Programação Genérica

Programação Orientada a Objetos — QXD0007

UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz

gomes.atilio@ufc.br

Universidade Federal do Ceará

1º semestre/2022

Leitura para esta aula

- **Capítulo 20 (Classes e Métodos Genéricos)** do livro Paul Deiltel, Harvey Deitel, Java Como Programar, Décima Edição, Pearson.

- **Capítulo 8 (Generic Programming)** do livro:
  Cay S. Horstmann. *Core Java, Volume I – Fundamentals*. Tenth Edition.

Motivação

Motivação

Analise a classe abaixo.

```
1  // Printing array elements using overloaded methods.
2  public class OverloadedMethods {
3      public static void main(String[] args) {
4          // create arrays of Integer, Double and Character
5          Integer[] integerArray = {1, 2, 3, 4, 5, 6};
6          Double[] doubleArray = {1.1,2.2,3.3,4.4,5.5,6.6,7.7};
7          Character[] characterArray = {'H','E','L','L','O'};
8
9          System.out.printf("Array integerArray contains: ");
10         printArray(integerArray); // pass an Integer array
11         System.out.printf("Array doubleArray contains: ");
12         printArray(doubleArray); // pass a Double array
13         System.out.printf("Array characterArray contains: ");
14         printArray(characterArray); // pass a Character array
15     }
```

Motivação (cont.)


```
16      public static void printArray(Integer[] inputArray) {
17          for (Integer element : inputArray)
18              System.out.printf("%s ", element);
19          System.out.println();
20      }
21
22      public static void printArray(Double[] inputArray) {
23          for (Double element : inputArray)
24              System.out.printf("%s ", element);
25          System.out.println();
26      }
27
28      public static void printArray(Character[] inputArray) {
29          for (Character element : inputArray)
30              System.out.printf("%s ", element);
31          System.out.println();
32      }
33  }
```

Motivação (cont.)

- Problema: note que os métodos sobrecarregados `printArray` da classe `OverloadedMethods` são praticamente idênticos, mudando apenas no tipo de dado do array. Isso vai contra o princípio do reuso.

- Questão: É possível escrever apenas um método `printArray` neste caso?

Motivação (cont.)

- Solução usada antes do Java 5.0: Herança + Object

```
1     public static void printArray(Object[] inputArray) {
2         for (Object element : inputArray)
3             System.out.printf("%s ", element);
4         System.out.println();
5     }
```

Motivação (cont.)

- Solução usada antes do Java 5.0: Herança + Object

```
1     public static void printArray(Object[] inputArray) {
2         for (Object element : inputArray)
3             System.out.printf("%s ", element);
4         System.out.println();
5     }
```

- Solução usada a partir do Java 5.0: Tipos Genéricos

```
1     public static <T> void printArray(T[] inputArray) {
2         for (T element : inputArray)
3             System.out.printf("%s ", element);
4         System.out.println();
5     }
```

Motivação (cont.)

- Solução usada antes do Java 5.0: Herança + Object

```
1     public static void printArray(Object[] inputArray) {
2         for (Object element : inputArray)
3             System.out.printf("%s ", element);
4         System.out.println();
5     }
```

- Solução usada a partir do Java 5.0: Tipos Genéricos

```
1     public static <T> void printArray(T[] inputArray) {
2         for (T element : inputArray)
3             System.out.printf("%s ", element);
4         System.out.println();
5     }
```

Apesar dos métodos acima obterem o mesmo resultado, o uso de tipos genéricos provê benefícios que veremos durante esta aula.

Introdução

Introdução

- Programação Genérica significa escrever código que pode ser reusado por objetos de diferentes tipos.
  - Adicionada ao JDK a partir da versão 5.0

Introdução

- Programação Genérica significa escrever código que pode ser reusado por objetos de diferentes tipos.
  - Adicionada ao JDK a partir da versão 5.0

- Antes dos genéricos:

```
1 ArrayList files = new ArrayList();
2 // Você não sabe que tipo de objeto está
3 // armazenado dentro da lista
4 String filename = (String) files.get(0);
```

- Antes dos genéricos não havia checagem de tipos

- Erros eram encontrados apenas em tempo de execução

Introdução

- Solução dos genéricos: parâmetros de tipo
  - O compilador agora pode checar os tipos

```
1 ArrayList<String> files = new ArrayList<>();
2 // Você não precisa mais da operação de cast
3 String filename = files.get(0);
```

Introdução

- Solução dos genéricos: parâmetros de tipo
  - O compilador agora pode checar os tipos

```
1 ArrayList<String> files = new ArrayList<>();
2 // Você não precisa mais da operação de cast
3 String filename = files.get(0);
```

  - *Genéricos* deixam o código mais seguro e fácil de ler.

