Interfaces

POO

Prof. Marcio Delamaro

O que é interface

- É um template de classe
- Outras classes podem seguir esse template
- Chamamos isso de “implementar” a interface
- Isso permite que classes que não pertencem a uma mesma hierarquia sejam tratados como iguais
- Herança múltipla (???)

Exemplo banal

```
public interface FiguraGeometrica {
    public String getNomeFigura();
    public int getArea();
    public int getPerimetro();
}
```

Exemplo banal

```
public interface FiguraGeometrica {
    public String getNomeFigura();
    public int getArea();
    public int getPerimetro();
}
```

- Ao definir a interface, estamos definindo um “contrato” que outras classes vão ter que implementar.

Exemplo banal

```
public interface FiguraGeometrica {
    public String getNomeFigura();
    public int getArea();
    public int getPerimetro();
}
```

- Ao definir a interface, estamos definindo um “contrato” que outras classes vão ter que implementar.
- Note que nenhum dos métodos tem uma implementação.

Implementando

- public class Quadrado implements FiguraGeometrica
  public class Circulo implements FiguraGeometrica

Implementando

- public class Quadrado implements FiguraGeometrica
  public class Circulo implements FiguraGeometrica

- Significa que essa classe tem que implementar métodos definidos na interface
- Objetos do tipo Quadrado e Círculo podem ser tratados como objetos do tipo FiguraGeometrica

Quadrado

```
public class Quadrado implements FiguraGeometrica {
    private int lado;
    public Quadrado (int l) {
        lado = l;
    }
public int getLado() {
        return lado;
}
@Override
public int getArea() {
        int area = 0;
        area = lado * lado;
        return area;
}
@Override
public int getPerimetro() {
….
}
@Override
public String getNomeFigura() {
        return "quadrado";
}
}
```

Quadrado

```
public class Quadrado implements FiguraGeometrica {
    private int lado;
    public Quadrado (int l) {
        lado = l;
    }
public int getLado() {
        return lado;
}
@Override
public int getArea() {
        int area = 0;
        area = lado * lado;
        return area;
}
@Override
public int getPerimetro() {
….
}
@Override
public String getNomeFigura() {
        return "quadrado";
}
}
```

Implementações obrigatórias da interface.

Círculo

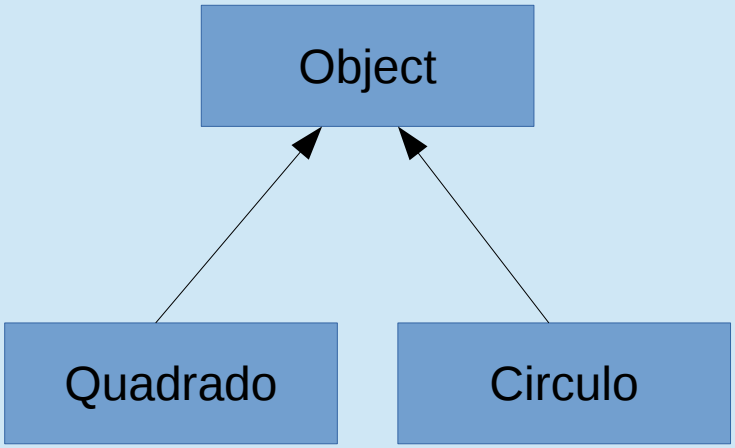
```
public class Circulo implements FiguraGeometrica {
private int raio;


public Circulo(int r) {
   raio = r;
}
public int getRaio() {
    return raio;
}
@Override
public String getNomeFigura() {
    return "Círculo";
}
@Override
public int getArea() {
    return (int) (3.14 * raio * raio);
}
@Override
public int getPerimetro() {
    return (int) (3.14 * raio * 2);
}
}
```

