Utilidade das Interfaces

Consideremos que pretendíamos ordenar objetos pertencentes a uma classe Empregado, caracterizados por dois atributos, nome e vencimento, existentes num contentor, por exemplo num *array*.

Classe Empregado:

```
package interfaces;

public class Empregado {

    private String nome;
    private float vencimento;

    public Empregado(String nome, float vencimento) {
         this.nome = nome;
         this.vencimento = vencimento;
    }
    public Empregado() {
         this("sem nome",0);
    }
    public String getNome() {
         return this.nome;
    }
    public float getVencimento() {
         return this.vencimento;
    }

    public void setNome(String nome) {
         this.nome = nome;
    }

    public void setVencimento(float vencimento) {
         this.vencimento = vencimento;
     }

     public String toString() {
         return this.nome + " tem o vencimento de " + this.vencimento;
     }
}
```

Poderíamos usar a seguinte classe de teste.

Classe Teste:

```
package interfaces;

public class Teste {

    public static void main(String[] args) {

        Empregado[] a = new Empregado[3];

        a[0] = new Empregado("Miguel", 200.0f);
        a[1] = new Empregado("João", 400.0f);
        a[2] = new Empregado("António", 250.0f);

        for (int i = 0; i < a.length; i++) {
          System.out.println( a[i] );
        }
     }
}
```

Saída produzida pelo programa:

```
Miguel tem o vencimento de 200.0
João tem o vencimento de 400.0
António tem o vencimento de 250.0
```

Em Java existe a **interface Comparable** (java.lang.Comparable) que declara o método:

`public int compareTo(Object obj)`

este método compara o objeto ao qual é aplicado com o objeto *obj* e retorna um inteiro negativo, zero, ou positivo conforme o objeto ao qual é aplicado é menor, igual, ou maior que o objeto *obj*.

A **classe Arrays** (java.util.Arrays) contém exclusivamente métodos estáticos que operam ou retornam *arrays*.

Entre outros possui o seguinte método:

`public static void sort(Object[] a)`

este método ordena os elementos do *array a* por ordem crescente, mas é necessário que todos os elementos do *array* implementem a interface *Comparable*, e todos os elementos possam ser comparáveis entre si.

Versão 1:

Ordenação por vencimentos (classes Empregado1 e Teste1).

Numa 1.ª versão poderíamos modificar a classe Empregado para tornar os objetos comparáveis entre si. Vamos supor que pretendíamos ordenar por *vencimento*. Designemos por Empregado1 a nova classe.

Classe Empregado1:

```
package interfaces;

public class Empregado1 implements Comparable {

    private String nome;
    private float vencimento;

    public Empregado1(String nome, float vencimento) {
         this.nome = nome;
         this.vencimento = vencimento;
    }
    public Empregado1() {
         this("sem nome",0);
    }
    public String getNome() { return this.nome; }
    public float getVencimento() { return this.vencimento; }

    public void setNome(String nome) {
         this.nome = nome;
    }

    public void setVencimento(float vencimento) {
         this.vencimento = vencimento;
     }

     public String toString() {
         return this.nome + " tem o vencimento de " + this.vencimento;
     }

     public int compareTo(Object obj) {
         Empregado1 e = (Empregado1) obj;
         return (int) (this.vencimento - e.vencimento);
     }
}
```

A ordem explicitamente codificada dentro da classe, através do método *compareTo()*, designa-se por ***ordem natural*** dos objetos.

Poderíamos usar a seguinte classe de teste.

Classe Teste1:

```
package interfaces;

import java.util.Arrays;

public class Teste1 {

    public static void main(String[] args) {

        Empregado1[] a = new Empregado1[3];

        a[0] = new Empregado1("Miguel", 200.0f);
        a[1] = new Empregado1("João", 400.0f);
        a[2] = new Empregado1("António", 250.0f);

        for (int i = 0; i < a.length; i++) {
          System.out.println( a[i] );
        }

        Arrays.sort(a);

        System.out.println("---------------");

        for (int i = 0; i < a.length; i++) {
          System.out.println( a[i] );
        }
     }
}
```

Saída produzida pelo programa:

```
Miguel tem o vencimento de 200.0
João tem o vencimento de 400.0
António tem o vencimento de 250.0
---------------
Miguel tem o vencimento de 200.0
António tem o vencimento de 250.0
João tem o vencimento de 400.0
```

Versão 2:

Ordenação por nomes (classes Empregado2 e Teste2).

