Computação Orientada a Objetos

Coleções Java Parte I

Slides baseados em:
Deitel, H.M.; Deitel P.J. Java: Como Programar, Pearson Prentice Hall, 6a Edição, 2005. Capítulo 19

Profa. Karina Valdivia Delgado
EACH-USP

Estrutura for aprimorada

```
for(StackTraceElement element: traceElements){
    System.out.printf("%s\t", element.getClassName());
    System.out.printf("%s\t", element.getLineNumber());
    System.out.printf("%s\n", element.getMethodName());
}
```

A instrução for aprimorada itera pelos elementos de um array ou uma coleção sem utilizar um contador.

```
Rastreamento de pilha
Classe                  Linha    Metodo
graphics.Triangle       17       <init>
demo.TestGraphics       35       main
```

Sintaxe: Estrutura for aprimorada

```
for ( tipo objeto: nomeDaColeção){
   instruções
}
```

Exemplo:

```
for(StackTraceElement element: traceElements){
        System.out.printf("%s\t", element.getClassName());
        System.out.printf("%s\t", element.getLineNumber());
        System.out.printf("%s\n", element.getMethodName());
    }
```

Propósito: Iterar pelos elementos de um array ou coleção

Objetivo:

- Discutir em detalhes as estruturas de dados pré-empacotadas (estrutura de coleções), interfaces e algoritmos para manipular essas estruturas.
- Aprenderemos:
  - O que são coleções
  - Listas: ArrayList e LinkedList
  - Algoritmos de coleções
  - Conjuntos: HashSet e TreeSet
  - Mapas: Hashtable, HashMap e TreeMap
  - Como utilizar iteradores

Como criar e manipular estruturas de dados?

- Abordagem mais “baixo nível”
  - Criar cada elemento de cada estrutura de dados e modificar essas estruturas manipulando diretamente seus elementos e as referências a seus elementos.

Como criar e manipular estruturas de dados?

- Abordagem mais “alto nível”
  - Utilizar a estrutura de coleções de Java, que contém estruturas de dados pré-empacotadas, interfaces e algoritmos para manipular essas estruturas.
- Com as coleções, os programadores utilizam estruturas de dados existentes, sem se preocupar com a maneira como elas estão implementadas.

O que é uma coleção?

- É uma estrutura de dados (um objeto) que agrupa referências a vários outros objetos.
- Usadas para armazenar, recuperar e manipular elementos que formam um grupo natural (normalmente objetos do mesmo tipo).

