO

- Don't call us we'll call you
- O padrão Template define o conjunto de passos de um algoritmo
- Subclasses não podem mudar o algoritmo (final)
- Facilita o reuso de código

PADRÃO ITERATOR

PADRÃO ITERATOR – Objetivo

- Padrão Comportamental.
- Fornece um meio de acessar sequencialmente um objeto agregrado (por ex, uma coleção) sem expor a sua representação.

PADRÃO ITERATOR - Motivação

- Pode-se desejar percorrer uma estrutura (por ex., uma lista) de maneiras diferentes, dependendo da tarefa a ser executada.
- Não é adequado “inchar”, por ex., uma suposta classe Lista com várias operações, uma para cada tipo de travessia.

PADRÃO ITERATOR - Motivação

- Exemplo:

  •Uma panquecaria e um restaurante estão fusionándo-se. Eles querem usar o cardápio da Panquecaria no café da manhã e o cardápio do Restaurante no almoço. O problema é que os itens de menú tem sido armacenados em um ArrayList e em um vetor. Nenhum dos proprietários desejam mudar suas implementações.

PADRÃO ITERATOR - Motivação

- Imagine que você tenha sido contratado pela nova companhia, formada pela fusão da Panquecaria com o Restaurante e precisa imprimir todos os items dos menús.

PADRÃO ITERATOR - Motivação

- Recuperara os itens dos cardápios:

```
PancakeHouseMenu pancakeHouseMenu= new PancakeHouseMenu();
ArrayList breakfastItems = pancakeHouseMenu.getMenuItems();
DinerMenu dinerMenu = new DinerMenu();
MenuItems[] lunchItems = dinerMenu.getMenuItems():
```

- Percorrendo os itens do menu do café da manhã:

```
for(int i=0; i < breakfastItems.size(); ++i)
{
  MenuItem menuItem=
        (MenuItem) breakfastItems.get(i);
}
```

- Percorrendo os itens do almoço:

```
for(int i=0; I < lunchItems.length; i++)
{
    MenuItem menuItem = lunchItems[i];
}
```

PADRÃO ITERATOR - Motivação

- Dois laços são necessários ao invés de um.
- Se um terceiro restaurante é incluído, tres laços seriam necessários.
- Princípios violados:
  - Estamos codificando para implementação ao invés da interface.
  - O método implementado precisa conhecer a estrutura interna das coleções
  - Código duplicado.

PADRÃO ITERATOR - Motivação

- Devemos encapsular tudo aquilo que varia, i.e., encapsular a iteração
- A classe DinerMenu e a classe PancakeMenu precisam implementar um método chamado createIterator().
- O Iterator é usado para acessar os elementos de cada coleção sem saber o seu tipo (ArrayList ou vetor)

