

## PG II

# Programação Orientada por Objectos em Java

### Definição de Classes:

- Resumo
- Métodos de instância
- toString()
- this
- Object
- Object.toString()
- Override (redefinir)
- Teste de igualdade entre objectos
- Object.equals()
- Ponto.equals()
- valueOf()
- javadoc

- Os programas em Java são desenvolvidos no âmbito de Classes;
- Para replicar a mesma **Estrutura** e **Comportamento** (i.e. Atributos e Métodos) em diversos **objectos**, que diferem entre si apenas nos valores dos seus atributos, então esses **atributos** e **métodos**, deverão ser de **instância**, (i.e. Classe Automovel, Ponto e Triangulo);
- Para criar um algoritmo que **não depende do contexto de cada instância** (i.e. NumMax e Arrays) então os **atributos** e **métodos** que formam esse algoritmo deverão ser declarados no ambiente de classe – **static**;
- Tipicamente numa classe podem ser encontrados:

Variáveis de classe

Variáveis de instância

Construtor

Método Estático

Método de Instância

```
package aula03;
public class Ponto {
    private static int numPontos;
    public int cordX, cordY;
    public Ponto(int x, int y) {
        cordX = x; cordY = y;
        numPontos++;
    }
    public static int getNumPontos() {
        return numPontos;
    }
    public double distancia(Ponto p) {
        double dx = cordX - p.cordX;
        double dy = cordY - p.cordY;
        return Math.sqrt( dx*dx + dy*dy );
    }
}
```

# Métodos de instância

Cada uma das classes desenvolvidas até ao momento apresenta o seu conjunto de métodos próprios, que caracterizam o seu **comportamento**, exemplo:

- Automovel → velocidadeMax();
- Ponto → distancia(Ponto p);
- Triangulo → perimetro();

Além dos métodos característicos de cada classe existem outros métodos que usualmente se encontram em qualquer classe e que disponibilizam serviços de grande utilidade.

Para se obter o resultado que a seguir se apresenta é necessário implementar o seguinte código:

```
package aula04;
import aula03.Ponto;
import pg2.io.IO;
public class Teste1{
    public static void main(String [] args){
        Ponto p1 = new Ponto(2,3), p2 = new Ponto(5,7);
        IO.cout.writeln("\nDistancia entre o ponto (" + p1.cordX + "," + p1.cordY +
            ") e o ponto (" + p2.cordX + "," + p2.cordY + ") = " +
            p1.distancia(p2));
    }
}
```

C:\ Command Prompt

```
D:\work>java aula04.Teste1

Distancia entre o ponto <2,3> e o ponto <5,7> = 5.0
```

