

PG II

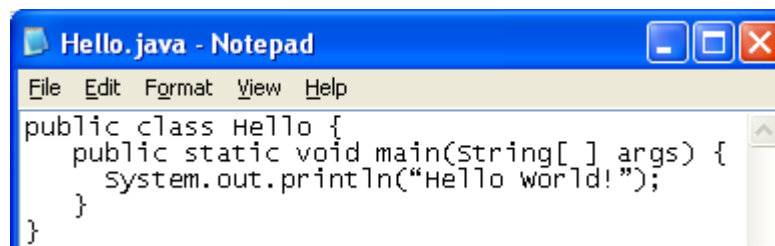
Programação Orientada por Objectos em Java

Introdução:

- Objectivo - OOP
- Programação em Java
- Máquina virtual Java - JVM
- Classes e Funções
- Packages
- Classe pg2.io.IO
- Exemplo: Máximo de dois números
- Troca de mensagens
- Exemplo: Máximo de três números
- Classe Math
- Os Packages da API 1.3

- O principal tema de estudo de PG2 é a **programação usando orientação por objectos**.
- **Algumas das vantagens de se programar usando orientação por objectos são:**
 - Facilidade de reutilização;
 - Abstracção dos detalhes de implementação;
 - Organização modular.
- Como forma de aprendizagem do paradigma **OOP** (**Object Oriented Programming**), será usada a linguagem de programação **Java**;
- **Porquê Java?**
 - Porque cumpre o requisito de ser uma **OOPL** (**Object Oriented Programming Language**);
 - Porque exige um menor esforço de aprendizagem, que por exemplo C++;
 - Porque sintacticamente é muito semelhante ao JavaScript, reduzindo o esforço de aprendizagem de uma nova sintaxe.
- Atingido este objectivo, os alunos poderão aplicar os conceitos de **OOP** noutras linguagens **OOPL**, que cumpram os requisitos deste paradigma, como C++, C# e Smaltalk.

- O **código fonte** de um programa em **Java**, é armazenado num ficheiro de texto, normalmente com extensão **.java**. Exemplo:



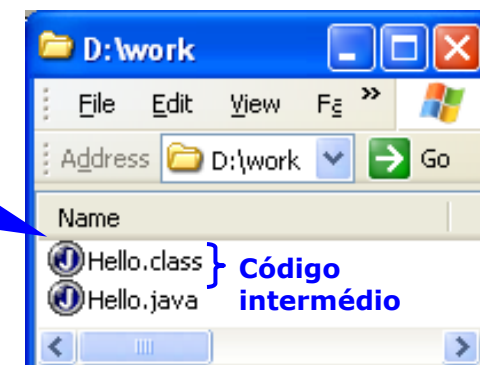
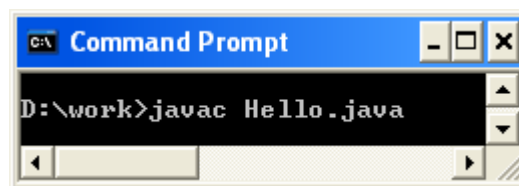
```
File Edit Format View Help
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

- Para que possa ser executado, o **código fonte** terá que ser previamente compilado para **código intermédio**.

Hello.java

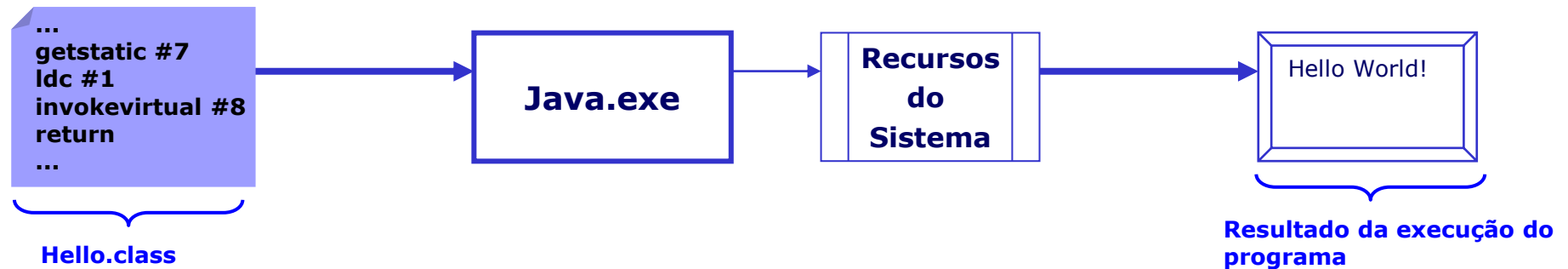


Código fonte

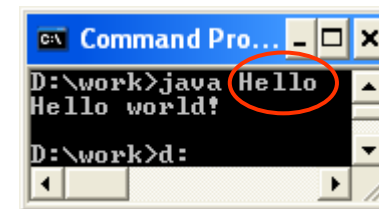
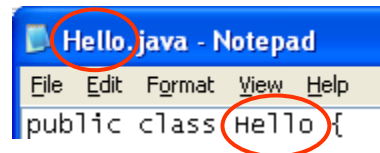


A compilação do código fonte, pode ser feita por linha de comando através do utilitário **javac**, incluído no **Java SDK (Software Development Kit - <http://java.sun.com/>)**

- Os programas Java são executados por uma **JVM** (**Java Virtual Machine**), que sabe interpretar os byte-codes do **código intermédio Java**:



- De notar, que o **nome do ficheiro** que contem o código fonte deve ter exactamente o **nome da classe**, incluindo maiúsculas/minúsculas:



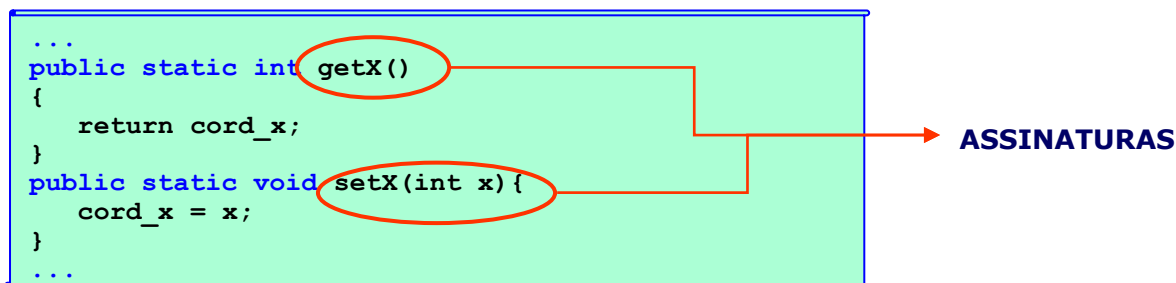
- Podemos executar programas Java em qualquer ambiente que tenha integrada uma **JVM**, como é o caso dos **Browsers** (NetScape, Explorer, HotJava) e da maioria dos Sistemas Operativos (Linux, JavaOS, Windows, etc). As **JVM** para os vários **SO**, estão disponíveis em <http://java.sun.com/>, com a designação de **JRE – Java Runtime Environment**.

Classes e Funções

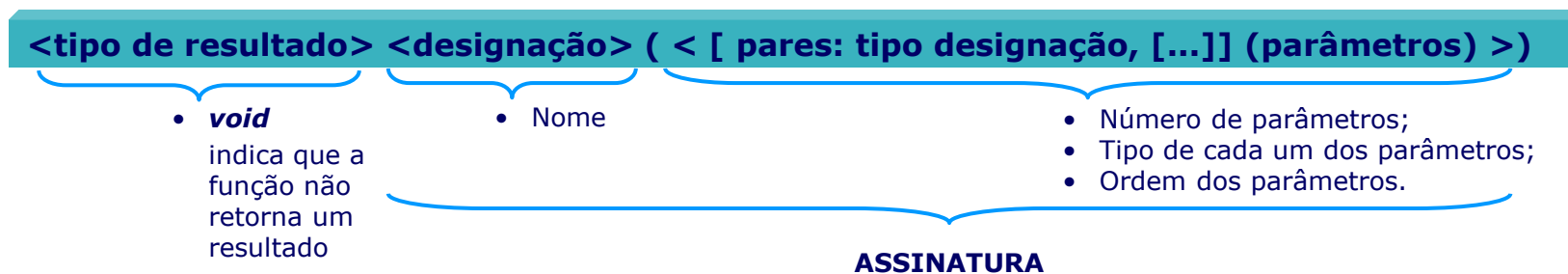
Classe é a única entidade abordada até aqui e é nela que residem os algoritmos implementados. Como forma de organização dos algoritmos serão usadas **Funções** dentro das **Classes**.

A definição de uma função é composta por:

- Cabeçaho ("header");
- Corpo ("body").



Em Java, a estrutura mais básica de **declaração do cabeçalho de uma função** é feita da seguinte forma:



A **assinatura** (*signature*) de uma função é formada pelo seu nome e parâmetros.

De uma maneira geral, os primeiros exemplos de programas Java que irão ser abordados, estarão organizados em funções:

MyFirstClass.java

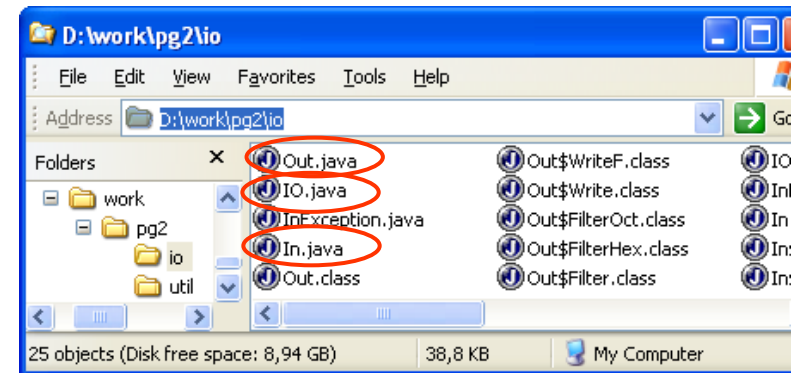
```
public class MyFirstClass{  
  