Introdução

- Solução dos genéricos: parâmetros de tipo
  - O compilador agora pode checar os tipos

```
1 ArrayList<String> files = new ArrayList<>();
2 // Você não precisa mais da operação de cast
3 String filename = files.get(0);
```

  - *Genéricos* deixam o código mais seguro e fácil de ler.

- Como você usará programação genérica?
  - **Nível básico:** usando classes genéricas.
  - **Nível intermediário:** quando você encontrar seu primeiro erro enigmático usando classes genéricas.
  - **Nível avançado:** implementando suas próprias classes genéricas.

Métodos Genéricos

Métodos Genéricos

- Em Java, podemos definir também métodos genéricos.
  - O parâmetro de tipo é definido logo antes do tipo de retorno do método:

```
1 class ArrayAlg
2 {
3     public static <T> T getMiddle(T[] a)
4     {
5       return a[a.length / 2];
6     }
7 }
```

  - Métodos genéricos podem ser definidos tanto dentro de classes genéricas quanto de classes comuns.

Métodos Genéricos

- Em Java, podemos definir também métodos genéricos.
  - O parâmetro de tipo é definido logo antes do tipo de retorno do método:

```
1 class ArrayAlg
2 {
3     public static <T> T getMiddle(T[] a)
4     {
5       return a[a.length / 2];
6     }
7 }
```

  - Métodos genéricos podem ser definidos tanto dentro de classes genéricas quanto de classes comuns.

- Chamadas do método genérico:
  - `String middle = Arrayalg.<String>getMiddle("John","Q.","Public");`

Métodos Genéricos

- Em Java, podemos definir também métodos genéricos.
  - O parâmetro de tipo é definido logo antes do tipo de retorno do método:

```
1 class ArrayAlg
2 {
3     public static <T> T getMiddle(T[] a)
4     {
5       return a[a.length / 2];
6     }
7 }
```

  - Métodos genéricos podem ser definidos tanto dentro de classes genéricas quanto de classes comuns.

- Chamadas do método genérico:
  - `String middle = Arrayalg.<String>getMiddle("John","Q.","Public");`
  - `String middle = Arrayalg.getMiddle("John","Q.","Public");`

Métodos Genéricos – Exemplo


```
1  public class GenericMethodTest {
2    // método genérico
3    public static <T> void printArray(T[] array) {
4      for(T element : array)
5        System.out.printf("%s ", element);
6      System.out.println();
7    }
8
9    public static void main(String[] args) {
10     Integer[] integerArray = {1,2,3,4,5};
11     Double[] doubleArray = {1.1,2.2,3.3};
12     Character[] charArray = {'o','l','á'};
13
14     System.out.println("\nintegerArray contains:");
15     printArray(integerArray);
16     System.out.println("\ndoubleArray contains:");
17     printArray(doubleArray);
18     System.out.println("\ncharArray contains:");
19     printArray(charArray);
20   }
21 }
```

Exercício

- Escreva um método genérico chamado min que recebe um vetor de objetos e retorna o menor objeto.

Variáveis de Tipo Limitadas

- Algumas vezes, uma classe ou método precisa impor restrições aos parâmetros de tipo. Abaixo está um exemplo típico.

```
1 class ArrayAlg {
2     public static <T> T min(T[] a) {
3       if(a == null || a.length == 0) return null;
4       T smallest = a[0];
5       for(int i = 1; i < a.length; i++)
6           if (smallest.compareTo(a[i]) > 0) smallest = a[i];
7       return smallest;
8     }
9 }
```

- Problem: A variável `smallest` tem tipo T, o que significa que ela pode ser um objeto de uma classe arbitrária.
  Como garantir que a classe T tenha o método `compareTo`?