Círculo

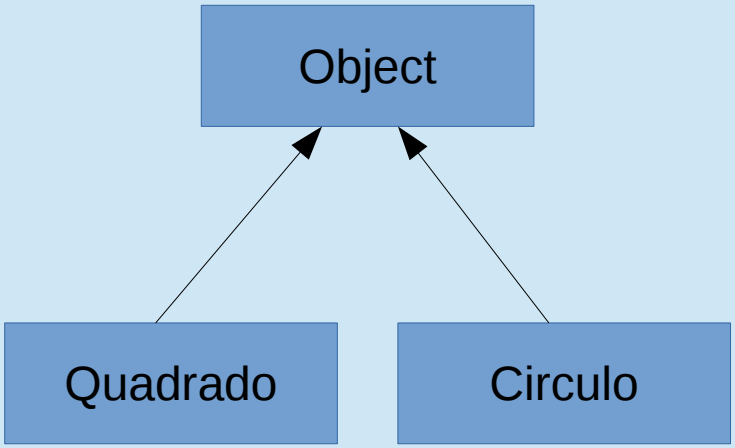
```
public class Circulo implements FiguraGeometrica {
private int raio;


public Circulo(int r) {
   raio = r;
}
public int getRaio() {
    return raio;
}
@Override
public String getNomeFigura() {
    return "Círculo";
}
@Override
public int getArea() {
    return (int) (3.14 * raio * raio);
}
@Override
public int getPerimetro() {
    return (int) (3.14 * raio * 2);
}
}
```

Implementações obrigatórias da interface.

Hierarquia

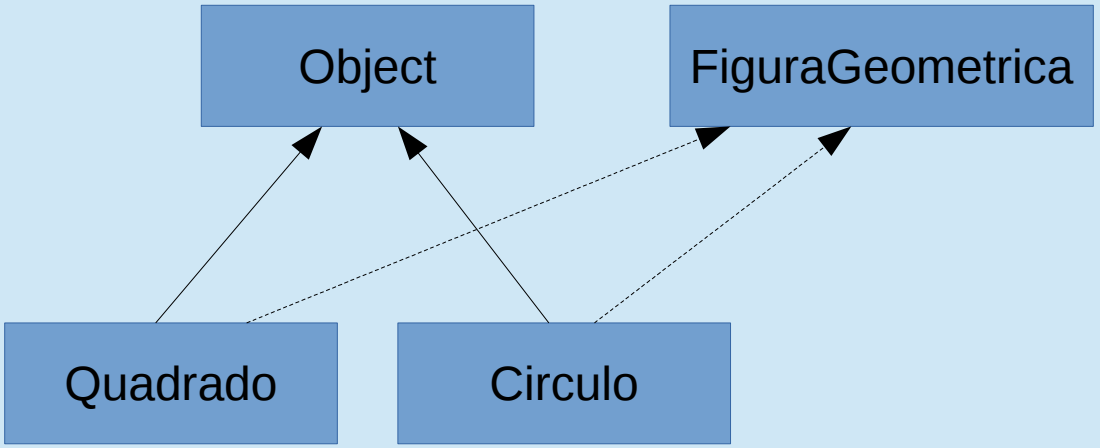


Hierarquia




```
Quadrado x = new Quadrado(10);
Circulo y = new Circulo(15);
x instanceof Quadrado // true
x instanceof Object // true
y instanceof Circulo // true
y instanceof Object // true
```

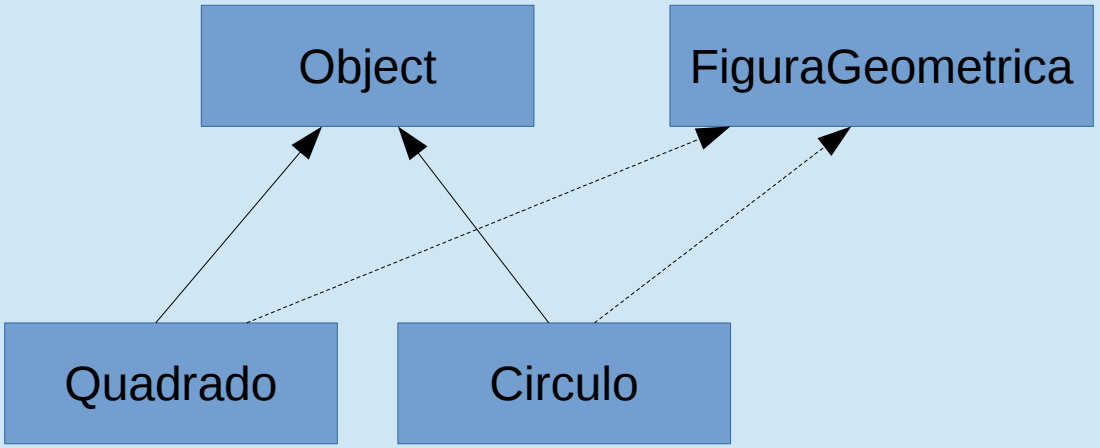
Hierarquia




```
Quadrado x = new Quadrado(10);
Circulo y = new Circulo(15);
x instanceof Quadrado // true
x instanceof Object // true
y instanceof Circulo // true
y instanceof Object // true
```

```
x instanceof FiguraGeometrica // true
y instanceof FiguraGeometrica // true
```