# toString()

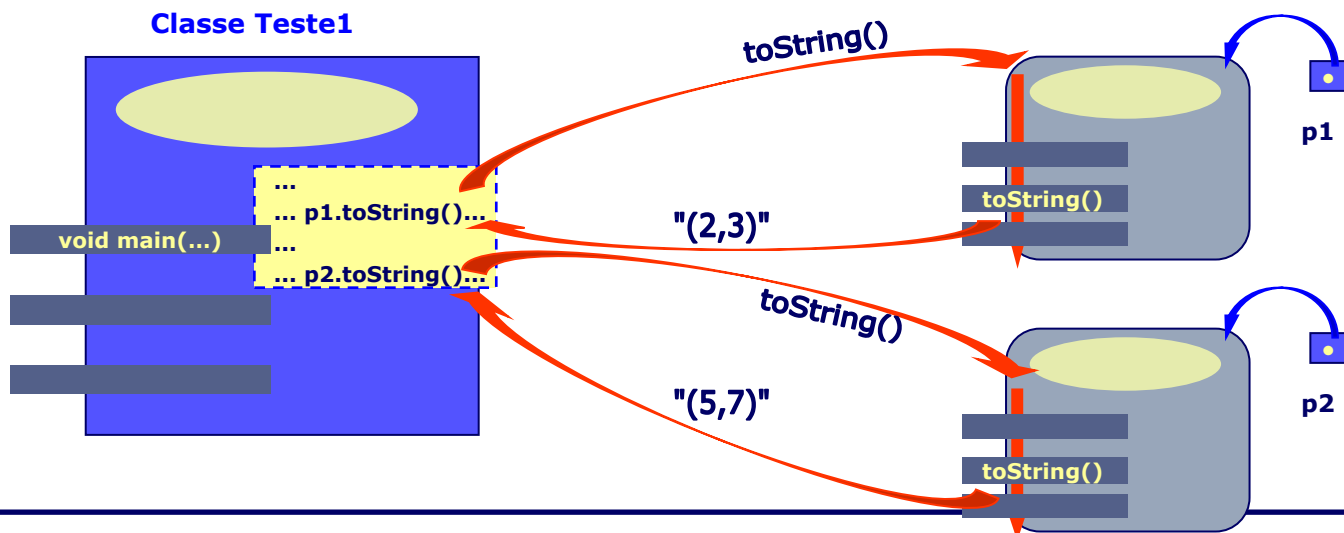
De cada vez que na consola é referido um determinado ponto (objecto da classe Ponto) é necessário recorrer à instrução:

```
... (" + p1.cordX + "," + p1.cordY + ") ...
```

Faz assim sentido que a classe Ponto disponibilize um serviço que **devolva uma representação em String da respectiva instância**, evitando a repetição de código de cada vez que na consola se faz referência a um determinado ponto.

Seja então acrescentada à classe Ponto, o método **toString()** que implementa este serviço, e que fará substituir a instrução anterior por:

```
... p1.toString() ...
```



O resultado "(2,3)" retornado pela instrução `p1.toString()` é igual ao resultado dado pela instrução, `"(" + p1.cordX + "," + p1.cordY + ")"`. Por sua vez o resultado "(5,7)" retornado pela instrução `p2.toString()` é igual ao resultado dado pela instrução, `"(" + p2.cordX + "," + p2.cordY + ")"`.

Implementando o método `toString()` na classe `Ponto`, teremos:

```
public String toString(){  
    return "(" + _____.cordX + "," + _____.cordY + ")";  
}
```

**De que objecto são as variáveis `cordX` e `cordY`?**

São do mesmo objecto a quem foi enviada a mensagem `toString()`.

Como é que dentro do método `toString()` se refere o **próprio** objecto?

```
public String toString(){  
    return "(" + this.cordX + "," + this.cordY + ")";  
}
```

**this** é a "referência especial", do objecto (contexto) em que o método está a ser definido.

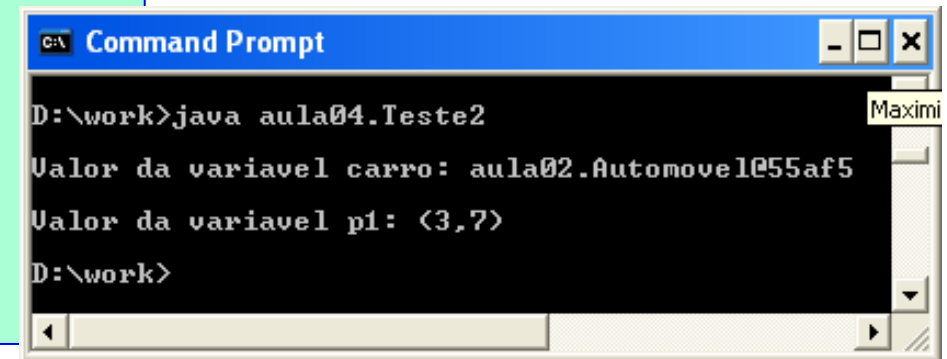
No entanto, nestas situações, a utilização da referência **this** pode ser omitida, uma vez que está implícita.