Se agora quiséssemos ordenar os objetos por *nome* poderíamos criar uma nova classe Empregado2 com um método de ordenação diferente.

Classe Empregado2:

```
package interfaces;

public class Empregado2 implements Comparable {

    private String nome;
    private float vencimento;

    public Empregado2(String nome, float vencimento) {
         this.nome = nome;
         this.vencimento = vencimento;
    }
    public Empregado2() {
         this("sem nome",0);
    }
    public String getNome() { return this.nome; }
    public float getVencimento() { return this.vencimento; }

    public void setNome(String nome) {
         this.nome = nome;
    }

    public void setVencimento(float vencimento) {
         this.vencimento = vencimento;
     }

     public String toString() {
         return this.nome + " tem o vencimento de " + this.vencimento;
     }

     public int compareTo(Object obj) {
         Empregado2 e = (Empregado2) obj;
         return this.nome.compareTo(e.nome);
     }
}
```

Na última linha de código, o método *compareTo(String)* é aplicado a um objeto String (nome). Trata-se portanto do método *compareTo()* da classe String.

Poderíamos usar a seguinte classe de teste.

Classe Teste2:

```
package interfaces;

import java.util.Arrays;

public class Teste2 {
    public static void main(String[] args) {

        Empregado2[] a = new Empregado2[3];

        a[0] = new Empregado2("Miguel", 200.0f);
        a[1] = new Empregado2("João", 400.0f);
        a[2] = new Empregado2("António", 250.0f);

        for (int i = 0; i < a.length; i++) {
          System.out.println( a[i] );
        }

        Arrays.sort(a);

        System.out.println("---------------");

        for (int i = 0; i < a.length; i++) {
          System.out.println( a[i] );
        }
     }
}
```

Saída produzida pelo programa:

```
Miguel tem o vencimento de 200.0
João tem o vencimento de 400.0
António tem o vencimento de 250.0
---------------
António tem o vencimento de 250.0
João tem o vencimento de 400.0
Miguel tem o vencimento de 200.0
```

Versão 3:

Ordenação por vencimentos (classes Empregado, EmpregadoVencimento e Teste3). Uma solução melhor consistiria em obter objetos comparáveis entre si **sem modificar** a classe original Empregado. Vamos supor que pretendíamos ordenar por *vencimento*. Criámos uma subclasse de Empregado, *EmpregadoVencimento*, para obter objetos ordenáveis por vencimento.

Classe EmpregadoVencimento:

```
package interfaces;
public class EmpregadoVencimento extends Empregado implements Comparable {
    public EmpregadoVencimento(String nome, float vencimento) {
        super(nome, vencimento);
    }
    public int compareTo(Object obj) {
        EmpregadoVencimento e = (EmpregadoVencimento) obj;
        return (int) (this.getVencimento() - e.getVencimento());
    }
}
```

Classe Teste3:

```
package interfaces;
import java.util.Arrays;
public class Teste3 {
    public static void main(String[] args) {

        EmpregadoVencimento[] a = new EmpregadoVencimento[3];

        a[0] = new EmpregadoVencimento("Miguel", 200.0f);
        a[1] = new EmpregadoVencimento("João", 400.0f);
        a[2] = new EmpregadoVencimento("António", 250.0f);

        for (int i = 0; i < a.length; i++){
          System.out.println( a[i] );
        }

        Arrays.sort(a);
        System.out.println("---------------");
        for (int i = 0; i < a.length; i++) {
          System.out.println( a[i] );
        }
     }
}
```

Saída produzida pelo programa:

```
Miguel tem o vencimento de 200.0
João tem o vencimento de 400.0
António tem o vencimento de 250.0
---------------
Miguel tem o vencimento de 200.0
António tem o vencimento de 250.0
João tem o vencimento de 400.0
```

Versão 4:

Ordenação por nomes (classes Empregado, EmpregadoNome e Teste4).