Hierarquia

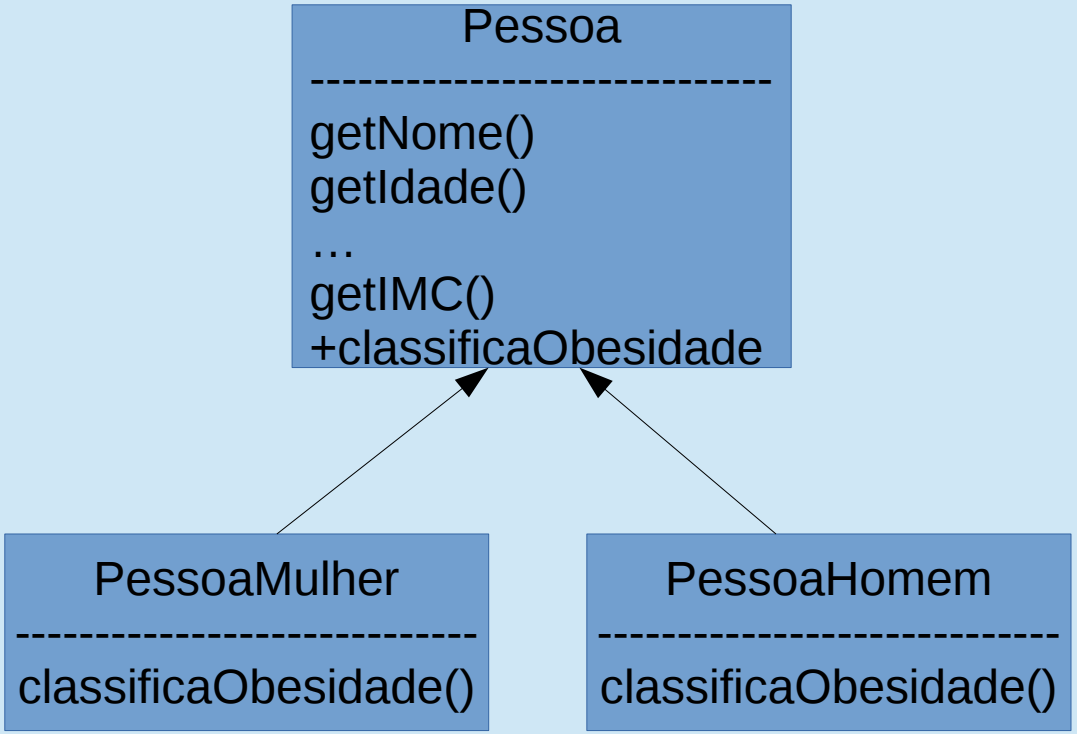



```
Quad                                                    // true
Circ                                                    // true
x in
x in
y in
y in
```

```
static public void main(String args[]) {
    FiguraGeometrica vet[] = new FiguraGeometrica[100];
    Quadrado x = new Quadrado(10);
    Circulo y = new Circulo(15);
    vet[0] = x;
    vet[1] = y;
}
```