PADRÃO ITERATOR - Motivação

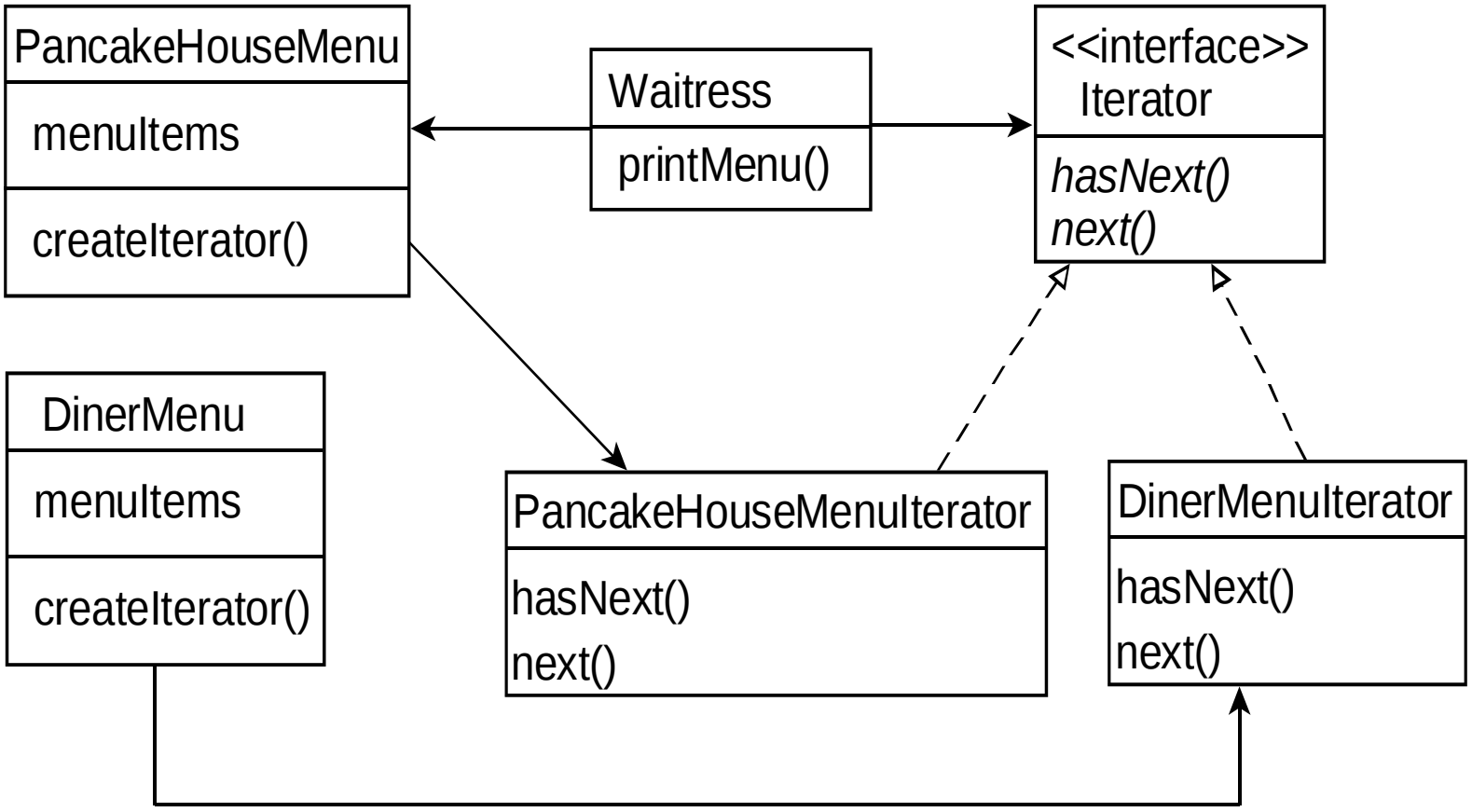
- Percorrendo os itens do menu do café da manhã:

```
Iterator iterator = breakfastMenu.createIterator();
while(iterator.hasNext())
{
    MenuItem menuItem = (MenuItem)iterator.next();
}
```

- Percorrendo os itens do almoço:

```
Iterator iterator = lunchMenu.createIterator();
while(iterator.hasNext())
{
    MenuItem menuItem = (MenuItem)iterator.next();
}
```