Para obter objetos comparáveis, ordenados por *nome*, **sem modificar** a classe original Empregado, criámos uma subclasse de Empregado, *EmpregadoNome*.

Classe EmpregadoNome:

```
package interfaces;
public class EmpregadoNome extends Empregado implements Comparable {

    public EmpregadoNome(String nome, float vencimento) {
        super(nome, vencimento);
    }

    public int compareTo(Object obj) {
        EmpregadoNome e = (EmpregadoNome) obj;
        return this.getNome().compareTo(e.getNome());
    }
}
```

Classe Teste4:

```
package interfaces;

import java.util.Arrays;

public class Teste4 {

    public static void main(String[] args) {

        EmpregadoNome[] a = new EmpregadoNome[3];

        a[0] = new EmpregadoNome("Miguel", 200.0f);
        a[1] = new EmpregadoNome("João", 400.0f);
        a[2] = new EmpregadoNome("António", 250.0f);

        for (int i = 0; i < a.length; i++){
          System.out.println( a[i] );
        }

        Arrays.sort(a);
        System.out.println("---------------");
        for (int i = 0; i < a.length; i++) {
          System.out.println( a[i] );
        }
    }
}
```

Saída produzida pelo programa:

```
Miguel tem o vencimento de 200.0
João tem o vencimento de 400.0
António tem o vencimento de 250.0
---------------
António tem o vencimento de 250.0
João tem o vencimento de 400.0
Miguel tem o vencimento de 200.0
```

Qualquer uma das soluções anteriores obriga a criar novas classes para obter objetos ordenáveis por um dado critério, e a criar e usar objetos desses tipos de dados.

Vamos construir outras soluções mais flexíveis.

Em Java existe a **interface Comparator** (java.lang.Comparator) que declara o método:

`public int compare(Object obj1, Object obj2)`

este método compara a ordem dos seus dois argumentos. Retorna um inteiro negativo, zero, ou positivo conforme o primeiro argumento *obj1* é menor, igual, ou maior que o segundo *obj2*.

A **classe Arrays** (java.util.Arrays), que contém exclusivamente métodos estáticos que operam ou retornam *arrays*, possui entre outros, os seguintes métodos:

`public static void sort(Object[] a)`

este método ordena os elementos do *array a* por ordem crescente, mas é necessário que todos os elementos do *array* implementem a interface *Comparable*, e todos os elementos possam ser comparáveis entre si.

`public static void sort(Object[] a, Comparator c)`

este método ordena os elementos do *array* a pela ordem induzida pelo comparador c.

Versão 5:

Ordenação por vencimentos (classes Empregado, ComparadorVencimento e Teste5).

Nesta solução também vamos obter objetos comparáveis entre si **sem modificar** a classe original Empregado. Para além disso, os objetos que vamos criar serão objetos da classe original Empregado. Para ordenar por *vencimento* vamos criar uma classe, *ComparadorVencimento*, que implementa a interface *Comparator*, contendo apenas o método de ordenação *compare()*.

Classe ComparadorVencimento:

```
package interfaces;

import java.util.Comparator;

public class ComparadorVencimento implements Comparator {

    public int compare(Object obj1, Object obj2) {
        Empregado e1 = (Empregado) obj1;
        Empregado e2 = (Empregado) obj2;
        return (int) (e1.getVencimento() – e2.getVencimento());
    }
}
```