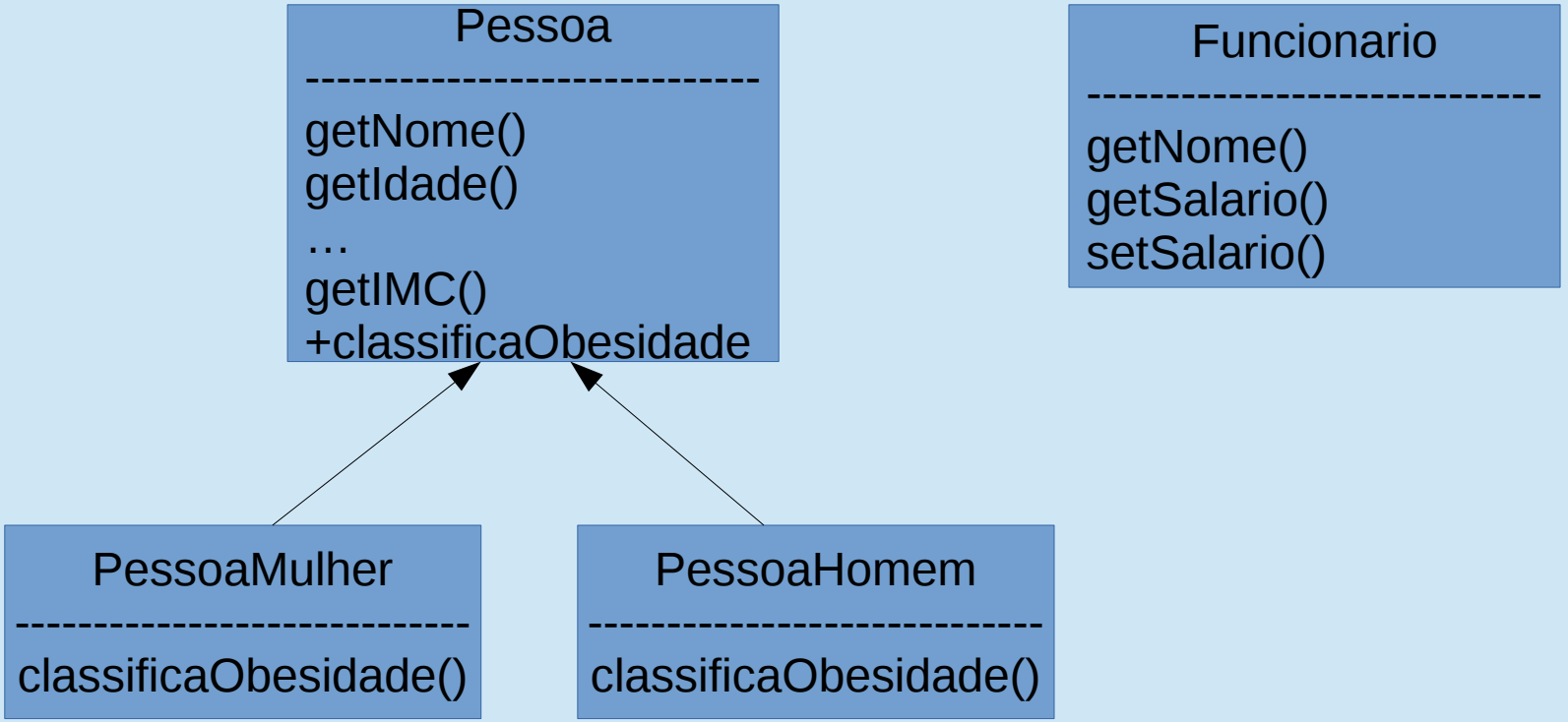
Pessoas na academia

- Vamos voltar ao sistema da acaddemia
- Temos as classes para representar as pessoas (alunos) que fazem parte da academia



