ESTRITAMENTE **CONFIDENCIAL**

ELABORATA
INFORMÁTICA

LINGUAGEM JAVA

Ordenação

Descrição

- A ordenação de itens é a ação de ordenar itens segundo um critério de ordem pré-definido. Na Linguagem Java é possível ordenar números, strings e objetos utilizando recursos fornecidos pela API Collections.

Tipos de Ordenação

- Ordenação de números
- Ordenação de strings
- Ordenação de objetos

Ordenação de Números

- Números armazenados em vetores podem ser ordenados através do método Array.sort().

Ordenação de números

```
...
int numeros[] = {3, 1, 4, 2, 5, 0, 9, 7, 6, 8};

Arrays.sort(numeros);

for (int i=0; i < numeros.length; i++)
  System.out.print(numeros[i] + " ");
...
```

Ordenação decrescente

- Para ordenar números em ordem decrescente, o vetor de números precisa ser declarado como Integer e não com o tipo primitivo int.

Ordenação decrescente

```
...
Integer numeros[] = {3, 1, 4, 2, 5, 0, 9, 7, 6, 8};

Arrays.sort(numeros, Collections.reverseOrder());

for (int i=0; i < numeros.length; i++)
  System.out.print(numeros[i] + " ");
...
```

Ordenação de Strings

- Strings armazenadas em listas podem ser ordenadas através do método Collections.sort().

Ordenação de Strings

```
...
List<String> bairros = new ArrayList<String>();

bairros.add("Capão Raso");
bairros.add("Pinheirinho");
bairros.add("Agua Verde");
bairros.add("Batel");

Collections.sort(bairros);

for (String b : bairros)
  System.out.println(b);
...
```

Ordenação decrescente

```
...
List<String> bairros = new ArrayList<String>();

bairros.add("Capão Raso");
bairros.add("Pinheirinho");
bairros.add("Agua Verde");
bairros.add("Batel");

Collections.sort(bairros, Collections.reverseOrder());

for (String b : bairros)
  System.out.println(b);
...
```

Ordenação de Objetos

- Objetos armazenadas em listas podem ser ordenados através da implementação da interface Comparable.

Ordenação de Objetos

```
public class Aluno
{
  private int matricula;
  private String nome;
  private char sexo;

  public Aluno() {}

  public Aluno(int matricula, String nome, char sexo)
  {
    this.matricula = matricula;
    this.nome = nome;
    this.sexo = sexo;
  }

  // Métodos getters e setters
}
```

Ordenação de Objetos

```
...
List<Aluno> alunos = new ArrayList<Aluno>();

alunos.add(new Aluno(1004, "Francisco Alvarenga", 'M'));
alunos.add(new Aluno(1001, "Amália Silveira Silva", 'F'));
alunos.add(new Aluno(1002, "Danusa Pedrosa", 'F'));

Collections.sort(alunos);

for (Aluno a : alunos)
  System.out.println(a.getMatricula() + ", " + a.getNome());
...
```

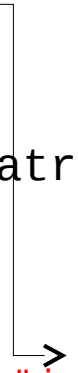
Ordenação de Objetos

```
...
List<Aluno> alunos = new ArrayList<Aluno>();

alunos.add(new Aluno(1004, "Francisco Alvarenga", 'M'));
alunos.add(new Aluno(1001, "Amália Silveira Silva", 'F'));
alunos.add(new Aluno(1002, "Danusa Pedrosa", 'F'));

Collections.sort(alunos);

for (Aluno a : alunos)
  System.out.println(a.getMatricula() + ", " + a.getNome());
...
```



```
Exception in thread "main" java.lang.ClassCastException:
collections.comparable.Aluno cannot be cast to java.lang.Comparable
at java.util.Arrays.mergeSort(Unknown Source)
at java.util.Arrays.sort(Unknown Source)
at java.util.Collections.sort(Unknown Source)
```

Solução???

- Implementar Interface Comparable

Interface Comparable

```
public class Aluno implements Comparable<Aluno>
{
  private int matricula;
  private String nome;
  private char sexo;

  // Construtores

  // Métodos getters e setters

  @Override
  public int compareTo(Aluno aluno) {
    return this.nome.compareTo(aluno.getNome());
  }
}
```

Interface Comparable

```
...
List<Aluno> alunos = new ArrayList<Aluno>();

alunos.add(new Aluno(1002, "Francisco Alvarenga", 'M'));
alunos.add(new Aluno(1004, "Amália Silveira Silva", 'F'));
alunos.add(new Aluno(1001, "Danusa Pedrosa", 'F'));

Collections.sort(alunos);

for (Aluno a : alunos)
  System.out.println(a.getMatricula() + ", " + a.getNome());
...
```

Situação

- Como ordenar a lista de alunos pela matrícula?

Como a **Interface Comparable** permite apenas um tipo de ordenação, será necessário modificar o método **compareTo**.

Solução???

- Implementar a Interface Comparator

Interface Comparator

```
public class Aluno
{
  private int matricula;
  private String nome;
  private char sexo;

  public Aluno() {}

  public Aluno(int matricula, String nome, char sexo)
  {
    this.matricula = matricula;
    this.nome = nome;
    this.sexo = sexo;
  }

  // Métodos getters e setters
}
```

Interface Comparator

```
public class AlunoOrdenarNome implements Comparator<Aluno>
{
  @Override
  public int compare(Aluno aluno1, Aluno aluno2)
  {
    return aluno1.getNome().compareTo(aluno2.getNome());
  }

}
```

Interface Comparator

```
public class AlunoOrdenarMatricula implements
Comparator<Aluno>
{
  @Override
  public int compare(Aluno aluno1, Aluno aluno2)
  {
    return aluno1.getMatricula() - aluno2.getMatricula();
  }
}
```

Interface Comparator

```
public class ComparatorSample
{
  public static void main(String[] args)
  {
    List<Aluno> alunos = new ArrayList<Aluno>();

    alunos.add(new Aluno(1002, "Francisco Alvarenga", 'M'));
    alunos.add(new Aluno(1004, "Amália Silveira Silva", 'F'));
    alunos.add(new Aluno(1001, "Danusa Pedrosa", 'F'));

    Collections.sort(alunos, new AlunoOrdenarMatricula());

    for (Aluno a : alunos)
      System.out.println(a.getMatricula() + ", " + a.getNome());
  }
}
```

OBRIGADO!

ESTRITAMENTE **CONFIDENCIAL**

ELABORATA
INFORMÁTICA

www.**elaborata**.com.br

Horário de Atendimento Comercial
Segunda à sexta – das 9:00h às 19:30h e
Sábado - das 8:00h às 15:00h.

Rua Monsenhor Celso, 256 - 1º Andar
Centro - Curitiba - PR

41.3324.0015
41.99828.2468

cursos@elaborata.com.br