Quando foi abordado o conceito de tipo referenciado, verificou-se que ao ser invocado o método `pg2.io.IO.writeln()`, recebendo como parâmetro um objecto, era apresentada na consola uma referência para o respectivo objecto.

Seja analisada novamente esta situação, mas com dois objectos de classes diferentes:

```
package aula04;
import aula02.Automovel;
import aula03.Ponto;
import pg2.io.IO;

public class Teste2{
    public static void main (String args[]) {
        Automovel carro = new Automovel(1500, 120, 200);
        Ponto p1 = new Ponto(3,7);
        IO.cout.writeln("\nValor da variavel carro: " + carro);
        IO.cout.writeln("\nValor da variavel p1: " + p1);
    }
}
```



```
Command Prompt
D:\work>java aula04.Teste2
Valor da variavel carro: aula02.Automovel@55af5
Valor da variavel p1: <3,7>
D:\work>
```

Facilmente se percebe que para o objecto da classe `Ponto` foi apresentado o resultado do método `toString()`, enquanto que para o objecto da classe `Automovel`, onde não foi definido este método, aparece uma representação de uma referência para este objecto.

Porque é que foi automaticamente invocado o método `toString()`?

# Object.toString()

Já é conhecido que o operador "+" assume a função de concatenação quando um dos parâmetros é uma String, convertendo também para String os restantes parâmetros .

Neste caso para converter o objecto da classe Ponto para String, foi invocado ao respectivo objecto o método `toString()`, porque todos os objectos têm definido o método `toString()`. **Porquê?**

Todas as classes herdam (derivam) da classe `Object`. Herdar significa "ter", ou seja qualquer objecto "tem" os mesmos atributos e métodos, das instancias de `Object`.

Ou seja, o resultado "aula02.Automovel@55af5" obtido para o objecto da classe `Automovel`, não foi mais que o resultado da execução do método `toString()`, conforme definido em `Object`.

```

o class java.lang.Object
  o class java.lang.Boolean (implements java.io.Serializable)
  o class java.lang.Character (implements java.lang.Character)
  o class java.lang.Character.Subset
    o class java.lang.Character.UnicodeBlock
  o class java.lang.Class (implements java.io.Serializable)
  o class java.lang.ClassLoader
  o class java.lang.Compiler
  o class java.lang.Math
  o class java.lang.Number (implements java.io.Serializable)
    o class java.lang.Byte (implements java.lang.Number)
    o class java.lang.Double (implements java.lang.Number)
    o class java.lang.Float (implements java.lang.Number)
    o class java.lang.Integer (implements java.lang.Number)
    o class java.lang.Long (implements java.lang.Number)
    o class java.lang.Short (implements java.lang.Number)
  o class java.lang.Package
  o class java.security.Permission (implements java.io.Serializable)
    o class java.security.BasicPermission (implements java.security.Permission)
    o class java.lang.RuntimePermission (implements java.security.Permission)
  o class java.lang.Process
  o class java.lang.Runtime
  o class java.lang.SecurityManager
  o class java.lang.StrictMath
  o class java.lang.String (implements java.lang.Character)
  o class java.lang.StringBuffer (implements java.io.Serializable)
  o class java.lang.System
  o class java.lang.Thread (implements java.lang.Runnable)