Pessoas na academia

- Vamos adicionar uma classe para funcionários



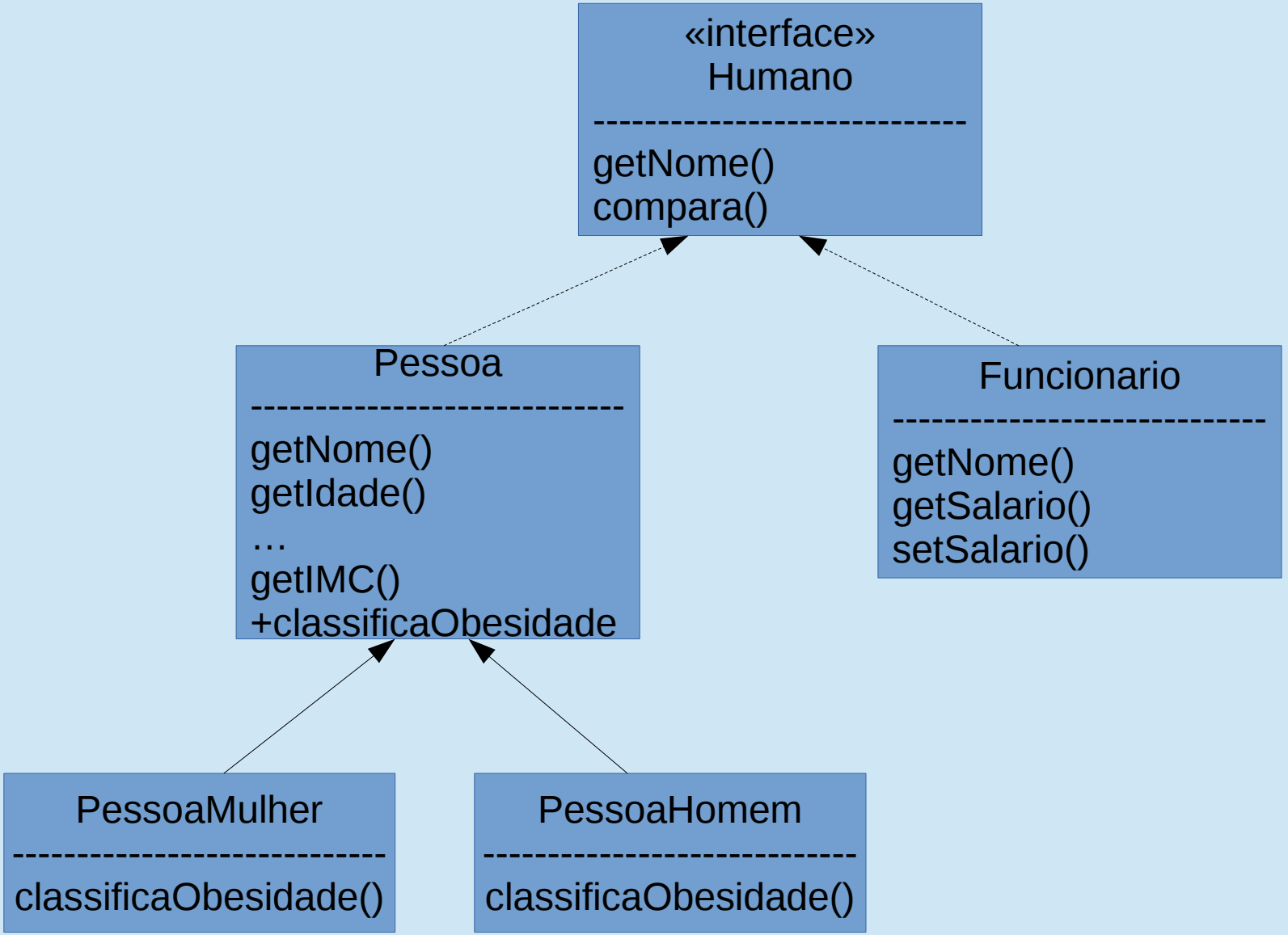
Hierarquia

- Não existe qualquer relação entre as duas classes Pessoas e Funcionarios
- Ou seja, não existe qualquer forma de polimorfismo entre objetos dessas classes

Hierarquia

- Não existe qualquer relação entre as duas classes Pessoas e Funcionarios
- Ou seja, não existe qualquer forma de polimorfismo entre objetos dessas classes
- Vamos agora supor que seja necessário comparar dois objetos “humanos”
- Por exemplo para saber se são o mesmo indivíduo (supondo que se possa saber pelo nome)

Hierarquia na academia



Interface Humano

```
public interface Humano {

public String getNome();

public boolean compara(Humano x);

}
```

Implementação Humano

```
public class Funcionario implements Humano {
private String nome;
private double salario;

    @Override
    public String getNome() {
        return nome;
    }


    @Override
    public boolean compara(Humano x) {
        String s = getNome();
        String r = x.getNome();
        if ( s == null ) return r == null;
        return s.equals(r);
    }
```

A classe Pessoa precisa ter uma implementação desses dois métodos também

Abstrata ou concreta

Implementação Humano

```
public class Funcionario implements Humano {
private String nome;
private double salario;

    @Override
    public String getNome() {
        return nome;
    }


    @Override
    public boolean compara(Humano x) {
        String s = getNome();
        String r = x.getNome();
        if ( s == null ) return r == null;
        return s.equals(r);
    }
```

A classe Pessoa precisa ter uma implementação desses dois métodos também

Abstrata ou concreta

Implementação Humano

```
public class Funcionario implements Humano {
private String nome;
private double salario;

    @Override
    public String getNome() {
        return nome;
    }


    @Override
    public boolean compara(Humano x) {
        String s = getNome();
        String r = x.getNome();
        if ( s == null ) return r == null;
        return s.equals(r);
    }
```

A classe Pessoa precisa ter uma implementação desses dois métodos também

Abstrata ou concreta

Dessa forma qualquer objeto que implemente a interface Humano pode ser comparado com um Funcionario. Ou com qualquer outro objeto que também implemente.