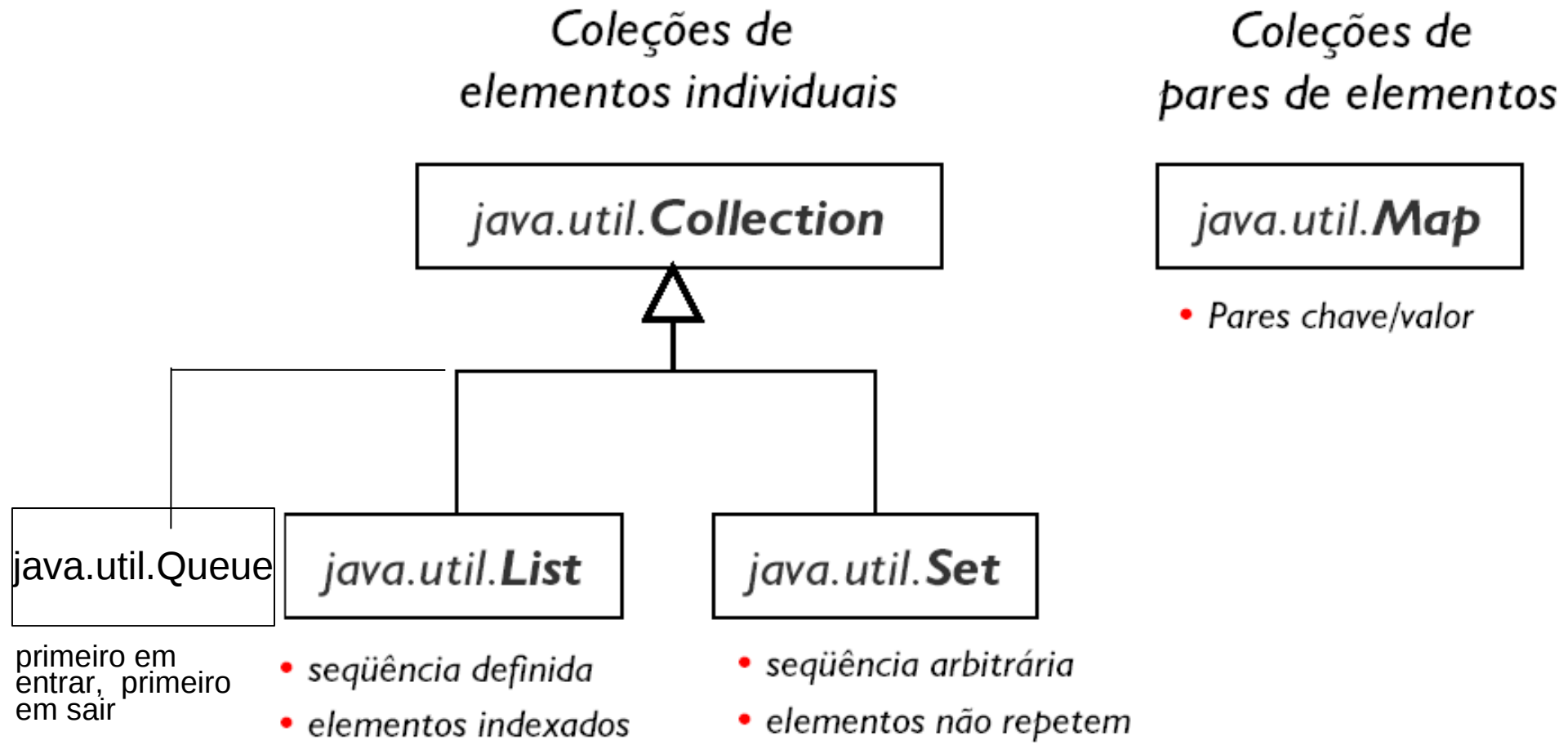
O pacote java.util

- As classes e interfaces da estrutura de coleções são membros do pacote **`java.util`**.

Interfaces da estrutura de coleções

- As interfaces da estrutura de coleções Java declaram operações a serem realizadas genericamente em vários tipos de coleções.
  - As principais estruturas de dados são implementadas usando apenas duas interfaces.
  - Existe uma única interface para iterar sobre os dados de qualquer estrutura