Variáveis de Tipo Limitadas

- Solução: Restringir o parâmetro de tipo para que ele seja um **subtipo** do tipo que você deseja:

```
1  class ArrayAlg {
2      public static <T extends Comparable<T>> T min(T[] a) {
3        if(a == null || a.length == 0) return null;
4        T smallest = a[0];
5        for(int i = 1; i < a.length; i++)
6            // We know for sure that compareTo is available
7            if (smallest.compareTo(a[i]) > 0) smallest = a[i];
8        return smallest;
9      }
10 }
```

- Use a palavra-chave `extends` tanto para classes quanto para interfaces.

Variáveis de Tipo Limitadas

- Solução: Restringir o parâmetro de tipo para que ele seja um **subtipo** do tipo que você deseja:

```
1  class ArrayAlg {
2      public static <T extends Comparable<T>> T min(T[] a) {
3        if(a == null || a.length == 0) return null;
4        T smallest = a[0];
5        for(int i = 1; i < a.length; i++)
6            // We know for sure that compareTo is available
7            if (smallest.compareTo(a[i]) > 0) smallest = a[i];
8        return smallest;
9      }
10 }
```

- Use a palavra-chave `extends` tanto para classes quanto para interfaces.

- É possível usar mútiplos limitantes:
  `T extends Comparable & Serializable`
  - No máximo um dos limitantes pode ser uma classe e, se ele for, deve aparecer primeiro na lista.

Métodos Genéricos – Sobrecarga

- Os métodos genéricos podem ser sobrecarregados por outros métodos genéricos ou por métodos não genéricos.

- Os métodos não genéricos têm precedência maior em relação ao genérico.

Métodos Genéricos – Sobrecarga


```
1  class SobrecargaTeste {
2    public static <T extends Comparable> T min(T[] a) {
3      if(a == null || a.length == 0) return null;
4      T smallest = a[0];
5      for(int i = 1; i < a.length; i++)
6        if(smallest.compareTo(a[i]) > 0)
7          smallest = a[i];
8      return smallest;
9    }
10
11   public static int min(int[] a) {
12     if(a == null || a.length == 0)
13       throw new RuntimeException("vetor vazio");
14     int smallest = a[0];
15     for(int i = 1; i < a.length; i++)
16       if(smallest > a[i])
17         smallest = a[i];
18     return smallest;
19   }
```

Métodos Genéricos – Sobrecarga


```
20    public static void main(String[] args) {
21      Short[] arr = new Short[0];
22      System.out.println("min = " + min(arr));
23      int[] vec = new int[0];
24      System.out.println("min = " + min(vec));
25    }
26  }
```

Sobrecarga de métodos genéricos – Atividade

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Sobrecarregue o método `printArray`, visto anteriormente e exibido abaixo, de modo que ele receba dois argumentos inteiros adicionais: `lowSubscript` e `highSubscript`. Uma chamada para este método imprime somente a porção do array no intervalo especificado. Valide `lowSubscript` e `highSubscript`. Se algum deles estiver fora do intervalo, o método `printArray` deve lançar uma `InvalidSubscriptException`; caso contrário, `printArray` deve retornar o número de elementos impressos. Escreva uma função `main` que use as duas versões da função `printArray` com arrays de Integer and Double.

```
1  // método genérico
2  public static <T> void printArray(T[] array) {
3    for(T element : array)
4      System.out.printf("%s ", element);
5    System.out.println();
6  }
```

Classes Genéricas

Classe Genérica (Tipo parametrizado)

- Uma classe com uma ou mais variáveis de tipo
  - **Parâmetros de tipo** são introduzidos depois do nome da classe, entre colchetes angulares ⟨ e ⟩.
  - Parâmetros de tipo são visíveis no corpo da classe.

```
1 class Pair<T, U> {
2   private T first;
3   private U second;
4   // ...
5 }
```

  - É comum usar letras maiúsculas para as parâmetros de tipo.

Classe Genérica (Tipo parametrizado)

- Uma classe com uma ou mais variáveis de tipo
  - **Parâmetros de tipo** são introduzidos depois do nome da classe, entre colchetes angulares ⟨ e ⟩.
  - Parâmetros de tipo são visíveis no corpo da classe.

```
1 class Pair<T, U> {
2   private T first;
3   private U second;
4   // ...
5 }
```

  - É comum usar letras maiúsculas para as parâmetros de tipo.