Algumas regras

- Interfaces podem ter atributos mas eles são sempre públicos e finais
- Interfaces podem ter métodos estáticos (Java 8)
- Interfaces podem ter implementação default (Java 8)

```
public interface Humano {
   default public String getNome() {
      return null;
   }

   default public boolean compara(Humano x) {
      return false;
   }
}
```

Usos de interfaces

- Muitas vezes a API java obriga o desenvolvedor a implementar interfaces
- Dessa forma ela pode usar os objetos criados
- Um exemplo: sistema de contas bancárias
- Ao final de cada iteração, vamos ordenar o array de contas

Sistema de contas

```
public static void main(String[] args) throws Exception {
int op = 0;
Contas ct = new Contas();
while (op != 8) {
   op = leOpcao();
   switch (op) {… }
   ct.ordena();
}
private void ordena() {
  Arrays.sort(contas);
}
```

Sistema de contas

```
public static void main(String[] args) throws Exception {
int op = 0;
Contas ct = new Contas();
while (op != 8) {
    op = leOpcao();
}
p
}
```

```
Exception in thread "main" java.lang.ClassCastException: PoupancaSimples cannot be cast to
java.lang.Comparable
        at
java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:320)
        at java.util.ComparableTimSort.sort(ComparableTimSort.java:188)
        at java.util.Arrays.sort(Arrays.java:1312)
        at Contas.ordena(Contas.java:102)
        at Contas.main(Contas.java:91)
```

Por que?

Por que?

- Declaração do nosso array de contas:
  ```
  private ContaBancaria contas[] = new ContaBancaria[100];
  ```
- Quando uma conta bancária é maior ou menor do que outra?

Por que?

- Declaração do nosso array de contas:
```
private ContaBancaria contas[] = new ContaBancaria[100];
```
- Quando uma conta bancária é maior ou menor do que outra?
- O método *Arrays.sort(Object)* não sabe

Por que?

- Declaração do nosso array de contas:
  ```
  private ContaBancaria contas[] = new ContaBancaria[100];
  ```
- Quando uma conta bancária é maior ou menor do que outra?
- O método *Arrays.sort(Object)* não sabe
- *public static void sort(Object[] a)*
  Sorts the specified array of objects into ascending order, according to the natural ordering of its elements.

Por que?

- Declaração do nosso array de contas:
  ```
  private ContaBancaria contas[] = new ContaBancaria[100];
  ```
- Quando uma conta bancária é maior ou menor do que outra?
- O método *Arrays.sort(Object)* não sabe
- *public static void sort(Object[] a)*
  Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. All elements in the array must implement the *Comparable* interface.

Por que?

- Declaração do nosso array de contas:
  ```
  private ContaBancaria contas[] = new ContaBancaria[100];
  ```
- Quando uma conta bancária é maior ou menor do que outra?
- O método *Arrays.sort(Object)* não sabe
- *public static void sort(Object[] a)*
  Sorts the specified array of objects into ascending order, according to the natural ordering of its elements.

Interface Comparable

int compareTo(T o)

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Interface Comparable

int compareTo(T o)

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

- Então, nós temos que definir para todas as classes que são ContaBancaria o método *compareTo*

Implementando Comparable

- `public abstract class ContaBancaria implements Comparable<ContaBancaria>`
- Ao fazer essa alteração, o compilador reclama que ContaEspecial, PoupancaOuro e PoupancaSimples precisam implementar compareTo(ContaBancaria)
- Podemos fazer isso nas subclasses ou na superclasse
- Depende do que consideramos ordenação de contas

Comparação Normal

- Vamos considerar que as contas são ordenadas pelo saldo. Saldo maior, significa conta maior.

Comparação Normal

- Vamos considerar que as contas são ordenadas pelo saldo. Saldo maior, significa conta maior.

ContaBancaria:

```
@Override
public int compareTo(ContaBancaria b) {
   double x = this.getSaldo();
   double y = b.getSaldo();
   if ( x == y ) return 0;
   if ( x < y ) return -1;
   return 1;
}
```

Comparação anormal

- Contas especiais são sempre menores que as outras
- Poupanças especias são sempre menores do que as poupanças simples
- Quando comparamos dois objetos do mesmo tipo, usamos o saldo para decidir quem é menor.

Comparação anormal

- Contas especiais são sempre menores que as outras
- Poupanças especias são sempre menores do que as poupanças simples
- Quando comparamos dois objetos do mesmo tipo, usamos o saldo para decidir quem é menor.
- OK, podem fazer!!!!