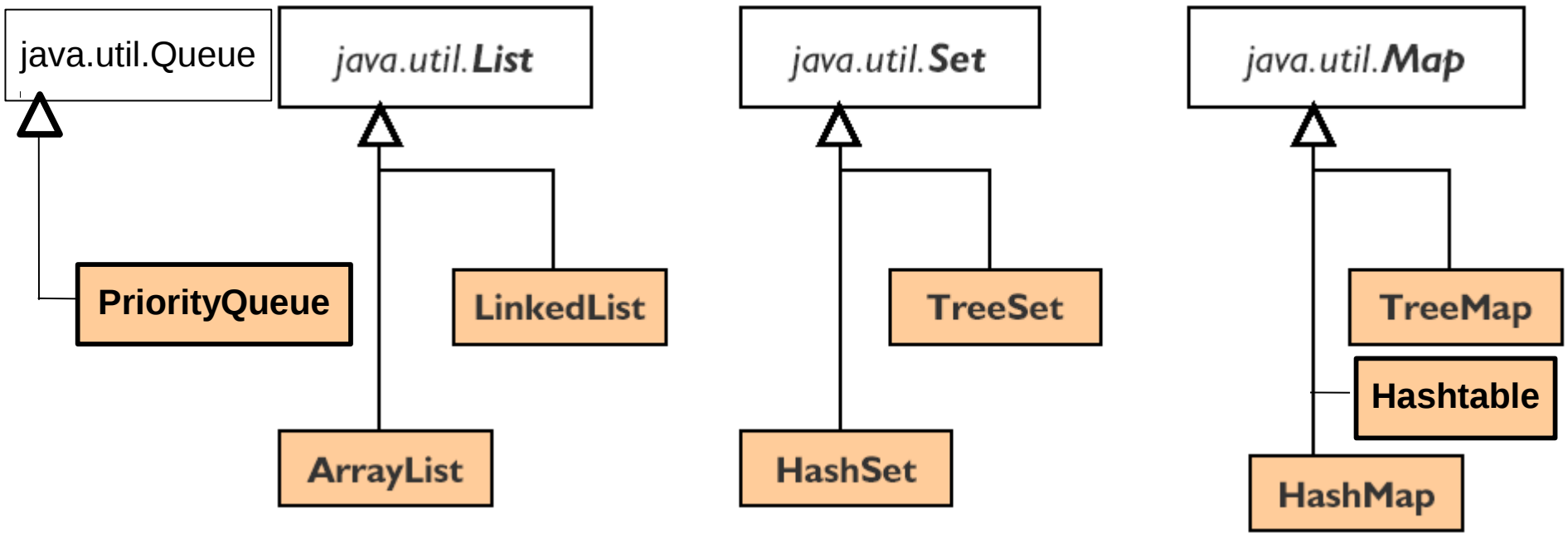
Interfaces da estrutura de coleções



Implementações da estrutura de coleções

- Várias implementações para essas interfaces são fornecidas dentro da estrutura de coleções Java.

Implementações da estrutura de coleções



Coleções com referências `Object`

- Nas primeiras versões Java, as classes na estrutura de coleções armazenavam e manipulavam referências **`Object`.**
- => era permitido armazenar qualquer objeto em uma coleção.
- Problemas ao recuperar referências **`Object`** de uma coleção, elas precisam ser convertidas no tipo apropriado (coerção).