- Classes genéricas agem como fábricas de classes comuns. Os tipos genéricos são substituídos por tipos já criados.

```
1 Pair<Integer, String> pair = new Pair<>();
2 Integer first = pair.getFirst();
3 String second = pair.getSecond();
```

Classes Genéricas - Exemplo

Projeto Pair

```
1  public class Pair<T> {
2      private T first;
3      private T second;
4
5      public Pair() { first = null; second = null; }
6      public Pair(T first, T second) {
7          this.first = first;
8          this.second = second;
9      }
10
11     public T getFirst() { return first; }
12     public T getSecond() { return second; }
13
14     public void setFirst(T newValue) { first = newValue; }
15     public void setSecond(T newValue) { second = newValue; }
16 }
```

Classes Genéricas - Exemplo


```
1  class ArrayAlg {
2      /**
3       * Gets the minimum and maximum of an array of strings
4       * @param a an array of strings
5       * @return a pair with the min and max value, or null if a
        is null or empty
6       */
7      public static Pair<String> minmax(String[] a) {
8          if(a == null || a.length == 0) return null;
9          String min = a[0];
10         String max = a[0];
11         for (int i = 1; i < a.length; i++) {
12             if(min.compareTo(a[i]) > 0) min = a[i];
13             if(max.compareTo(a[i]) < 0) max = a[i];
14         }
15         return new Pair<>(min, max);
16     }
17 }
```

Classes Genéricas - Exemplo

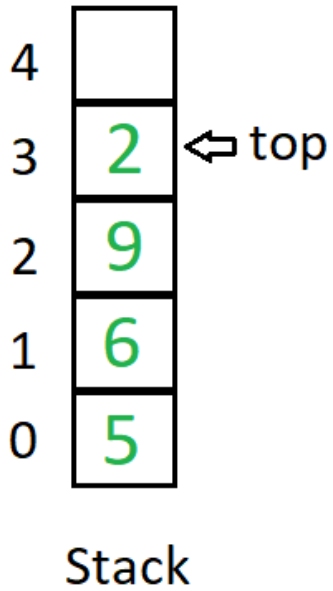

```
1 public class App {
2     public static void main(String[] args) throws Exception {
3         String[] words = {"Mary","had","a","little","lamb"};
4         Pair<String> mm = ArrayAlg.minmax(words);
5         System.out.println("min = " + mm.getFirst());
6         System.out.println("max = " + mm.getSecond());
7     }
8 }
```

Exemplo de Classe Genérica: Stack

Exemplo – Implementando uma Pilha

A estrutura de dados pilha pode ser implementada usando um array redimensionável e possui duas operações básicas:

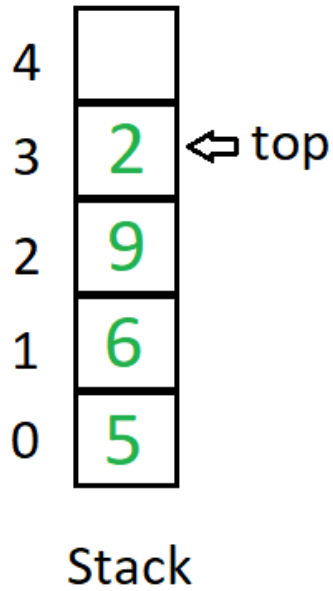
- `push(x)`: colocar o elemento `x` no topo da pilha
- `pop()` remover o elemento que está no topo da pilha e retornar o seu valor.



Exemplo – Implementando uma Pilha

A estrutura de dados pilha pode ser implementada usando um array redimensionável e possui duas operações básicas:

- `push(x)`: colocar o elemento `x` no topo da pilha
- `pop()` remover o elemento que está no topo da pilha e retornar o seu valor.



- Ao tentar remover elementos de uma pilha vazia, uma exceção deve ser lançada.
- **Atividade:** Implementar uma pilha genérica usando como base um ArrayList.

Type Erasure

Type Erasure – Métodos Genéricos

- Quando o compilador traduz o código para bytecode, os métodos genéricos têm seus argumentos substituídos por tipos de verdade.
  - Por padrão, se o parâmetro de tipo for não-limitado, o tipo **Object** é utilizado.
  - Diferentemente do que ocorre com *templates* em C++, em que uma cópia para cada tipo utilizado é criada.