```

Qualquer classe deriva directa ou indirectamente, da classe **Object**

Descrição de **toString()** na classe `Object`

## toString

```
public String toString()
```

Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

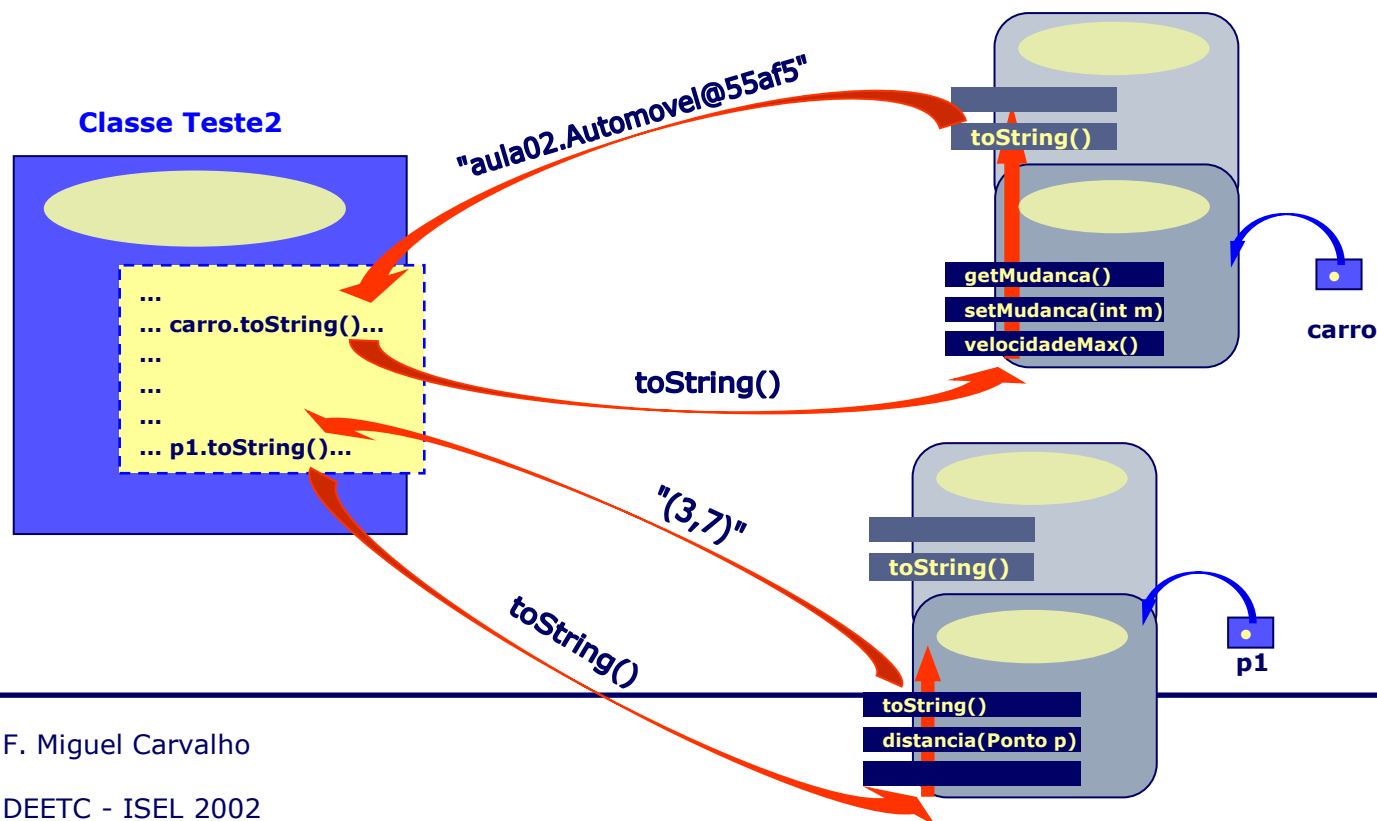
# Override (redefinir)

A descrição do método `toString()` de `Object` acrescenta ainda que:

- "It is recommended that all subclasses override this method."

Foi exactamente isto que foi feito no caso da classe `Ponto`, ou seja, o método `toString()` que foi implementado, não se trata da **definição** de um novo método mas da **redefinição** de um método.

Quando é enviada a mensagem `toString()`, a um objecto de uma classe que não tem este método redefinido então o resultado obtido é o que está implementado em `Object`.



