PROGRAMAÇÃO ORIENTADA A OBJETOS

Coleções
Parte 2

ROTEIRO

- **Iteradores**
- **Algoritmos de Collections**
- **Exemplos**

Iteradores

- **A função de objetos do tipo *Iterator* é permitir percorrer e remover elementos de uma coleção**
  - **Toda coleção possui um método que retorna um *Iterator***
  - **A interface Interator apresenta 3 métodos**

```
public interface Iterator<E> {
        boolean hasNext(); E
        next();
        void remove();
}
```

Iteradores

- Métodos da interface Interator
  - boolean hasNext()
    - Retorna true se há elementos a serem lidos no iterador
  - E next()
    - Retorna o próximo elemento do iterador
  - void remove()
    - Remove o último elemento obtido pela chamada de next()
    - Só é possível chamar uma vez para cada chamada de next()
    - Se essa regra for desrespeitada, uma exceção é lançada

Iteradores

- **O método remove() de Iterator é a única maneira segura de alterar uma coleção durante uma iteração**

```
Antes:  5    9    12    18    25
        55   67   81    83
```

```
static void filtro(List<Integer> list) {
        for (Iterator<Integer> it = list.iterator(); it.hasNext(); ) if (it.next() > 10)
                        it.remove();

}
```

```
Saída:    5  9
```

```
static void filtro(List<Integer> list) { for (int i = 0; i < list.size();
        i++)
                if (list.get(i) > 10)
                        list.remove(i);
}
```

```
Saída:  5    9    18
        55   81
```

Iteradores

- **Vantagem de iteradores**
  - **não depende do tipo de coleção**
  - **a *Interface Collection* provê método *iterator()* o que permite criar uma solução genérica**
- **Não são todas as coleções que têm um método de remoção por índice como List e cada coleção possui uma maneira de percorrer os elementos**

Algoritmos de Collections

- A classe Collections possui alguns métodos estáticos para manipular coleções
- Por exemplo, os algoritmos de List
  - Sort: Classifica os elementos de uma List.
  - binarySearch: Localiza um objeto em uma List.
  - reverse: Inverte os elementos de uma List.
  - shuffle: Ordena aleatoriamente os elementos de uma List.
  - fill: Configura todo elemento List para referir-se a um objeto especificado.
  - copy: Copia referências de uma List em outra.

Algoritmos de Collections

- **Além dos listados anteriormente há também:**
  - **min: Retorna o menor elemento em uma Collection.**
  - **max: Retorna o maior elemento em uma Collection.**
  - **addAll Acrescenta todos os elementos em um array a uma coleção.**
  - **frequency: Calcula quantos elementos na coleção são iguais ao elemento especificado.**
  - **Disjoint: Determina se duas coleções não têm nenhum elemento em comum**

- **Exemplo com o algoritmo sort**

```
public class Sort1
{
 private static final String suits[] = { "Hearts", "Diamonds", "Clubs", "Spades" };
 // exibe elementos do array
 public void printElements()
 {
        List< String > list = Arrays.asList( suits )
        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
 public static void main( String args[] )
 {
   Sort1 sort1 = new Sort1();
   sort1.printElements();
  } // fim do main
} // fim da classe Sort1
```

- **Exemplo com os algoritmos reverse, fill, copy, max e min**

```
public class Teste {
        private final Character[] letters = { 'F´, 'C', 'M' };
        private Character[] lettersCopy;
        private List<Character> list;
        private List<Character> copyList;

        // cria uma List e a manipula com métodos de Collections
        public Teste()
        {
         list = Arrays.asList( letters ); // obtém List
         lettersCopy = new Character[ 3 ];
        copyList = Arrays.asList( lettersCopy ); // visualização
         de lista de lettersCopy
        System.out.println( "Initial list: " );
        output( list );
```

- **Exemplo com os algoritmos reverse, fill, copy, max e min**

```
Collections.reverse( list ); // inverte a ordem
System.out.println( "\nAfter calling reverse: " );
output( list )
Collections.copy( copyList, list ); // copia List
System.out.println( "\nAfter copying: " );
output( copyList );
Collections.fill( list, ‘R’ ); // preenche a lista com Rs
System.out.println( "\nAfter calling fill: " );
output( list );
} // fim do construtor Teste
```

- **Exemplo com os algoritmos reverse, fill, copy, max e min**

```
// gera saída de informações de List
private void output( List< Character > listRef )
{
System.out.print( "The list is: " );

for ( Character element : listRef )
System.out.printf( "%s ", element );

System.out.printf( "\nMax: %s", Collections.max( listRef ) );
System.out.printf( " Min: %s\n", Collections.min( listRef ) );
} // fim do método output

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

- **Exemplo com o algoritmo binarySearch**

```
public class BinarySearchTest {
 private static final String colors[] = {"red", "white", "blue",
"black", "yellow", "purple", "tan", "pink"};
private List<String> list; // referência ArrayList
// cria, classifica e gera saída da lista
public BinarySearchTest()
{
 list = new ArrayList< String >( Arrays.asList( colors ) );
 Collections.sort( list ); // classifica a ArrayList
 System.out.printf( "Sorted ArrayList: %s\n", list );
 } // fim do construtor BinarySearchTest
```

Exemplos

- **Exemplo com o algoritmo binarySearch**

```
// pesquisa vários valores na lista
private void search()
{
 printSearchResults( colors[ 3 ] ); // primeiro item
 printSearchResults( colors[ 0 ] ); // item intermediário
 printSearchResults( colors[ 7 ] ); // último item
 printSearchResults( "aqua" ); // abaixo do mais baixo
 printSearchResults( "gray" ); // não existe
 printSearchResults( "teal" ); // não existe
 } // fim do método search
```

- **Exemplo com o algoritmo binarySearch**

```
// realiza pesquisas e exibe resultado da pesquisa
private void printSearchResults( String key )
{
 int result = 0;
 System.out.printf( "\nSearching for: %s\n", key );
 result = Collections.binarySearch( list, key );

 if ( result >= 0 )
 System.out.printf( "Found at index %d\n", result );
 else
 System.out.printf( "Not Found (%d)\n",result );
 } // fim do método printSearchResults

public static void main( String args[] )
{
BinarySearchTest binarySearchTest = new BinarySearchTest();
binarySearchTest.search();
} // fim de main
} // fim da classe BinarySearchTest
```

Exemplos

- **Com iterador**

```
public class MyCollections {
        public static void main(String[] args) {
                List<Integer> myList = new ArrayList<Integer>();
                myList.add(1);
                myList.add(2);
                myList.add(2);

                for (Integer listElements : myList) {
                        System.out.println(listElements);
                }
```

Exemplos

- **Com iterador**

```
                Set<Integer> mySet = new HashSet<Integer>();
                mySet.add(1);
                mySet.add(2);
                mySet.add(3);
                mySet.add(1);
                Iterator<Integer> iMySet = mySet.iterator();
                while(iMySet.hasNext()){
                        System.out.println(iMySet.next());
                }
        }
}
```

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

Coleções
Parte 2