PADRÃO ITERATOR - Motivação primeira solução



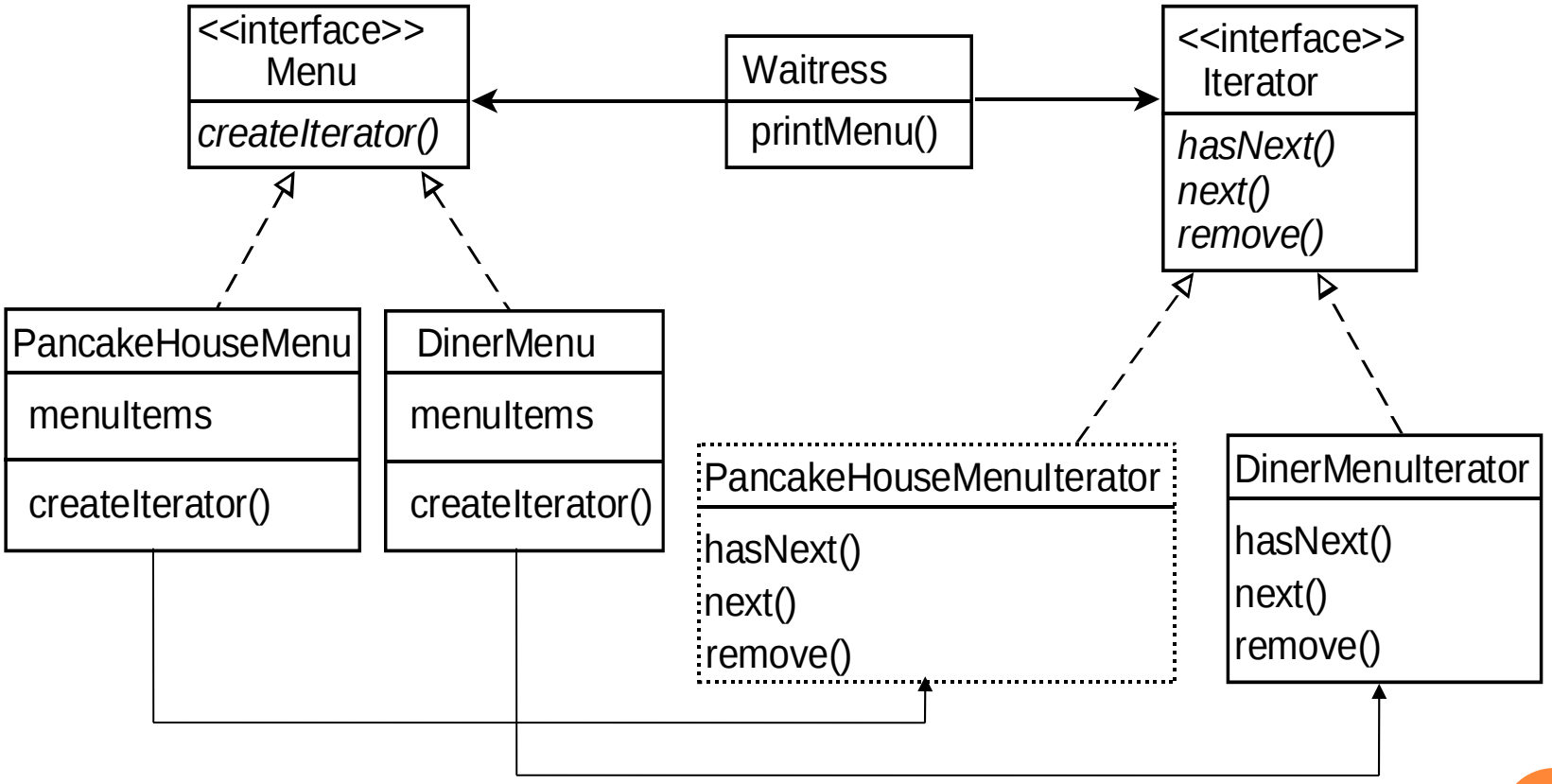
PADRÃO ITERATOR - Motivação

- Por que não estamos usando a interface Iterator de Java?
- Ela é mais poderosa do que a nossa interface Iterator caseira ...
- A interface Iterator de Java tem os seguintes métodos:
  - hasNext()
  - next()
  - remove()
- O método remove é opcional. Se não quiser permitir a presença de remove(), você terá que incorporar a exceção UnsupportedOperationException

PADRÃO ITERATOR - Motivação

- Mudando o código para usar java.util.iterator:
  - Apagar a classe PancakeHouseMenuIterator uma vez que a classe ArrayList tem um método que devolve um Java Iterator.
  - Mudar a classe DinerMenuIterator para implementar o Java Iterator
- Um outro problema: todos os menus devem ter a mesma interface.
  - Incluir a interface Menu