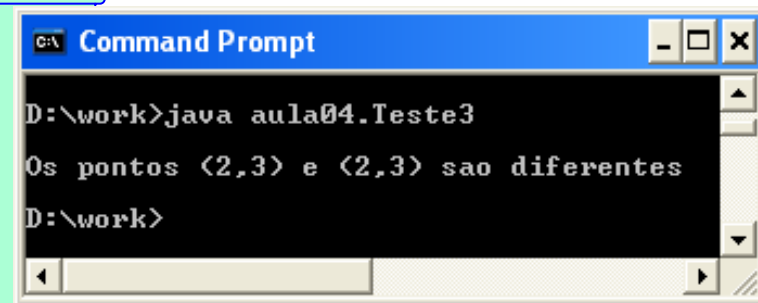
Estas classes podem ser vistas como uma extensão da classe `Object`



# Teste de igualdade entre objectos

Seja analisado o seguinte código:

```
package aula04;
import aula03.Ponto;
import pg2.io.IO;
public class Teste3{
    public static void main(String [] args){
        Ponto p1 = new Ponto(2,3);
        Ponto p2 = new Ponto(2,3);
        String igualdade = p1==p2? "iguais":"diferentes";
        IO.cout.writeln("\nOs pontos " + p1 + " e " + p2 + " sao " + igualdade);
    }
}
```

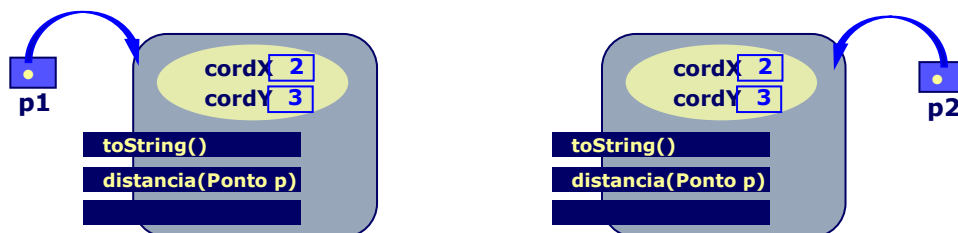


```

C:\> Command Prompt
D:\work>java aula04.Teste3
Os pontos <2,3> e <2,3> sao diferentes
D:\work>

```

Já é conhecido que o resultado da comparação “`p1 == p2`” é *false*, isto porque se trata de uma comparação entre referências de objectos e não dos valores da sua estrutura.



A estrutura dos objectos referenciados por p1 e p2, é igual (tem os mesmos valores), mas tratando-se de instancias distintas as suas referências também são distintas, ou seja, o valor de p1 é diferente de p2.

# Object.equals()

Se quisermos realmente comparar a estrutura interna dos dois objectos, teremos que recorrer à seguinte instrução: `p1.cordX == p2.cordX && p1.cordY == p2.cordY`

Um **teste de igualdade** entre dois objectos é uma necessidade usual quando se programa por objectos. Faz assim sentido que cada objecto tenha a capacidade de “dizer” se um outro objecto qualquer é, ou não, igual a si.

O método `equals()` está definido na classe `Object` e tem um comportamento semelhante ao “==”, ou seja, faz apenas uma comparação das referências dos objectos.

Faz sentido que à semelhança do `toString()`, sendo um método herdado por todas as classes tenha um comportamento genérico, conforme descrito na sua especificação:

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x==y` has the value `true`).

Dados dois objectos quaisquer, por exemplo: `Object o1 = new Object(), o2 = new Object();` sendo a condição, `o1 == o2`, *false*, então o resultado da instrução `o1.equals(o2)`, também é *false*.

# Ponto.equals()

A definição de `equals()`, herdada na classe `Ponto`, não satisfaz o requisitos de um teste de igualdade entre duas instâncias desta classe. Logo, à semelhança do que foi feito para o método `toString()`, o método `equals()` também deverá ser **redefinido** com uma nova implementação.