Type Erasure – Métodos Genéricos

- Anteriormente, vimos o seguinte método:

```
1   // método genérico
2   public static <T> void printArray(T[] array) {
3     for(T element : array)
4       System.out.printf("%s ", element);
5     System.out.println();
6   }
```

- Abaixo, é apresentado o equivalente ao método genérico acima depois de compilado.

```
1   public static void printArray(Object[] array) {
2     for(Object element : array)
3       System.out.printf("%s ", element);
4     System.out.println();
5   }
```

Type Erasure – Métodos Genéricos

- Anteriormente, vimos o seguinte método:

```
1     public static <T extends Comparable<T>> T min(T[] a) {
2       if(a == null || a.length == 0) return null;
3       T smallest = a[0];
4       for(int i = 1; i < a.length; i++)
5           // We know for sure that compareTo is available
6           if (smallest.compareTo(a[i]) > 0) smallest = a[i];
7       return smallest;
8     }
```

- Abaixo, está o equivalente depois de compilado:

```
1     public static Comparable min(Comparable[] a) {
2       if(a == null || a.length == 0) return null;
3       Comparable smallest = a[0];
4       for(int i = 1; i < a.length; i++)
5           // We know for sure that compareTo is available
6           if (smallest.compareTo(a[i]) > 0) smallest = a[i];
7       return smallest;
8     }
```

Type Erasure – Classes Genéricas

Exemplo de uso de método genérico:

```
1 public class Test02 {
2   public static void main(String[] args) {
3     Integer[] arr = {6,5,3,1,2,7,9,8};
4
5     Integer minimum = min(arr);
6
7     System.out.println("Mínimo: " + minimum);
8   }
9 }
```

Type Erasure – Classes Genéricas

Exemplo de uso de método genérico:

```
1 public class Test02 {
2   public static void main(String[] args) {
3     Integer[] arr = {6,5,3,1,2,7,9,8};
4
5     Integer minimum = min(arr);
6
7     System.out.println("Mínimo: " + minimum);
8   }
9 }
```

A versão depois de compilada:

```
1 public class Test02 {
2   public static void main(String[] args) {
3     Integer[] arr = {6,5,3,1,2,7,9,8};
4
5     Integer minimum = (Integer) min(arr);
6
7     System.out.println("Mínimo: " + minimum);
8   }
9 }
```

Type Erasure – Classes Genéricas

Exemplo de classe genérica vista anteriormente:

```
1  public class Pair<T> {
2      private T first;
3      private T second;
4
5      public Pair() { first = null; second = null; }
6      public Pair(T first, T second) {
7          this.first = first;
8          this.second = second;
9      }
10
11     public T getFirst() { return first; }
12     public T getSecond() { return second; }
13
14     public void setFirst(T newValue) { first = newValue; }
15     public void setSecond(T newValue) { second = newValue; }
16 }
```

Type Erasure – Classes Genéricas

A versão depois de compilada:

```
1  public class Pair {
2      private Object first;
3      private Object second;
4
5      public Pair() { first = null; second = null; }
6      public Pair(Object first, Object second) {
7          this.first = first;
8          this.second = second;
9      }
10
11     public Object getFirst() { return first; }
12     public Object getSecond() { return second; }
13
14     public void setFirst(Object newValue) {
15         first = newValue;
16     }
17     public void setSecond(Object newValue) {
18         second = newValue;
19     }
20 }
```

Restrições e limitações do uso de genéricos

Algumas restrições e limitações

- Parâmetros de tipo não podem ser instanciados com tipos primitivos.
  - Alternativamente, você pode usar as classes empacotadoras.

Algumas restrições e limitações

- Parâmetros de tipo não podem ser instanciados com tipos primitivos.
  - Alternativamente, você pode usar as classes empacotadoras.

- Você não pode criar arrays de tipos parametrizados.
  - Alternativamente, crie ArrayLists de tipos parametrizados.

Algumas restrições e limitações

- Parâmetros de tipo não podem ser instanciados com tipos primitivos.
  - Alternativamente, você pode usar as classes empacotadoras.

- Você não pode criar arrays de tipos parametrizados.
  - Alternativamente, crie ArrayLists de tipos parametrizados.

- Você não pode lançar ou capturar instâncias de uma classe genérica.
  - Isso acontece porque é proibido a uma classe genérica estender a classe `Throwable`.
  - Por exemplo, a seguinte definição não compilará:
    ```
    public class Problem<T> extends Exception { ... }
    ```

Algumas restrições e limitações

- Um parâmetro de tipo **não pode ser usado em expressões new** dentro da própria classe parametrizada.

```
1 public class Pclass<T> {
2     public PClass() {
3     T object = new T();
4     T[] a = new T[10];
5     Pair<String>[] b = new Pair<String>[10];
6   }
7 }
```

Nenhuma destas expressões é válida.