PADRÃO ITERATOR - Motivação



PADRÃO ITERATOR - Motivação

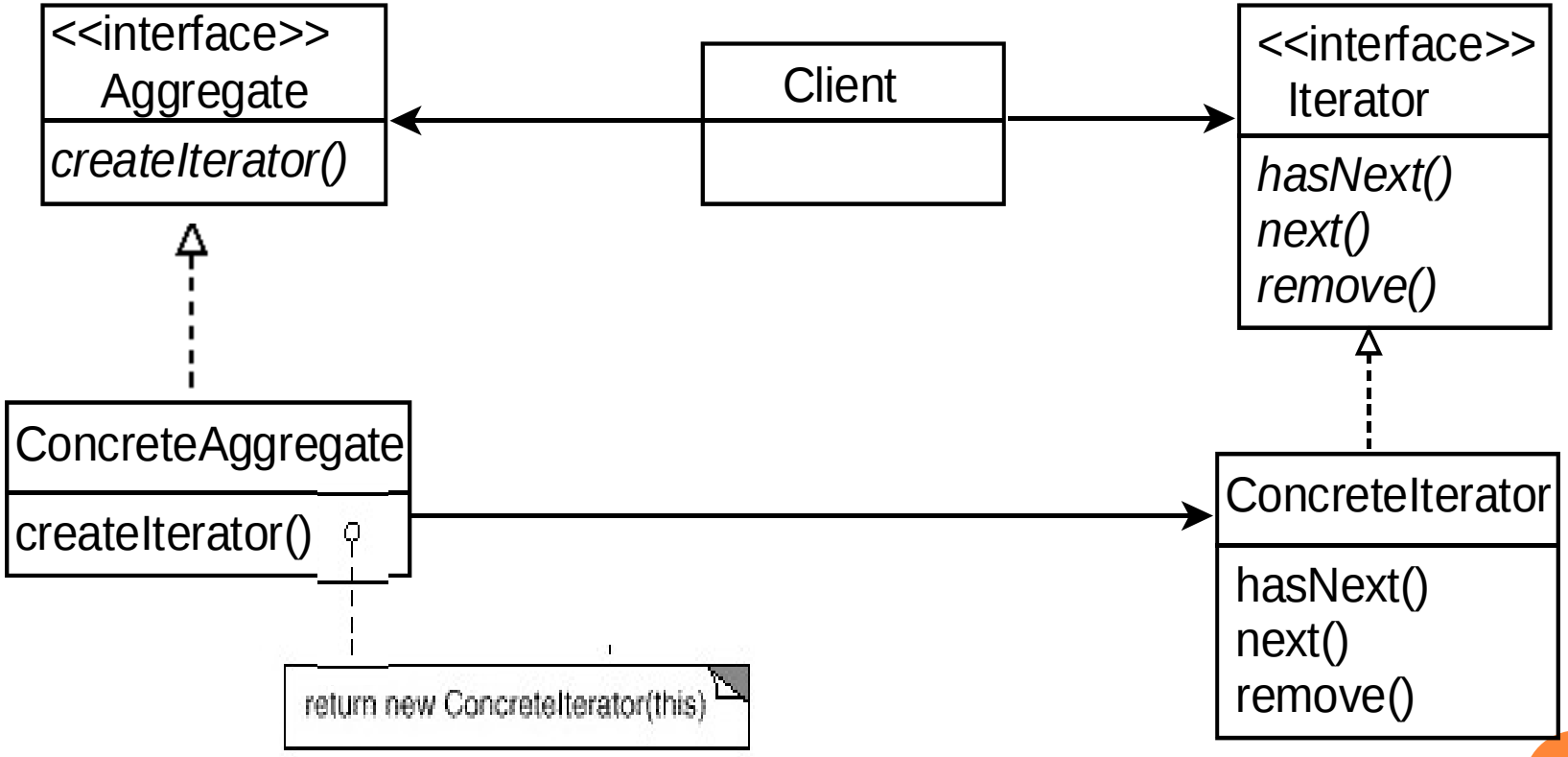
```
public class DinerMenuIterator implements Iterator {
  MenuItem[] items;
  int position=0;
  public DinerMenuIterator(MenuItem[] items){
     //Implementation
  }
   public Object next()   {
     MenuItem menuItem=items[position];
     position=position + 1;
     return menuItem;
   }
  public boolean hasNext(){
   if(position >=items.lengh||items[position]==null){
     return false;}
   else{
     return true;}
   }
  public void remove(){
     //Implementation
  }
}
```

implementar o método

PADRÃO ITERATOR - Motivação

- O padrão Iterator permite que o implementador forneça uma interface uniforme para percorrer vários tipos de objetos agregados.
- A ideia chave é
  - retirar do objeto agregado as responsabilidades de acesso e percurso e
  - delegá-las a um objeto Iterator que definirá um protocolo de percurso.

