

# Computação Orientada a Objetos

Coleções - continuação

*Profa. Thienne Johnson*

*EACH/USP*



# Conteúdo

- ▶ Java, como programar, 6<sup>a</sup> edição
  - Deitel & Deitel
  - Capítulo 19

# Algoritmos de coleções

- ▶ A estrutura de coleções fornece vários algoritmos para operações em coleções
- ▶ Esses algoritmos estão implementados como métodos `static` da classe `Collections`

# Algoritmos de coleções

- ▶ Algoritmos que operam em objetos da classe `List`:
  - `sort`: classifica os elementos da lista
  - `binarySearch`: localiza um elemento da lista
  - `reverse`: inverte os elementos da lista
  - `shuffle`: "embaralha" os elementos da lista
  - `fill`: preenche uma lista
  - `copy`: copia referências de uma lista em outra lista

# Algoritmos de coleções

- ▶ Algoritmos que operam em objetos da classe **Collection**:
  - **min**: retorna o menor elemento em uma coleção
  - **max**: retorna o maior elemento em uma coleção
  - **frequency**: calcula quantos elementos em uma coleção são iguais a um elemento especificado
  - **disjoint**: determina se duas coleções não têm nenhum elemento em comum

# Algoritmo `Sort`

- ▶ O algoritmo `sort` classifica (ordena) os elementos de uma lista `List`
  - os tipos dos elementos da lista devem implementar a interface `Comparable`
- ▶ A ordem entre os elementos da lista é determinada pela ordem natural do tipo dos elementos
  - esquema implementado pela classe no método `compareTo`

# Algoritmo Sort - Exemplo

Classifica os elementos de  
uma lista em ordem crescente

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;

public class Sort1 {
    private static final String suits[] =
        { "Hearts", "Diamonds", "Clubs", "Spades" };

    public void printElements(){ // exhibe elementos do array
        List< String > list = Arrays.asList( suits ); // cria List

        // gera saída da lista
        System.out.printf( "Unsorted array elements:\n%s\n", list);

        Collections.sort( list ); // classifica ArrayList

        // gera saída da lista
        System.out.printf( "Sorted array elements:\n%s\n", list);
    } // fim do método printElements
```

# Algoritmo Sort - Exemplo

Chama o método `asList` da classe `Arrays` para permitir que o conteúdo do array seja manipulado como uma lista

```
import java.util.*;
import java.util.Collections;
import java.util.List;

public class Sort {
    private static final String[] suits =
        { "Hearts", "Diamonds", "Clubs", "Spades" };

    public void printElements() { // exibe elementos do array
        List< String > list = Arrays.asList( suits ); // cria List
        // gera saída da lista
        System.out.printf( "Unsorted array elements:\n%s\n", list );

        Collections.sort( list ); // classifica ArrayList

        // gera saída da lista
        System.out.printf( "Sorted array elements:\n%s\n", list );
    } // fim do método printElements
}
```



# Algoritmo Sort - Exemplo

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;

public class Sort1 {
    private static final String[] HEARTS = { "Hearts", "Diamonds", "Clubs", "Spades" };

    public void printElements() {
        List< String > list = Arrays.asList( HEARTS ); // cria List
        // gera saída da lista
        System.out.printf( "Unsorted array elements:\n%s\n", list );

        Collections.sort( list ); // classifica ArrayList

        // gera saída da lista
        System.out.printf( "Sorted array elements:\n%s\n", list );
    } // fim do método printElements
}
```

Chamada implícita ao método `toString` da classe `List` para gerar a saída do conteúdo da lista