A nova implementação deve respeitar a mesma relação de equivalência, isto é:

- **Reflectiva:** Para qualquer valor referenciado de `x`, `x.equals(x)`, deve retornar `true`;
- **Simétrica:** Para quaisquer valores referenciados `x` e `y`, a instrução `x.equals(y)` só deve retornar `true`, se e só se, `y.equals(x)` também devolver `true`.
- **Transitiva:** Para quaisquer valores referenciados `x`, `y` e `z`, se as instruções `x.equals(y)` e `y.equals(z)` devolverem `true`, então `x.equals(z)` também deverá retornar `true`.

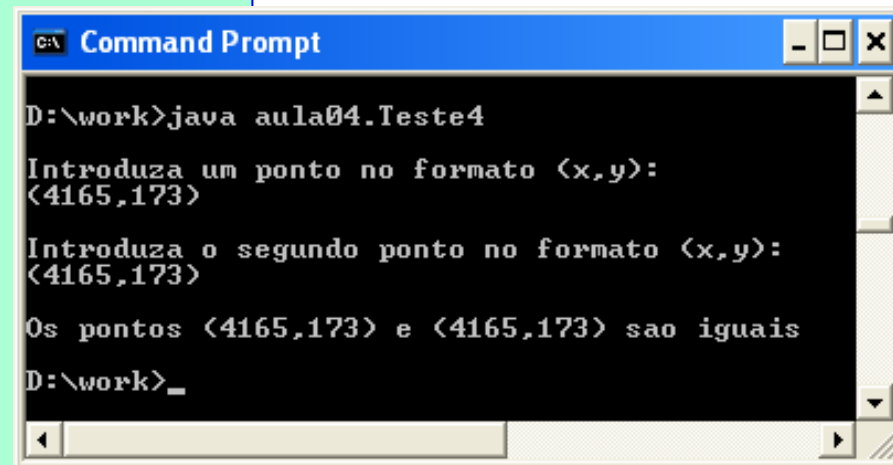
A redefinição do método `equals()` na classe `Ponto`, fica da seguinte forma:

<pre>public boolean equals(Object o){     if( o == null)         return false;     if(!(o instanceof Ponto))         return false;     Ponto p = (Ponto) o;     if( this.cordX == p.cordX &amp;&amp; this.cordY == p.cordY)         return true;     return false; }</pre>	<div style="border: 1px solid blue; padding: 5px; margin-bottom: 10px; width: fit-content;">Assinatura igual à existente na classe <code>Object</code></div> <div style="border: 1px solid blue; padding: 5px; margin-bottom: 10px; width: fit-content;">Se o parâmetro passado for <code>null</code>, devolve <i>false</i></div> <div style="border: 1px solid blue; padding: 5px; width: fit-content;">Se o parâmetro passado não for do mesmo tipo, os objectos são diferentes e é retornado <i>false</i></div>
--	--

Seja agora considerado o seguinte teste:

```
package aula04;
import aula03.Ponto;
import pg2.io.IO;
public class Teste4{
    public static void main(String [] args){
        Ponto p1,p2;
        IO.cout.writeln("\nIntroduza um ponto no formato (x,y):");
        String s = IO.cin.readLine();
        int i = s.indexOf(',');
        int x = Integer.parseInt(s.substring(1,i));
        int y = Integer.parseInt(s.substring(i+1,s.length()-1));
        p1 = new Ponto(x,y);

        IO.cout.writeln("\nIntroduza o segundo ponto no formato (x,y):");
        s = IO.cin.readLine();
        i = s.indexOf(',');
        x = Integer.parseInt(s.substring(1,i));
        y = Integer.parseInt(s.substring(i+1,s.length()-1));
        p2 = new Ponto(x,y);
        String igualdade = p1.equals(p2)? "iguais":"diferentes";
        IO.cout.writeln("\nOs pontos " + p1 + " e " + p2 + " sao " + igualdade);
    }
}
```



Rapidamente se percebe que existe uma duplicação de instruções quando se pretende instanciar um Ponto a partir da String introduzida pelo utilizador.