Padrão Iterator - Estrutura

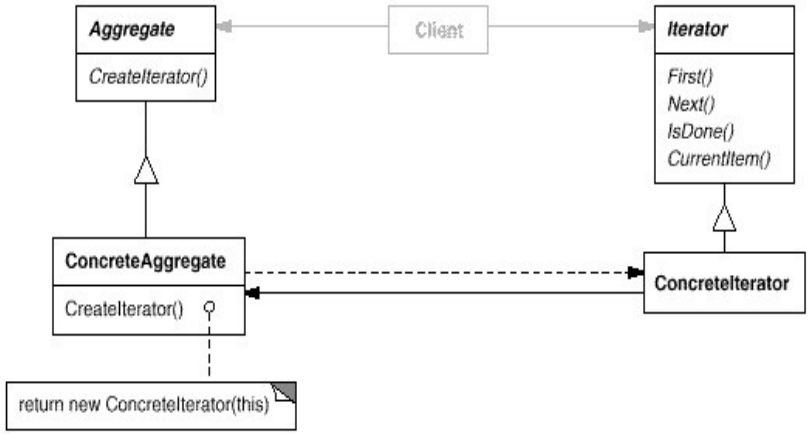


Padrão Iterator

- Participantes:
  - Iterator:
    - Define uma interface para acessar e percorrer elementos
  - ConcreteIterator
    - Implementa a interface de Iterador
    - Mantém o controle da posição corrente no percurso do agregado.
  - Aggregate
    - Define uma interface para a criação de um objeto Iterator
  - ConcreteAggregate
    - Implementa a interface de criação do Iterator para retornar uma instância do ConcreteIterator apropriado

PADRÃO ITERATOR – Algumas informações

- Os nomes clássicos de métodos adotados para o padrão Iterator são:
  - first()
  - next()
  - isDone()
  - currenItem()



- A interface java.util.Enumeration é uma implementação mais antiga e foi substituída por java.util.Iterator.
- Um iterador pode avançar ou retroceder

Padrão Iterator – Exercício

- Incorpore um novo cardápio CafeMenu ao sistema. O novo cardápio está armazenado numa HashMap, que tem como chave o nome do MenuItem e como valor o preço.

Mudanças

- A classe *CafeMenu* deve implementar a interface *Menu*.
- Apagar o método *getItems()* da classe *CafeMenu*.
- Agregar um método *createIterator ()* na classe *CafeMenu.*
- Mudar a classe *Waitress*
  - Declarar uma instância of *Menu* para *CafeMenu*.
  - Mudar o método *printMenu() para pegar o* iterator da classe *CafeMenu* e imprimir o menu.

Problemas com o código?

```
public class Waitress{
Menu pancakeHouseMenu;
Menu dinerMenu;
Menu cafeMenu;
//implementar construtor
public void printMenu()
{
  Iterator pancakeIterator =
              pancakeHouseMenu.createIterator();
  Iterator dinerIterator = dinerMenu.createIterator();
  Iterator cafeIterator = cafeMenu.createIterator();

  System.out.println("MENU\n----\nBREAKFAST");
  printMenu(pancakeIterator);
  System.out.println("\nLUNCH");
  printMenu(dinerIterator);
  System.out.println(“\nDINNER”);
  printMenu(cafeIterator);
}
private void printMenu(Iterator iterator){
\\implementation
}
}
```

Problemas com o código?

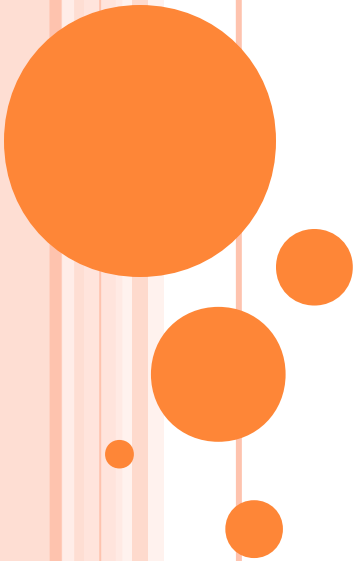
```
public class Waitress{
Menu pancakeHouseMenu;
Menu dinerMenu;
Menu cafeMenu;
//implementar construtor
public void printMenu()
{
  Iterator pancakeIterator =
              pancakeHouseMenu.createIterator();
  Iterator dinerIterator = dinerMenu.createIterator();
  Iterator cafeIterator = cafeMenu.createIterator();

  System.out.print
  printMenu(pancak
  System.out.print
  printMenu(dinerIterator);
  System.out.println("\nDINNER");
  printMenu(cafeIterator);
}
private void printMenu(Iterator iterator){
\\implementation}
}
```

Toda vez que acrescentarmos ou removermos um menu, teremos que abrir este método e modificá-lo

Problemas com o código?

Na classe Waitress podemos criar um ArrayList de menus e fazer com que o iterador dessa lista realize a iteração através de cada menú



Problemas com o código?

```
public class Waitress{
ArrayList<Menu> menus;
//implementar construtor
public void printMenu()
{
  Iterator menuIterator=menus.iterator();
  while(menuIterator.hasNext()){
      Menu menu= menuIterator.next();
      printMenu(menu.createIterator());

   }
}
private void printMenu(Iterator iterator){
\\implementation}
}
```