Classe Teste5:

```
package interfaces;

import java.util.Arrays;

public class Teste5 {
    public static void main(String[] args) {
        Empregado[] a = new Empregado[3];

        a[0] = new Empregado("Miguel", 200.0f);
        a[1] = new Empregado("João", 400.0f);
        a[2] = new Empregado("António", 250.0f);

        for (int i = 0; i < a.length; i++){ System.out.println( a[i] ); }

        ComparadorVencimento c1 = new ComparadorVencimento();
        Arrays.sort(a, c1);

        System.out.println("---------------");
        for (int i = 0; i < a.length; i++) { System.out.println( a[i] ); }
     }
}
```

Saída produzida pelo programa:

```
Miguel tem o vencimento de 200.0
João tem o vencimento de 400.0
António tem o vencimento de 250.0
---------------
Miguel tem o vencimento de 200.0
António tem o vencimento de 250.0
João tem o vencimento de 400.0
```

Ao objeto ComparadorVencimento demos o nome de "c1". Mas como tem um único uso, poderia ser **anónimo**.

Classe Teste5:

```
package interfaces;

import java.util.Arrays;

public class Teste5 {

    public static void main(String[] args) {

        Empregado[] a = new Empregado[3];

        a[0] = new Empregado("Miguel", 200.0f);
        a[1] = new Empregado("João", 400.0f);
        a[2] = new Empregado("António", 250.0f);

        for (int i = 0; i < a.length; i++){
          System.out.println( a[i] );
        }

        Arrays.sort(a, new ComparadorVencimento());

        System.out.println("---------------");
        for (int i = 0; i < a.length; i++) {
          System.out.println( a[i] );
        }
    }
}
```

Versão 6:

Ordenação por nomes (classes Empregado, ComparadorNome e Teste6).

Para ordenar por *nome* vamos criar uma classe, *ComparadorNome*, que implementa a interface *Comparator*, contendo apenas o método de ordenação *compare()*.

Classe ComparadorNome:

```
package interfaces;

import java.util.Comparator;

public class ComparadorNome implements Comparator {

    public int compare(Object obj1, Object obj2) {
        Empregado e1 = (Empregado) obj1;
        Empregado e2 = (Empregado) obj2;
        return (int) (e1.getNome().compareTo(e2.getNome()));
    }
}
```

Classe Teste6:

```
package interfaces;

import java.util.Arrays;

public class Teste6 {

    public static void main(String[] args) {

        Empregado[] a = new Empregado[3];

        a[0] = new Empregado("Miguel", 200.0f);
        a[1] = new Empregado("João", 400.0f);
        a[2] = new Empregado("António", 250.0f);

        for (int i = 0; i < a.length; i++){
          System.out.println( a[i] );
        }

        Arrays.sort(a, new ComparadorNome());

        System.out.println("---------------");
        for (int i = 0; i < a.length; i++) { System.out.println( a[i] ); }
     }
}
```

Saída produzida pelo programa:

```
Miguel tem o vencimento de 200.0
João tem o vencimento de 400.0
António tem o vencimento de 250.0
---------------
António tem o vencimento de 250.0
João tem o vencimento de 400.0
Miguel tem o vencimento de 200.0
```

As classes que contêm os métodos de ordenação, ComparadorVencimento e ComparadorNome, têm um único uso: o de criar um objeto que contém o método de ordenação. Classes que só são usadas uma única vez para criar um objeto podem ser definidas como **classes anónimas**.

Relativamente à classe ComparadorVencimento,

```
import java.util.Comparator;

public class ComparadorVencimento implements Comparator {

    public int compare(Object obj1, Object obj2) {
        Empregado e1 = (Empregado) obj1;
        Empregado e2 = (Empregado) obj2;
        return (int) (e1.getVencimento() – e2.getVencimento());
    }
}
```

A criação de um objeto desta classe pode ser feita com a seguinte sintaxe:

```
Comparator c1 = new  Comparator() {

     public int compare(Object obj1, Object obj2) {
           Empregado e1 = (Empregado) obj1;
           Empregado e2 = (Empregado) obj2;
           return (int) (e1.getVencimento() - e2.getVencimento());
     }
};
```