## ... valueOf()

Passando as respectivas instruções para dentro de uma função auxiliar, a classe Teste4 passa a ter a seguinte forma:

```
package aula04;
import aula03.Ponto;
import pg2.io.IO;
public class Teste4{
    private static Ponto valueOf(String s){
        int i = s.indexOf(',');
        int x = Integer.parseInt(s.substring(1,i));
        int y = Integer.parseInt(s.substring(i+1,s.length()-1));
        return new Ponto(x,y);
    }
    public static void main(String [] args){
        Ponto p1,p2;
        IO.cout.writeln("\nIntroduza um ponto no formato (x,y):");
        String s = IO.cin.readLine();
        p1 = valueOf(s);

        IO.cout.writeln("\nIntroduza o segundo ponto no formato (x,y):");
        s = IO.cin.readLine();
        p2 = valueOf(s);
        String igualdade = p1.equals(p2)? "iguais":"diferentes";
        IO.cout.writeln("\nOs pontos " + p1 + " e " + p2 + " sao " + igualdade);
    }
}
```

O método **static Ponto valueOf** tem como função instanciar um novo Objecto da classe **Ponto**, com base na String passada como parâmetro.

Sendo esta uma função que serve para instanciar especificamente objectos da classe **Ponto**, faz então sentido que seja disponibilizado pela própria classe **Ponto**.

## ... valueOf()



```
package aula03;

public class Ponto {
    private static int numPontos;
    public int cordX, cordY;
    public Ponto(int x, int y) {
        cordX = x; cordY = y;
        numPontos++;
    }
    public static int getNumPontos(){return numPontos;}
    public double distancia(Ponto p){
        double dx = cordX - p.cordX;
        double dy = cordY - p.cordY;
        return Math.sqrt( dx*dx + dy*dy );
    }
    public String toString(){
        return "(" + cordX + "," + cordY + ")";
    }
    public boolean equals(Object o){
        if( o == null) return false;
        if(!(o instanceof Ponto)) return false;
        Ponto p = (Ponto) o;
        if( this.cordX == p.cordX && this.cordY == p.cordY)
            return true;
        return false;
    }
    private static Ponto valueOf(String s){
        int i = s.indexOf(',');
        int x = Integer.parseInt(s.substring(1,i));
        int y = Integer.parseInt(s.substring(i+1,s.length()-1));
        return new Ponto(x,y);
    }
}
```

```
package aula04;
import aula03.Ponto;
import pg2.io.IO;
public class Teste4{
    public static void main(String [] args){
        Ponto p1,p2;
        IO.cout.writeln("\nIntroduza um ponto no formato (x,y):");
        String s = IO.cin.readLine();
        p1 = Ponto.valueOf(s);

        IO.cout.writeln("\nIntroduza o segundo ponto no formato (x,y):");
        s = IO.cin.readLine();
        p2 = Ponto.valueOf(s);
        String igualdade = p1.equals(p2)? "iguais":"diferentes";
        IO.cout.writeln("\nOs pontos " + p1 + " e " + p2 +
            " sao " + igualdade);
    }
}
```

A definição final da classe **Ponto** fica então de acordo com o código apresentado ao lado.

Por sua vez, na classe **Teste4** passa ser invocada a respectiva função **valueOf** da classe **Ponto**.

# Especificação da Classe

Um dos pontos importantes na implementação de uma classe é a **especificação** fornecida sobre a sua API (*Application Programming Interface*), conjunto de métodos e atributos disponibilizados.

Para qualquer classe incluída na **JRE** é possível consultar a sua respectiva descrição nos documentos de especificação da API.

Estes documentos são produzidos a partir dos **comentários** que estão junto do código fonte e através de um utilitário do Java SDK, que é o **javadoc**.