Algumas restrições e limitações

- **Você não pode referenciar variáveis de tipo em atributos e métodos estáticos.** Por exemplo, o seguinte código não compilará:

```
public class Singleton<T>
{
   private static T singleInstance; // Error

   public static T getSingleInstance() // Error
   {
      if (singleInstance == null) construct new instance of T
      return singleInstance;
   }
}
```

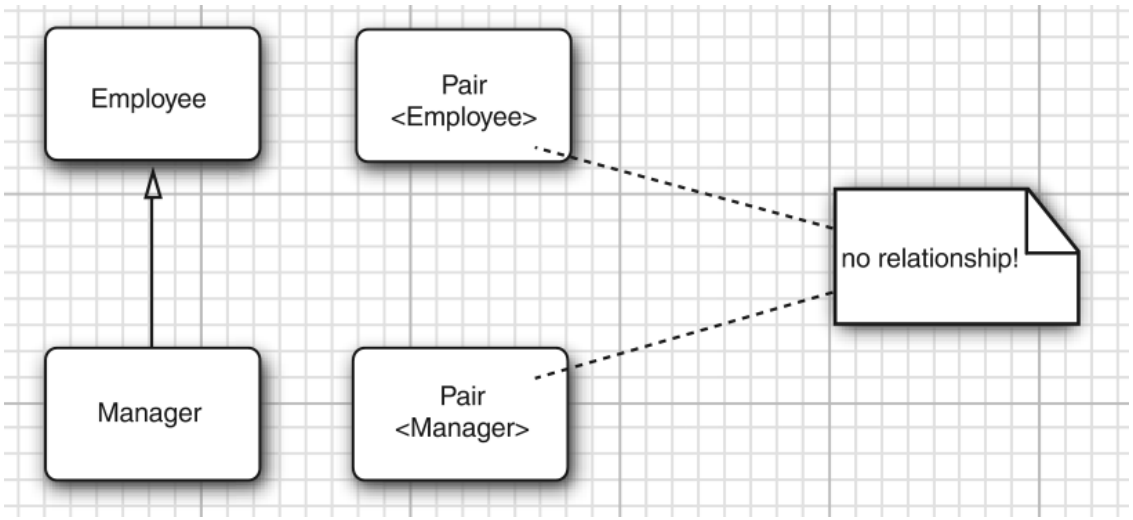
Classes Genéricas e Herança

Classes Genéricas e Herança

- Considere a classe `Empregado` e sua subclasse `Gerente`. A classe `Pair<Gerente>` é uma subclasse de `Pair<Empregado>`?