# Algoritmo Sort - Exemplo

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;

public class Sort1 {
    private static final String suits[] =
        { "pades" };
    public void printElements( List<String> list ) {
        List<String> list = Arrays.asList( suits ); // cria List
        // ...
        System.out.printf( "Unsorted array elements:\n%s\n", list );
        Collections.sort( list ); // classifica ArrayList
        // gera saída da lista
        System.out.printf( "Sorted array elements:\n%s\n", list );
    } // fim do método printElements
}
```

Classifica a lista list  
em ordem crescente

# Algoritmo BinarySearch

- ▶ O algoritmo **BinarySearch** tenta localizar um elemento em uma lista **List**
  - se o elemento for encontrado, seu índice é retornado
  - se o elemento não for encontrado, o algoritmo retorna um valor negativo

# Algoritmo **BinarySearch**

- ▶ Observação: o algoritmo **BinarySearch** espera que os elementos da lista estejam classificados em ordem crescente

Utiliza o algoritmo `BinarySearch` para procurar uma série de strings em uma `ArrayList`

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;
import java.util.ArrayList;

public class BinarySearchTest
{
    private static final String colors[] = { "red", "white",
        "blue", "black", "yellow", "purple", "tan", "pink" };
    private List< String > list; // ArrayList reference

    // cria, classifica e gera a saída da lista
    public BinarySearchTest()
    {
        list = new ArrayList< String >( Arrays.asList( colors ) );
        Collections.sort( list ); // sort the ArrayList
        System.out.printf( "Sorted ArrayList: %s\n", list );
    } // end BinarySearchTest constructor
}
```

# Algoritmo BinarySearch - Exemplo

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;
import java.util.ArrayList;

public class BinarySearchTest
{
    private static final String colors[] = { "red", "white",
        "blue", "black", "yellow", "purple", "tan", "pink" };

    {
        list = new ArrayList< String >( Arrays.asList( colors ) );
        Collections.sort( list ); // sort the ArrayList
        System.out.printf( "Sorted ArrayList: %s\n", list );
    } // end BinarySearchTest constructor
```

Classifica a lista com o método `sort`  
da interface `Collection`

# Algoritmo BinarySearch - Exemplo

```
import java.util.List;
import java.util.Arrays;
import java.util.Collections;
import java.util.ArrayList;

public class BinarySearchTest
{
    private static final String colors[] = { "red", "white",
        "blue", "black", "yellow", "purple", "tan", "pink" };
    private List< String > list; // ArrayList reference

    //
    public void Gera a saída da lista classificada
    {
        list = new ArrayList< String > ( Arrays.asList( colors ) );
        Collections.sort( list ); // sort the ArrayList
        System.out.printf( "Sorted ArrayList: %s\n", list );
    } // end BinarySearchTest constructor
```

# Algoritmo BinarySearch - Exemplo

Realiza a pesquisa e envia  
o resultado para a saída

```
private void printSearchResults( String key )
{
    int result = 0;

    System.out.printf( "\n Buscando o elemento: %s\n", key );
    result = Collections.binarySearch( list, key );

    if ( result >= 0 )
        System.out.printf( "Elemento encontrado no índice %d\n", result );
    else
        System.out.printf( "Elemento não encontrado! (%d)\n", result );
} // fim do método printSearchResults
```



# Algoritmo BinarySearch - Exemplo

Chama o método `binarySearch` da interface `Collection` para buscar um elemento (`key`) em uma lista (`list`)

```
private void printSearchResults( String key )
{
    int result = 0;

    System.out.printf( "\n Buscando o elemento: %s\n", key );
    result = Collections.binarySearch( list, key );

    if ( result >= 0 )
        System.out.printf( "Elemento encontrado no índice %d\n", result );
    else
        System.out.printf( "Elemento não encontrado! (%d)\n", result );
} // fim do método printSearchResults
```

# Conjuntos

- ▶ Um conjunto (**Set**) é uma coleção que contém elementos únicos não duplicados
- ▶ A classe **HashSet** implementa a interface **Set**

# HashSet – Exemplo

Exemplo de um método que aceita um argumento `Collection` e constrói uma coleção `HashSet` a partir desse argumento

```
private void printNonDuplicates( Collection< String > collection )
{
    // create a HashSet
    Set< String > set = new HashSet< String >( collection );

    System.out.println( "\n A coleção sem duplicatas: " );

    for ( String s : set )
        System.out.printf( "%s ", s );

    System.out.println();
} // end method printNonDuplicates
```

# HashSet – Exemplo

Por definição, coleções da classe `Set` não contêm duplicatas, então quando a coleção `HashSet` é construída, ela remove quaisquer duplicatas passadas pelo argumento `Collection`

```
private void printNonDuplicates( Collection< String > collection )
{
    // create a HashSet
    Set< String > set = new HashSet< String >( collection );

    System.out.println( "\n A coleção sem duplicatas: " );

    for ( String s : set )
        System.out.printf( "%s ", s );

    System.out.println();
} // end method printNonDuplicates
```

# HashSet – Exemplo

Qual é a saída desse programa?

```
import java.util.List;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;
import java.util.Collection;

public class SetTest {
    private static final String colors[] = { "vermelho", "branco", "azul",
        "verde", "cinza", "laranja", "amarelo", "branco", "rosa",
        "violeta", "cinza", "laranja" };

    public SetTest(){
        List< String > list = Arrays.asList( colors );
        System.out.printf( "ArrayList: %s\n", list );
        printNonDuplicates( list );
    } // end SetTest constructor

    public static void main( String args[] ){
        new SetTest();
    } // end main
} // end class SetTest
```

# HashSet – Exemplo

- ▶ Saída do programa:

```
ArrayList: [vermelho, branco, azul, verde, cinza, laranja, amarelo,  
branco, rosa, violeta, cinza, laranja]
```

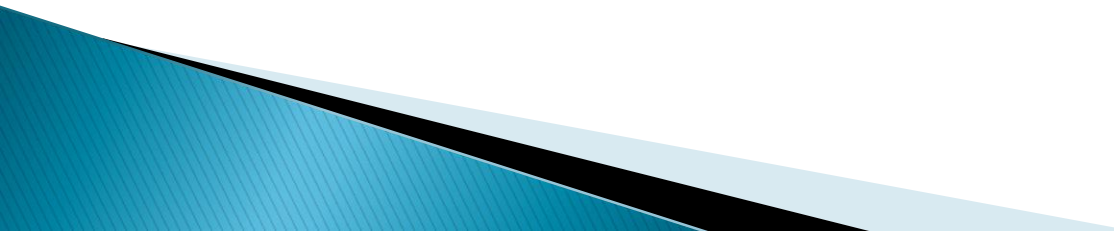
A coleção sem duplicatas:

```
vermelho branco azul verde cinza laranja amarelo rosa violeta
```

# Conjuntos ordenados

- ▶ A estrutura de coleções também inclui a interface **SortedSet**:
  - estende a interface **Set**
  - representa conjuntos que mantêm seus elementos ordenados
- ▶ A classe **TreeSet** implementa a interface **SortedSet**

# HashSet vs TreeSet

- ▶ *HashSet is much faster than TreeSet but offers no ordering guarantees.*
  - ▶ *If you need to use the operations in the SortedSet interface, or if value-ordered iteration is required, use TreeSet; otherwise, use HashSet.*
  - ▶ *It's a fair bet that you'll end up using HashSet most of the time.*
- 



# Coleção TreeSet

Inserir strings em uma coleção TreeSet

```
import java.util.Arrays;
import java.util.SortedSet;
import java.util.TreeSet;

public class SortedSetTest
{
    private static final String names[] = { "amarelo", "verde",
        "preto", "marrom", "cinza", "branco", "laranja", "vermelho", "verde" };

    // create a sorted set with TreeSet, then manipulate it
    public SortedSetTest()
    {
        // create TreeSet
        SortedSet< String > tree =
            new TreeSet< String >( Arrays.asList( names ) );
    }
}
```

# Coleção TreeSet

```
import java.util.Arrays;  
import java.util.SortedSet;  
import java.util.TreeSet;
```

```
public class SortedSetTest  
{
```

As strings são classificadas à medida em que são adicionadas à coleção **TreeSet**

```
{  
    // create TreeSet  
    SortedSet< String > tree =  
        new TreeSet< String >( Arrays.asList( names ) );
```

# Coleção TreeSet

```
System.out.println( "conjunto ordenado: " );  
printSet( tree );
```

Gera a saída do conjunto inicial de strings utilizando o método `printSet`

# Coleção TreeSet

Aceita uma coleção SortedSet como argumento e a imprime

```
private void printSet( SortedSet< String > set )
{
    for ( String s : set )
        System.out.print( s + " " );

    System.out.println();
}
```

# Coleção TreeSet

```
System.out.println( "conjunto ordenado: " );  
printSet( tree );
```

Saída:

```
conjunto ordenado:  
amarelo branco cinza laranja marrom preto verde vermelho
```

# Coleção TreeSet

```
System.out.print( "\n headSet (\"laranja\"):  " );  
printSet( tree.headSet( "laranja" ) );
```

Chama o método `headSet` da classe `TreeSet` para obter um subconjunto em que cada elemento é menor do que "laranja"

# Coleção TreeSet

```
System.out.print( "\n headSet (\"laranja\"):  " );  
printSet( tree.headSet( "laranja" ) );
```

Saída:

conjunto ordenado:

amarelo branco cinza laranja marrom preto verde vermelho

headSet("laranja"): amarelo branco cinza

# Coleção TreeSet

```
System.out.print( "\n tailSet (\"laranja\"):  " );  
printSet( tree.tailSet( "laranja" ) );
```

Chama o método `tailSet` da classe `TreeSet` para obter um subconjunto em que cada elemento é maior ou igual a "laranja"



# Coleção TreeSet

```
System.out.print( "\n tailSet (\"laranja\"):  " );  
printSet( tree.tailSet( "laranja" ) );
```

Saída:

conjunto ordenado:

amarelo branco cinza laranja marrom preto verde vermelho

headSet("laranja"): amarelo branco cinza

tailSet("laranja"): laranja marrom preto verde vermelho

# Coleção TreeSet

```
System.out.printf( "first: %s\n", tree.first() );  
System.out.printf( "last : %s\n", tree.last() );
```

Chama os métodos `first` e `last` da classe `TreeSet` para obter o menor e o maior elementos do conjunto

# Coleção TreeSet

```
System.out.printf( "first: %s\n", tree.first() );  
System.out.printf( "last : %s\n", tree.last() );
```

Saída:

conjunto ordenado:

amarelo branco cinza laranja marrom preto verde vermelho

headSet("laranja"): amarelo branco cinza

tailSet("laranja"): laranja marrom preto verde vermelho

first: amarelo

last: vermelho

# Mapas

<http://java.sun.com/javase/6/docs/api/java/util/Map.html>

- ▶ Um mapa (**Map**) associa chaves a valores e não pode conter chaves duplicatas
  - cada chave pode mapear somente um valor (mapeamento um para um)
- ▶ Classes que implementam a interface **Map**
  - **HashMap**
  - **TreeMap**

# Mapas (cont.)

- ▶ If you need key-ordered Collection-view iteration, use TreeMap;
- ▶ If you want maximum speed and don't care about iteration order, use HashMap;

# Mapas ordenados

- ▶ A interface (**SortedMap**) estende a interface **Map** e mantém as suas chaves ordenadas
- ▶ A classe **TreeMap** implementa a interface **SortedMap**

# Coleção HashMap

Utiliza uma coleção `HashMap` para contar o número de ocorrências de cada palavra em uma string

```
import java.util.StringTokenizer;
import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.TreeSet;
import java.util.Scanner;

public class WordTypeCount{
    private Map< String, Integer > map;
    private Scanner scanner;
    public WordTypeCount(){
        map = new HashMap< String, Integer >(); // cria HashMap
        scanner = new Scanner( System.in ); // cria scanner
        createMap(); // cria mapa baseado na entrada do usuário
        displayMap(); // apresenta conteúdo do mapa
    } // fim do construtor de WordTypeCount
```

# Coleção HashMap

```
1. import java.util.StringTokenizer;
2. import java.util.Map;
3. import java.util.HashMap;
4. import java.util.Set;
5. import java.util.TreeSet;
6.
```

Cria uma coleção `HashMap` vazia com chaves do tipo `String` e valores do tipo `Integer`

```
1.
2.
3.
4. public WordTypeCount() {
5.     map = new HashMap< String, Integer >(); // cria HashMap
6.     scanner = new Scanner( System.in ); // cria scanner
7.     createMap(); // cria mapa baseado na entrada do usuário
8.     displayMap(); // apresenta conteúdo do mapa
9. } //fim do construtor de WordTypeCount
```



# Coleção HashMap

```
1. import java.util.StringTokenizer;
2. import java.util.Map;
3. import java.util.HashMap;
4. import java.util.Set;
5. import java.util.TreeSet;
6. import java.util.Scanner;
```

Cria uma objeto `Scanner` que lê a entrada do usuário a partir do fluxo de entrada padrão

```
5.         map = new HashMap< String, Integer >(); // cria HashMap
6.         scanner = new Scanner( System.in ); // cria scanner
7.         createMap(); // cria mapa baseado na entrada do usuário
8.         displayMap(); // apresenta conteúdo do mapa
9.     } //fim do construtor de WordTypeCount
```

# Coleção HashMap

```
1. import java.util.StringTokenizer;
2. import java.util.Map;
3. import java.util.HashMap;
4. import java.util.Set;
5. import java.util.TreeSet;
6. import java.util.Scanner;
```

Chama o método `createMap` para armazenar no mapa o número de ocorrências de cada palavra na frase

```
5.         hashMap< String, Integer >(); // cria HashMap
6.         scanner = new Scanner( System.in ); // cria scanner
7.         createMap(); // cria mapa baseado na entrada do usuário
8.         displayMap(); // apresenta conteúdo do mapa
9.     } //fim do construtor de WordTypeCount
```

# Coleção HashMap

Método `createMap`: armazena em um mapa o número de ocorrências de cada palavra na frase do usuário

```
private void createMap(){
    System.out.println( "Entre com uma string:" ); // prompt para entrada do
    String input = scanner.nextLine();

    // cria um objeto StringTokenizer para a entrada
    StringTokenizer tokenizer = new StringTokenizer( input );

    // processando o texto de entrada
    while ( tokenizer.hasMoreTokens() ) // enquanto tiver mais entrada
    {
        String word = tokenizer.nextToken().toLowerCase(); // captura palavra

        // se o mapa contiver a palavra
        if ( map.containsKey( word ) ) // a palavra está no mapa?
        {
            int count = map.get( word ); // obtém contagem atual
            map.put( word, count + 1 ); // incrementa a contagem
        } // fim if
        else
            map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa
    } // end while
} // end method createMap
```

# Coleção HashMap

Invoca o método `nextLine` da classe `Scanner` para ler a entrada do usuário

```
private void createMap() {
    System.out.println( "Entre com uma frase:" ); // prompt para entrada do usuário
    String input = scanner.nextLine();

    // cria um objeto StringTokenizer para a entrada
    StringTokenizer tokenizer = new StringTokenizer( input );

    // processando o texto de entrada
    while ( tokenizer.hasMoreTokens() ) // enquanto tiver mais entrada
    {
        String word = tokenizer.nextToken().toLowerCase(); // captura palavra

        // se o mapa contiver a palavra
        if ( map.containsKey( word ) ) // a palavra está no mapa?
        {
            int count = map.get( word ); // obtém contagem atual
            map.put( word, count + 1 ); // incrementa a contagem
        } // fim if
        else
            map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa
    } // end while
} // end method createMap
```

# Coleção HashMap

Cria um objeto `StringTokenizer` para dividir a string de entrada em suas palavras componentes individuais

```
// cria um objeto StringTokenizer para a entrada
StringTokenizer tokenizer = new StringTokenizer( input );

// processando o texto de entrada
while ( tokenizer.hasMoreTokens() ) // enquanto tiver mais entrada
{
    String word = tokenizer.nextToken().toLowerCase(); // captura palavra

    // se o mapa contiver a palavra
    if ( map.containsKey( word ) ) // a palavra está no mapa?
    {
        int count = map.get( word ); // obtém contagem atual
        map.put( word, count + 1 ); // incrementa a contagem
    } // fim if
    else
        map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa
} // end while
} // end method createMap
```

# Coleção HashMap

Utiliza o método `hasMoreTokens` para determinar se há mais tokens na string sendo separada em tokens

```
// processando o texto de entrada
while ( tokenizer.hasMoreTokens() ) // enquanto tiver mais entrada
{
    String word = tokenizer.nextToken().toLowerCase(); // captura palavra

    // se o mapa contiver a palavra
    if ( map.containsKey( word ) ) // a palavra está no mapa?
    {
        int count = map.get( word ); // obtém contagem atual
        map.put( word, count + 1 ); // incrementa a contagem
    } // fim if
    else
        map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa
} // end while
} // end method createMap
```

# Coleção HashMap

```
private void createMap() {
    // ... entrada do usuário
    while ( tokenizer.hasMoreTokens() ) // enquanto tiver mais entrada
    {
        String word = tokenizer.nextToken().toLowerCase(); // captura palavra

        // se o mapa contiver a palavra
        if ( map.containsKey( word ) ) // a palavra está no mapa?
        {
            int count = map.get( word ); // obtém contagem atual
            map.put( word, count + 1 ); // incrementa a contagem
        } // fim if
        else
            map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa
    } // end while
} // end method createMap
```

Se houver mais tokens, o próximo é convertido em letras minúsculas

# Coleção HashMap

```
private void createMap(){  
    System.out.println( "Entre com uma frase:" ); // prompt para entrada do usuário
```

O próximo token é obtido com uma chamada ao método `nextToken` da classe `StringTokenizer`, que retorna uma string

```
{  
    String word = tokenizer.nextToken().toLowerCase(); // captura palavra  
  
    // se o mapa contiver a palavra  
    if ( map.containsKey( word ) ) // a palavra está no mapa?  
    {  
        int count = map.get( word ); // obtém contagem atual  
        map.put( word, count + 1 ); // incrementa a contagem  
    } // fim if  
    else  
        map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa  
} // end while  
} // end method createMap
```



# Coleção HashMap

```
private void createMap(){
    System.out.println( "Entre com uma frase:" ); // prompt para entrada do usuário
    String input = scanner.nextLine();

    // cria um objeto StringTokenizer para a entrada

    // se o mapa contiver a palavra
    if ( map.containsKey( word ) ) // a palavra está no mapa?
    {
        int count = map.get( word ); // obtém contagem atual
        map.put( word, count + 1 ); // incrementa a contagem
    } // fim if
    else
        map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa
} // end while
} // end method createMap
```

Chama o método `containsKey` da classe `Map` para determinar se a palavra já está no mapa

# Coleção HashMap

```
private void createMap(){
    System.out.println( "Entre com uma frase:" ); // prompt para entrada do usuário
    String input = scanner.nextLine();

    // cria um objeto StringTokenizer para a entrada
```

Se a palavra estiver no mapa, utiliza o método `get` da classe `Map` para obter o valor associado (a contagem) da chave no mapa

```
    // se a palavra
    if ( map.containsKey( word ) ) // a palavra está no mapa?
    {
        int count = map.get( word ); // obtém contagem atual
        map.put( word, count + 1 ); // incrementa a contagem
    } // fim if
    else
        map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa
} // end while
} // end method createMap
```

# Coleção HashMap

```
private void createMap(){
    System.out.println( "Entre com uma frase:" ); // prompt para entrada do usuário
    String input = scanner.nextLine();

    // cria um objeto StringTokenizer para a entrada
    StringTokenizer tokenizer = new StringTokenizer( input );
```

Incrementa esse valor e utiliza o método `put` da classe `Map` para substituir o valor associado à chave no mapa

```
        {
            int count = map.get( word ); // obtém contagem atual
            map.put( word, count + 1 ); // incrementa a contagem
        } // fim if
    else
        map.put( word, 1 ); // adiciona a nova palavra com contagem 1 ao mapa
} // end while
} // end method createMap
```

ada

ra palavra

# Coleção HashMap

```
1. import java.util.StringTokenizer;  
2. import java.util.Map;  
3. import java.util.HashMap;  
4. import java.util.Set;  
5. import java.util.TreeSet;  
6. import java.util.Scanner;
```

```
1. public class WordTypeCount{
```

Chama o método `displayMap` para exibir todas as entradas do mapa

```
6. Scanner scanner = new Scanner( System.in ); // cria scanner  
7. Map map = new HashMap(); // cria mapa baseado na entrada do usuário  
8. displayMap(); // apresenta conteúdo do mapa  
9. } //fim do construtor de WordTypeCount
```

# Coleção HashMap

Método `displayMap`: exibe todas as entradas armazenadas em um mapa

```
private void displayMap()
{
    Set< String > keys = map.keySet(); // obtém as chaves

    // ordena as chaves
    TreeSet< String > sortedKeys = new TreeSet< String >( keys );

    System.out.println( "O mapa contém:\nKey\t\tValue" );

    // gera a saída para cada chave no mapa
    for ( String key : sortedKeys )
        System.out.printf( "%s\t\t%s \n", key, map.get( key ) );
} // end method displayMap
```

# Coleção HashMap

Utiliza o método `keySet` da classe `HashMap` para obter um conjunto das chaves

```
private void displayMap()
{
    Set< String > keys = map.keySet(); // obtém as chaves

    // ordena as chaves
    TreeSet< String > sortedKeys = new TreeSet< String >( keys );

    System.out.println( "O mapa contém:\nKey\t\tValue" );

    // gera a saída para cada chave no mapa
    for ( String key : sortedKeys )
        System.out.printf( "%s\t\t%s\n", key, map.get( key ) );
} // end method displayMap
```

# Coleção HashMap

private

Cria um objeto `TreeSet` com as chaves, em que as chaves são ordenadas

```
// ordena as chaves
TreeSet< String > sortedKeys = new TreeSet< String >( keys );

System.out.println( "O mapa contém:\nKey\t\tValue" );

// gera a saída para cada chave no mapa
for ( String key : sortedKeys )
    System.out.printf("%s\t\t%s \n", key, map.get( key ) );
} // end method displayMap
```

# Coleção HashMap

```
private void displayMap()  
{  
    Set< String > keys = map.keySet(); // obtém as chaves  
  
    // imprime cada chave e seu valor no mapa  
    for ( String key : keys )  
        System.out.printf("%s\t\t%s \n", key, map.get( key ) );  
}  
// end method displayMap
```

Imprime cada chave e seu valor no mapa



# Coleção HashMap

Saída:

Entre com uma frase:

To be or not to be: that is the question

O mapa contém:

Key	Value
be	1
be:	1
is	1
not	1
or	1
question	1
that	1
the	1
to	2

# Exercícios teóricos para P1

- ▶ <http://www.usp.br/thienne/coo/material/exercicios-livro-p1.zip>