Relativamente à classe ComparadorNome,

```
import java.util.Comparator;

public class ComparadorNome implements Comparator {

    public int compare(Object obj1, Object obj2) {
        Empregado e1 = (Empregado) obj1;
        Empregado e2 = (Empregado) obj2;
        return (int) (e1.getNome().compareTo(e2.getNome()));
    }
}
```

A criação de um objeto desta classe pode ser feita com a seguinte sintaxe:

```
Comparator c1 = new  Comparator() {

     public int compare(Object obj1, Object obj2) {
           Empregado e1 = (Empregado) obj1;
           Empregado e2 = (Empregado) obj2;
           return (int) (e1.getNome().compareTo(e2.getNome()));
     }
};
```

Assim, a solução final para ordenação de objetos Empregado poderia ser obtida usando apenas a classe original Empregado e uma classe de teste, na qual para cada critério de ordenação se explicita o modo de ordenação através de um objeto *Comparator* de uma classe anónima.

Versão 7:

Ordenação por vencimentos e nomes (classes Empregado e Teste7).

Classe Teste7:

```
package interfaces;

import java.util.Arrays;

public class Teste7 {

    public static void main(String[] args) {

        Empregado[] a = new Empregado[3];

        a[0] = new Empregado("Miguel", 200.0f);
        a[1] = new Empregado("João", 400.0f);
        a[2] = new Empregado("António", 250.0f);

        for (int i = 0; i < a.length; i++){ System.out.println( a[i] ); }

        Arrays.sort(a, new Comparator() {
          public int compare(Object obj1, Object obj2) {
                Empregado e1 = (Empregado) obj1;
                Empregado e2 = (Empregado) obj2;
                return (int) (e1.getVencimento() - e2.getVencimento());
          }
        });

        System.out.println("---------------");
        for (int i = 0; i < a.length; i++) { System.out.println( a[i] ); }

        Arrays.sort(a, new Comparator() {
          public int compare(Object obj1, Object obj2) {
                Empregado e1 = (Empregado) obj1;
                Empregado e2 = (Empregado) obj2;
                return (int) (e1.getNome().compareTo(e2.getNome()));
          }
        });

        System.out.println("---------------");
        for (int i = 0; i < a.length; i++) { System.out.println( a[i] ); }

     }
}
```

Saída produzida pelo programa:

```
Miguel tem o vencimento de 200.0
João tem o vencimento de 400.0
António tem o vencimento de 250.0
---------------
Miguel tem o vencimento de 200.0
António tem o vencimento de 250.0
João tem o vencimento de 400.0
---------------
António tem o vencimento de 250.0
João tem o vencimento de 400.0
Miguel tem o vencimento de 200.0
```

Em resumo, vimos 2 maneiras de **ordenar objetos:**

1. Usando o método

   ```
   public static void Arrays.sort(Object[] arrayDeObjetos)
   ```

   os objetos do *arrayDeObjetos* têm de implementar a interface *Comparable*, e implementar o método *compareTo(Object o)*.

2. Mas para ordenar objetos que não implementam a interface *Comparable* é necessário fornecer um objeto *Comparator* – objeto que encapsula uma ordenação através do método *compare()*.

   ```
   public static void Arrays.sort(Object[] arrayDeObjetos, Comparator c)
   ```

   O método *compare()* compara os seus dois argumentos. Se qualquer um destes argumentos tem um tipo inapropriado para o *Comparator*, o método *compare* lança a exceção *ClassCastException*.

   Escrever um método *compare* é aproximadamente idêntico a escrever o método *compareTo*, exceto que o primeiro recebe ambos os objetos passados como argumentos.

   Se for necessário efetuar várias ordenações de uma mesma lista de objetos, em que cada ordenação usa um critério diferente, o modo mais adequado é através da interface *Comparator*:

   - Define-se um critério de ordenação através de um método

     int compare(Object obj1, Object obj2)

   - Cria-se um objeto de uma classe anónima que implemente a interface *Comparator* na qual se reescreve o método *compare()* com o critério de ordenação desejado.
   - Ordena-se usando o método *Arrays.sort(Object[] arrayDeObjetos, Comparator c1)*.