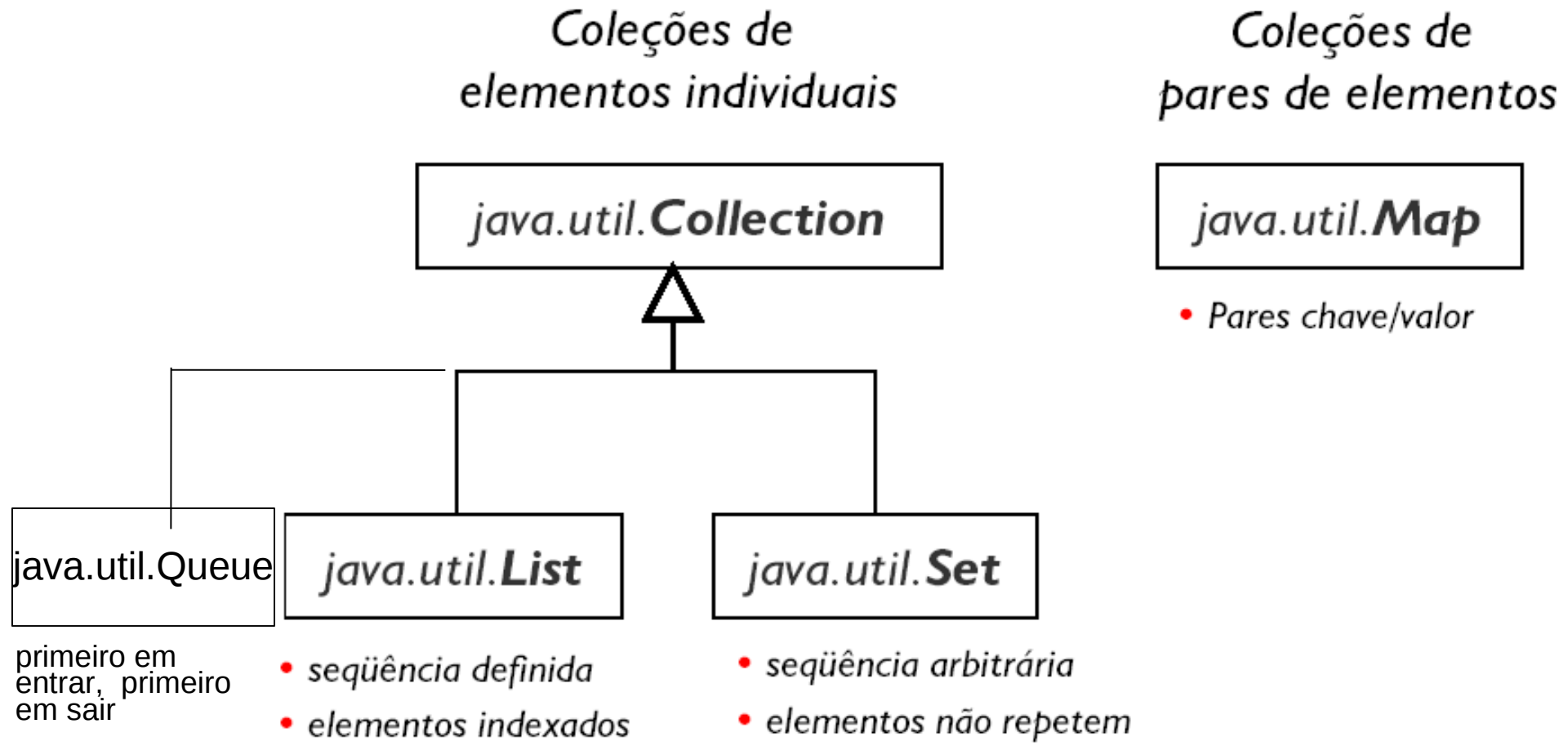
Coleções com genéricos

- A estrutura de coleções foi aprimorada para que seja possível especificar o tipo exato que será armazenado em uma coleção (genéricos).
- A verificação de tipos é feita em tempo de compilação
- O compilador assegura que os tipos apropriados à coleção estão sendo utilizados.

Coleções com genéricos

- Uma vez que o tipo armazenado em uma coleção é especificado, qualquer referência recuperada dessa coleção terá o tipo especificado.
- Isso elimina a necessidade de coerções que podem lançar exceções **`ClassCastException`** se o objeto referenciado não for do tipo apropriado.

Interface Collection



Interface **Collection**

- A **Interface Collection** é a raiz da hierarquia de coleções.
- É uma interface genérica
  - Ao declarar um objeto do tipo **Collection** deve-se especificar o tipo de objeto contido na coleção.
- A Interface Collection é utilizada para manipular coleções quando deseja-se obter o máximo de generalidade.

Interface `Collection`

- Operações básicas: atuam sobre elementos individuais em uma coleção, por ex:
  - adiciona elemento (**`add`**)
  - remove elemento (**`remove`**)
- Operações de volume: atuam sobre todos os elementos de uma coleção, por ex:
  - adiciona elementos da coleção (**`addAll`**)
  - remove elementos da coleção (**`removeAll`**)
  - mantém elementos da coleção (**`retainAll`**)

Interface `Collection`

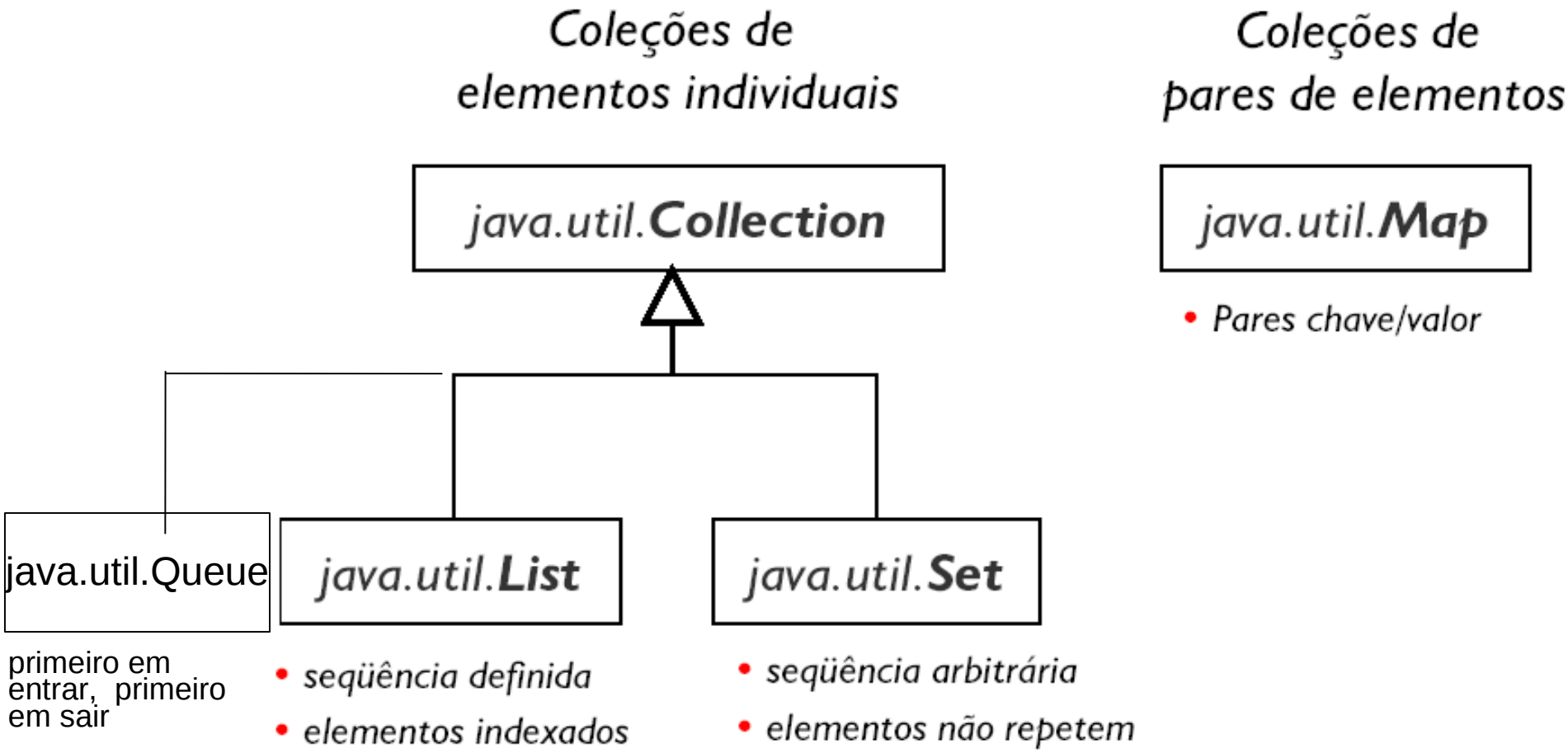
- Fornece operações para converter uma coleção em um array
  - **`Object[]`** **`toArray()`**

- Além disso, essa interface fornece um método **iterator()** que retorna um objeto **`Iterator`**:
  - permite a um programa percorrer a coleção e remover elementos da coleção durante a iteração.

Interface `Collection`

- Outros métodos:
  - determinar quantos elementos pertencem à coleção
    - `int size()`
  - determinar se uma coleção está ou não vazia
    - `boolean isEmpty()`

Interface List



Interface `List`

- Uma coleção do tipo `List` é uma Collection que tem uma sequência definida e que pode conter elementos duplicados.

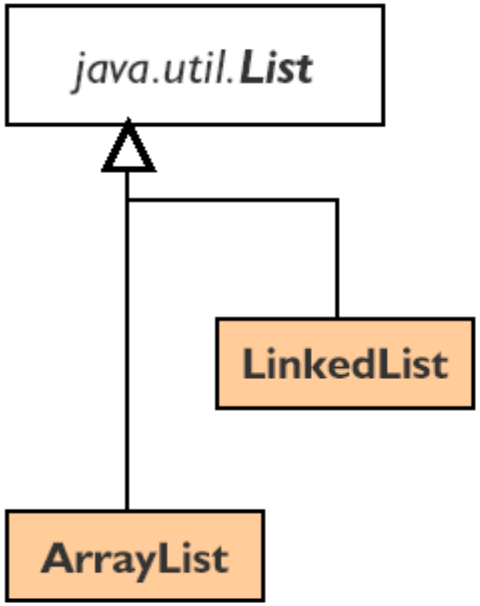
- Como os arrays o índice do primeiro elemento é zero.

Interface `List`

- Além dos métodos herdados de **Collection**, fornece métodos para:
  - manipular elementos via seus índices. Ex:
    - add(int index,Object o): Adiciona elemento. O tamanho da lista aumenta em 1.
    - remove(int index): Remove elemento da posição especificada e move todos os elementos após o elemento removido diminuindo o tamanho da lista em 1.
    - set(int index, Object o): Substitui elemento. O tamanho da lista permanece igual.
  - manipular um intervalo específico de elementos. Ex:
    - addAll(int index,Collection c) : Insere na posição especificada
    - subList(int fromIndex,int toIndex): obtem uma parte da lista, o indice final não faz parte do intervalo. Qualquer alteração na sublista também será feita na lista original (view)
  - recuperar elementos
    - get(int index)
  - obter um **`ListIterator`** para percorrer a lista.

Interface **`List`**

- **`List`** pode ser implementada por:
  - um vetor (array): classe **`ArrayList`**
  - ou uma lista ligada: classe **`LinkedList`**



Interface `List`: Exemplo

```
public static void main(String [ ] args) {
          List<Integer> L1;
          List<Integer> L2;
          L1 = new ArrayList<Integer>( );
          L2 = new LinkedList<Integer>( );
          for (int i = 0; i < 10 ; i++) {
             L1.add(i, i);
             L2.add(i, new Integer(i));
           }
           ...
   }
```

Iteradores de coleções (`Iterators`)

- Um iterador (objeto **`Iterator`**) percorre uma coleção, visitando os seus elementos.
  - funciona como uma espécie de ponteiro para o próximo elemento.

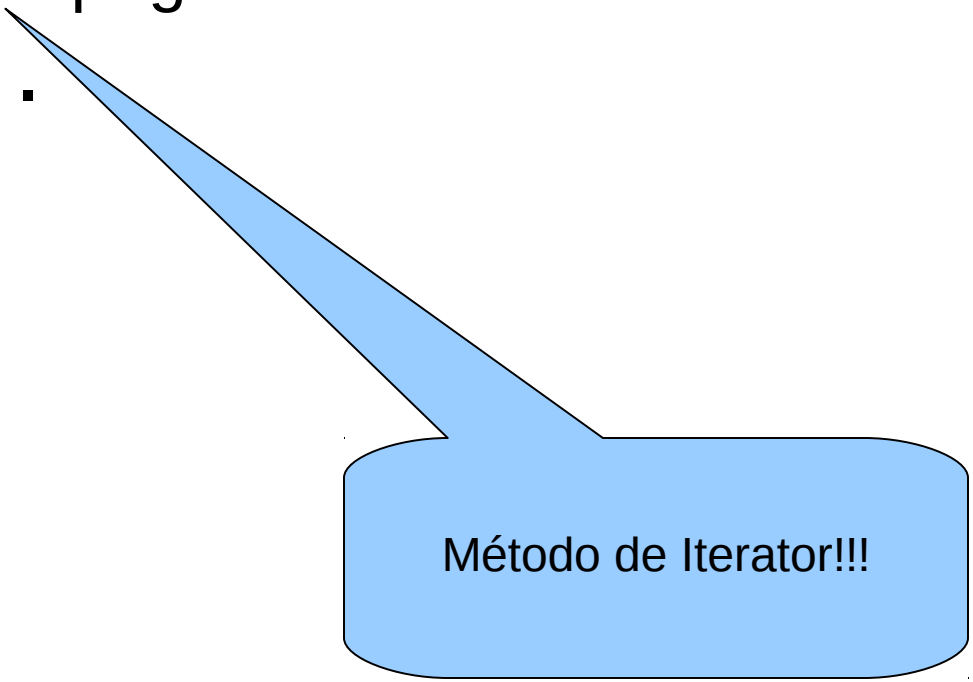
Iteradores de coleções (`Iterators`)

Passos para usar um objeto iterador:

1. Obtenha um iterador para a coleção usando o método **`iterator()`** da própria coleção.
   - Esse método retorna um objeto iterador, posicionado antes do primeiro objeto da coleção.
2. Verifique se há mais elementos na coleção com uma chamada ao método **`hasnext()`** do objeto iterador.
3. Obtenha o próximo objeto na coleção com o método **`next()`** do objeto iterador.

Iteradores de coleções (`Iterators`)

- O método **`remove()`** apaga o último item retornado pelo método **`next()`.**



Exemplo ArrayList e Iterator

- Tarefa1: colocar dois arrays de **`String`** em duas listas **`ArrayList.`**
- Tarefa 2: utilizar um objeto **`Iterator`** para remover da segunda coleção **`ArrayList`** todos os elementos que também estiverem na primeira coleção.

Exemplo ArrayList e Iterator

ArrayList list:

MAGENTA RED WHITE BLUE CYAN

ArrayList removeList:

RED WHITE BLUE

ArrayList list após chamar o método removeColors:

MAGENTA CYAN

Exemplo ArrayList e Iterator

```
public class CollectionTest {
   private static final String[] colors =
      { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
   private static final String[] removeColors =
      { "RED", "WHITE", "BLUE" };
   // cria ArrayList, adiciona Colors a ela e a manipula
   public CollectionTest() {
      List< String > list = new ArrayList< String >();
      List< String > removeList = new ArrayList< String >();
      for ( String color : colors )
         list.add( color );
      for ( String color : removeColors )
         removeList.add( color );
```

Exemplo ArrayList e Iterator

```
public class CollectionTest {
   private static final String[] colors =
      { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
   private static final String[] removeColors =
      { "RED", "WHITE", "BL
   // cria ArrayList, adicio
   public CollectionTest() {
      List< String > list =
      List< String > removeList =  ArrayList< String >();
      for ( String color : colors )
         list.add( color );
      for ( String color : removeColors )
         removeList.add( color );
```

Preenche a coleção **list** com objetos **String** armazenados no array **colors**

Exemplo ArrayList e Iterator

```
public class CollectionTest {
   private static final String[] colors =
      { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
   private static final String[] removeColors =
      { "RED", "WHITE", "BLUE" };
   // cria ArrayList, adiciona Colors a ela e a manipula
   public CollectionTest() {
      List< String >
      List< String >
      for ( String co
         list.add( color );
      for ( String color : removeColors )
         removeList.add( color );
```

Preenche a coleção **removelist** com objetos **String** armazenados no array **removeColors**

Exemplo ArrayList e Iterator

```
    System.out.println( "List: " );
    for ( int count = 0; count < list.size(); count++ )
      System.out.printf( "%s ", list.get(count));
    System.out.println( "\n RemoveList: " );
    for ( String color : removeList )
       System.out.printf( "%s ", color );
    // remove cores contidas em removeList
    removeColors( list, removeList );
    System.out.println("\n ArrayList after calling
  removeColors:");
    for ( String color : list )
       System.out.printf( "%s ", color );
} // fim do construtor CollectionTest
```

Exemplo ArrayList e Iterator

Chama o método **size** da interface **Collection** para obter o número de elementos da lista

```
    System.out.println( "List: " );
    for ( int count = 0; count < list.size(); count++ )
      System.out.printf( "%s ", list.get(count));
    System.out.println( "\n RemoveList: " );
    for ( String color : removeList )
       System.out.printf( "%s ", color );
    // remove cores contidas em removeList
    removeColors( list, removeList );
    System.out.println("\n ArrayList after calling
  removeColors:");
    for ( String color : list )
       System.out.printf( "%s ", color );
} // fim do construtor CollectionTest
```

Exemplo ArrayList e Iterator

Chama o método **get** da interface **List** para obter cada elemento da lista

```
    System.out.println( "List: " );
    for ( int count = 0; count < list.size(); count++ )
      System.out.printf( "%s ", list.get(count));
    System.out.println( "\n RemoveList: " );
    for ( String color : removeList )
       System.out.printf( "%s ", color );
    // remove cores contidas em removeList
    removeColors( list, removeList );
    System.out.println("\n ArrayList after calling
  removeColors:");
    for ( String color : list )
       System.out.printf( "%s ", color );
} // fim do construtor CollectionTest
```

Exemplo ArrayList e Iterator

A estrutura for aprimorada
poderia ter sido utilizada aqui !

```
    System.out.println(          : " );
    for ( int count = 0; count < list.size(); count++ )
      System.out.printf( "%s ", list.get(count));
    System.out.println( "\n RemoveList: " );
    for ( String color : removeList )
       System.out.printf( "%s ", color );
    // remove cores contidas em removeList
    removeColors( list, removeList );
    System.out.println("\n ArrayList after calling
  removeColors:");
    for ( String color : list )
       System.out.printf( "%s ", color );
} // fim do construtor CollectionTest
```

Exemplo ArrayList e Iterator

Remove de **collection1** as cores (objetos **String**) especificadas em **collection2**

```
private void removeColors( Collection< String > collection1,
   Collection< String > collection2){
      // obtém o iterador
      Iterator< String > iterator = collection1.iterator();

      // loop enquanto a coleção tiver itens
      while (iterator.hasNext())

         if (collection2.contains( iterator.next() ))
            iterator.remove();// remove Color atual
   } // fim do método removeColors
```

Exemplo ArrayList e Iterator

Permite que quaisquer objetos **Collection**s que contenham strings sejam passados como argumentos

```
private void removeColors( Collection< String > collection1,
   Collection< String > collection2){
      // obtém o iterador
      Iterator< String > iterator = collection1.iterator();

      // loop enquanto a coleção tiver itens
      while (iterator.hasNext())

         if (collection2.contains( iterator.next() ))
            iterator.remove();// remove Color atual
   } // fim do método removeColors
```

Exemplo ArrayList e Iterator

O método acessa os elementos da primeira coleção via um **Iterator**. Chama o método **iterator** para obter um iterador para **collection1**

```
private void removeColors( Collection< St          llection1,
   Collection< String > collection2){
      // obtém o iterador
      Iterator< String > iterator = collection1.iterator();

      // loop enquanto a coleção tiver itens
      while (iterator.hasNext())

         if (collection2.contains( iterator.next() ))
            iterator.remove();// remove Color atual
   } // fim do método removeColors
```

Exemplo ArrayList e Iterator

Chama o método **hasnext** da interface **Iterator** para determinar se a coleção tem mais elementos

```
private voi                rs( Collection< String > collection1,
   Collection<                collection2){
      // obtém o                r
      Iterator< Str              iterator = collection1.iterator();

      // loop enquanto a coleção tiver itens
      while (iterator.hasNext())

         if (collection2.contains( iterator.next() ))
            iterator.remove();// remove Color atual
   } // fim do método removeColors
```

Exemplo ArrayList e Iterator

Chama método **next** da interface **Iterator** para obter uma referência ao próximo elemento da coleção

```
private void removeColors( Collection<                    llection1,
   Collection< String > collection2){
      // obtém o iterador
      Iterator< String > iterator = colle                 iterator();

      // loop enquanto a coleção tiver ite
      while (iterator.hasNext())

         if (collection2.contains( iterator.next() ))
            iterator.remove();// remove Color atual
   } // fim do método removeColors
```

Exemplo ArrayList e Iterator

Utiliza o método **contains** da segunda coleção para determinar se a mesma contém o elemento retornado por **next**

```
private void rem                    llection< String > collection1,
   Collection< Str                  ction2){
      // obtém o ite
      Iterator< String              ator = collection1.iterator();

      // loop enquanto a              ção tiver itens
      while (iterator.hasNe          ())

         if (collection2.contains( iterator.next() ))
            iterator.remove();// remove Color atual
   } // fim do método removeColors
```

Erro de programação comum

- Se uma coleção for modificada por um de seus métodos depois de um iterador ter sido criado para essa coleção:
  - o iterador se torna imediatamente inválido! lançando **ConcurrentModificationException**