Classes Genéricas e Herança

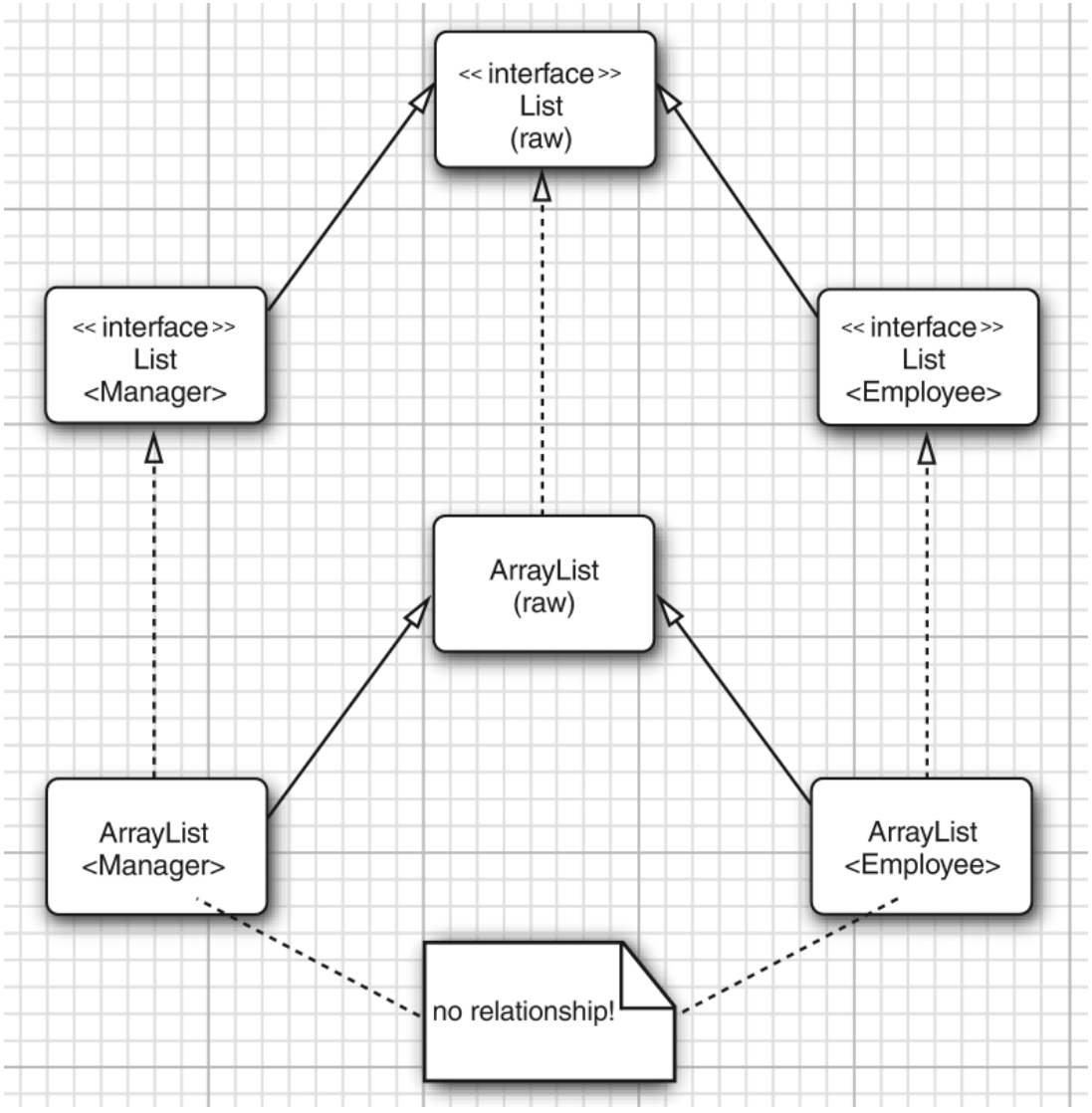
- Considere a classe `Empregado` e sua subclasse `Gerente`. A classe `Pair<Gerente>` é uma subclasse de `Pair<Empregado>`?
  - Resposta: Não.



Classes Genéricas e Herança

- Uma classe genérica pode ser derivada a partir de uma classe ou interface genérica.
  - Por exemplo, a classe `ArrayList<T>` implementa a interface `List<T>`. Isso significa que um `ArrayList<Gerente>` pode ser convertido para um `List<Gerente>`.
  - Contudo, um `ArrayList<Gerente>` não é um `ArrayList<Empregado>` ou `List<Empregado>`.

Classes Genéricas e Herança



Curingas (Wildcards)

Curingas em métodos genéricos

- Quando não pudermos determinar a classe específica dos elementos que serão passados a um genérico, podemos utilizar um **curinga (wildcard)**.
  - Por exemplo, em um método genérico que soma os elementos de um vetor, podemos não saber se tais elementos serão dos tipos `Integer` ou `Double`.
  - Podemos então indicar simplesmente que o tipo será o de uma classe que estende a classe `Number`.
    - De fato, `Integer` e `Double` são subclasses de `Number`.

- Um parâmetro curinga é indicado por ?, como abaixo:

```
ArrayList< ? extends Number > list
```

Curingas em métodos genéricos (cont.)


```
1    // total the elements, using wildcard in the List parameter
2    public static double sum(List<? extends Number> list) {
3       double total = 0; // initialize total
4
5       // calculate sum
6       for (Number element : list) {
7          total += element.doubleValue();
8       }
9
10      return total;
11   }
```

- Vantagem do uso de Wildcards: O método acima pode receber como parâmetro um objeto do tipo `List<T>` onde `T` pode ser trocado por qualquer subtipo da classe `Number`.

Curingas em métodos genéricos (cont.)

- Analisar o projeto WildCardTest. Ver as classes:
  - TotalNumbers.java
  - TotalNumbersErrors.java
  - WildcardTest.java
  - WildcardTest2.java

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM