


PROGRAMAÇÃO ORIENTADA A OBJETOS

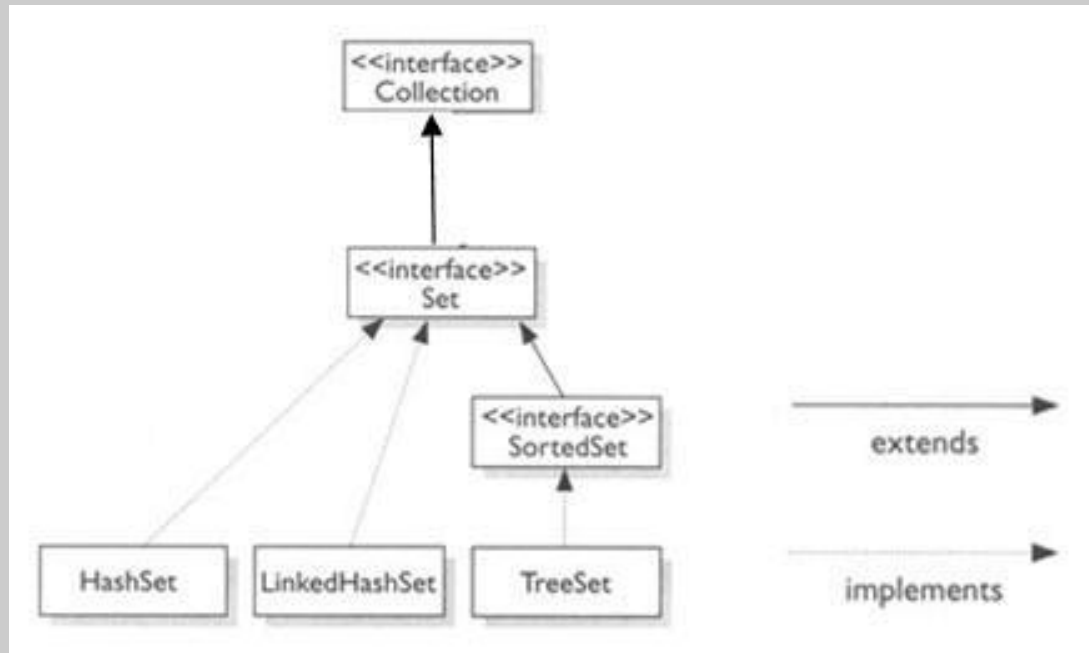
Conjuntos

ROTEIRO

- Conjuntos
 - Características
 - Declaração
 - Exemplos
- 

Conjuntos

- Um conjunto funciona de forma análoga aos conjuntos da matemática
- Trata-se de uma Collection que não permite elementos duplicados
- Um conjunto é representado pela interface Set, tendo como principais implementações as classes:
 - HashSet
 - LinkedHashSet
 - TreeSet



Conjuntos

- **Características Fundamentais**

- A ordem em que os elementos são armazenados pode não ser a mesma ordem em que os elementos foram inseridos no conjunto
- Ao percorrer um conjunto a sua ordem não é conhecida
- Velocidade na pesquisa de dados é mais rápida que um objeto do tipo List;
- Não precisa especificar a posição para adicionar um elemento;
- Não aceita valores duplicados. Se caso inserir um registro que já tenha no Set não será adicionado.
- Podem ser implementados como instâncias das classes HashSet ou TreeSet;
- Para lidar com conjuntos em Java utilizamos o pacote `java.util.set`

Conjuntos

- Declaração do Set
 - *Set E = new Type();*
 - *E* □ *é o objeto declarado*
 - *Type()* □ *Tipo de objeto da coleção a ser utilizado*
- O código abaixo cria um conjunto e adiciona vários elementos, e alguns deles repetidos

```
Set<String> cargos = new HashSet<>();  
cargos.add("Gerente");  
cargos.add("Diretor");  
cargos.add("Presidente");  
cargos.add("Secretária");  
cargos.add("Funcionário");  
cargos.add("Diretor");  
// imprime na tela todos os elementos  
System.out.println(cargos);
```

Conjuntos

- No caso do exemplo anterior, o segundo diretor não será armazenado e o método ***add*** retornará falso
- O fato de o set não considerar a ordem e não aceitar elementos repetidos, ainda assim permite que a implementação por exemplo do HashSet tenha um desempenho melhor em relação às List quando utilizado para pesquisa

Conjuntos

- **Classe HashSet**

- Faz parte do pacote `java.util`
- É uma implementação da interface `Set`, utilizando uma tabela Hash

- **Características**

- Não tem ordenação ao varrer ou imprimir, sendo que a ordem de saída não é a mesma da ordem de entrada
- Aceitam valores tipo `null`

Conjuntos

- Declaração do HashSet

- *Set E = new Type();*

- *E* □ *é o objeto declarado*

- *Type()* □ *Tipo de objeto da coleção a ser utilizado*

Conjuntos

- **Classe TreeSet**

- Faz parte do pacote `java.util`
- Faz parte da implementação da interface `Set`
- Armazena os elementos em uma árvore

- **Características**

- Os elementos inseridos dentro deste tipo de conjunto devem implementar a interface `Comparable`
- A ordenação é por elementos únicos
- Não suporta objetos nulos. Caso isso ocorra é lançado a exceção `NullPointerException`

Conjuntos

- Declaração do TreeSet
 - *Set E = new Type();*
 - *E* □ *é o objeto declarado*
 - *Type()* □ *Tipo de objeto da coleção a ser utilizado*

Exemplos

- Utilização de um HashSet para remover elementos duplicados

```
public class SetTest
```

```
{
```

```
private static final String colors[] = { "vermelho", "branco", "azul",  
"verde", "laranja", "branco", "cinza" };
```

```
// cria conjunto de array para eliminar duplicatas
```

```
private void printNonDuplicates( Collection< String > collection )
```

```
{
```

```
// cria um HashSet
```

```
Set< String > set = new HashSet< String >( collection );
```

```
System.out.println( "\nNonDuplicates are: " );
```

```
for ( String s : set )
```

```
System.out.printf( "%s ", s );
```

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

```
} // fim do método printNonDuplicates
```

Exemplos

- Utilização de um HashSet para remover elementos duplicados

// cria ArrayList e gera sua saída

public SetTest()

{

List< String > list = Arrays.asList(colors);

System.out.printf("ArrayList: %s\n", list);

printNonDuplicates(list);

} // fim do construtor SetT

public static void main(String args[])

{

new SetTest();

} // fim do main

} // fim da classe SetTest

Exemplos

- Utilização de TreeSet e SortedTest

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

// cria um conjunto classificado com TreeSet, e depois o manipula
public SortedSetTest()
{
// cria o TreeSet
SortedSet< String > tree =
new TreeSet< String >( Arrays.asList( names ) );

System.out.println( "sorted set: " );
printSet( tree ); // gera saída do conteúdo da árvore

// obtém headSet com base em "laranja"
System.out.print( "\nheadSet (\"laranja\"): " );
printSet( tree.headSet( "laranja" ) );
}
```

Exemplos

- Utilização de TreeSet e SortedTest

// obtém tailSet baseado em "laranka"

System.out.print("tailSet (\"laranja\"): ");

printSet(tree.tailSet("laranja"));

// obtém primeiro e últimos elementos

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

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

} // fim do construtor SortedSetTest

Exemplos

- Utilização de TreeSet e SortedTest

// gera saída do conteúdo

```
private void printSet( SortedSet< String > set )  
{  
    for ( String s : set )  
        System.out.printf( "%s ", s );  
        System.out.println();  
} // fim do método printSet
```

```
public static void main( String args[] )  
{  
    new SortedSetTest();  
} // fim do main  
} // fim da classe SortedSetTest
```

Referências

1. **Java Como Programar: Paul Deitel & Harvey Deitel - 10ª Edição**
2. **Java Como Programar: Paul Deitel & Harvey Deitel - 8ª Edição**

PROGRAMAÇÃO ORIENTADA A OBJETOS

Conjuntos