    public static void myFunc1(){...}  
  
    public static void myFunc2(){...}  
  
    public static void myFunc3(){...}  
  
    public static void main(String [] args){...}
```

Ainda **sem se explicar porquê** existirão **dois** pressupostos que serão seguidos, na implementação das classes Java:

1. Todas as funções serão declaradas como "**static**";
2. Existirá uma função designada "**main**" que recebe como parâmetro "**String[] args**".

Esta função é o "**entry point**" para a respectiva classe. Ou seja, é por esta função que a classe começa a ser executada.

Uma das bases para o desenvolvimento em Java, consiste na reutilização de outras classes já existentes e que nos disponibilizam serviços já implementados. Maioritariamente estas classes estão organizadas sobre a forma de **packages**, que servem para agrupar um conjunto de classes que se enquadram no mesmo âmbito. Um dos **packages** a usar no desenvolvimento dos primeiros programas de PG2 é o **package pg2.io**, que disponibiliza um conjunto de classes (**In**, **Out** e **IO**) para **leitura e escrita da consola**. Através da reutilização deste **package**, poderemos implementar programas que possam interagir com o utilizador através da consola, sem que seja necessário conhecer pormenores de implementação mais complexos, relacionados com as **streams**, cuja matéria só será abordada no último capítulo de PG2.



Classe pg2.io.IO

A classe que iremos reutilizar para fazer escrita e leitura da consola designa-se de **IO** e disponibiliza dois atributos/variáveis estáticas (*static*), **cin** e **cout**, que referenciam dois objectos com funções de leitura e escrita respectivamente. Para acedermos aos objectos referenciados por **cin** e **cout**, basta prefixar as respectivas variáveis com "**pg2.io.IO....**".

As funções disponibilizadas em pg2.io.IO.**cin** e pg2.io.IO.**cout**, são respectivamente :

char readChar()
int getChar()
byte readByte()
short readShort()
int readInt()
long readLong()
float readFloat()
double readDouble()
String readLine()
String readWord()

void write(char c)	void writeln(char c)
void write(byte b)	void writeln(byte b)
void write(short s)	void writeln(short s)
void write(int i)	void writeln(int i)
void write(long l)	void writeln(long l)
void write(float f)	void writeln(float f)
void write(double d)	void writeln(double d)
void write(String s)	void writeln(String s)
void write(Object o)	void writeln(Object o)

Cada uma das funções anteriores está destinada à leitura ou escrita de um determinado **tipo** de valor. Isto é, se pretendermos ler da consola um valor **inteiro** poderemos usar a função **readInt()**, enquanto que para ler um valor **real**, teremos que usar a função **readDouble()**. O mesmo se passa na operação de escrita, embora de forma mais despercebida porque o **nome da função** mantém-se independentemente do **tipo de valor** (**write** ou **writeln**), mudando apenas o tipo de parâmetros recebidos. **Em Java todas as variáveis têm um tipo definido explicitamente.** Mais à frente será visto detalhadamente cada um dos tipos Java.

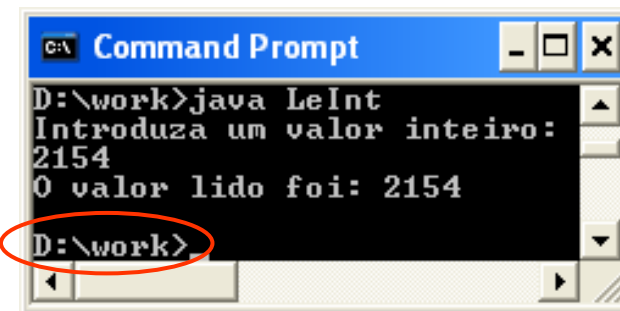
... Classe pg2.io.IO

Um exemplo simples de utilização da classe pg2.io.IO:

```

1 public class LeInt {
2     public static void main (String args[]){
3
4         int n; //variável local à função main,
5               //onde será guardado o valor lido
6         pg2.io.IO.cout.writeln("Introduza um valor inteiro:");
7         n = pg2.io.IO.cin.readInt();
8         pg2.io.IO.cout.writeln("O valor lido foi: " + n);
9     }
10 }

```

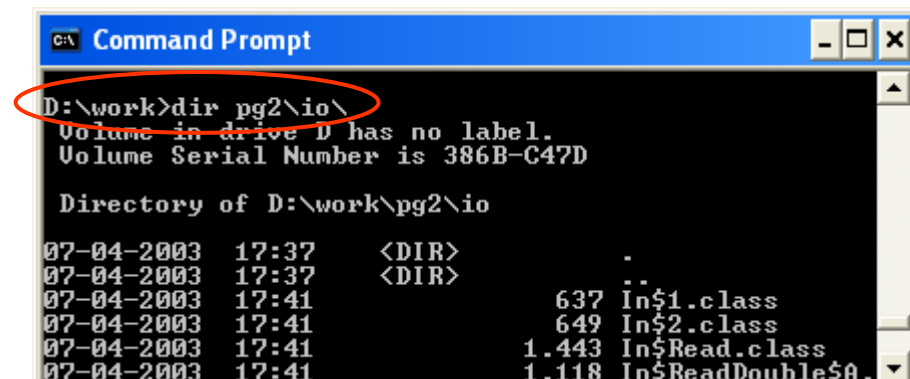


```

C:\ Command Prompt
D:\work>java LeInt
Introduza um valor inteiro:
2154
O valor lido foi: 2154
D:\work>

```

Neste caso o **package pg2.io** deverá estar localizado na directoria de trabalho, que é a directoria de execução da classe **LeInt**, ou seja **d:\work**.



```

C:\ Command Prompt
D:\work>dir pg2\io\
Volume in drive D has no label.
Volume Serial Number is 386B-C47D

Directory of D:\work\pg2\io

07-04-2003  17:37    <DIR>          .
07-04-2003  17:37    <DIR>          ..
07-04-2003  17:41                637 In$1.class
07-04-2003  17:41                649 In$2.class
07-04-2003  17:41            1.443 In$Read.class
07-04-2003  17:41            1.118 In$ReadDouble$A

```

Em alternativa à sintaxe **pg2.io.IO.cin....** poderá ser usado simplesmente **IO.cin....**, desde que antes da definição da classe **LeInt**, seja indicado a importação da respectiva classe **IO** através da instrução:

```
import pg2.io.IO;
```


Máximo de dois números inteiros

No próximo exemplo, a classe **NumMax** vai ser definida dentro do **package aula01**.

```

NumMax
package aula01;
import pg2.io.IO;

public class NumMax {

    public static int max (int a, int b) {
        if (a > b)
            return a;
        else
            return b;
    }

    public static void main (String args[]){

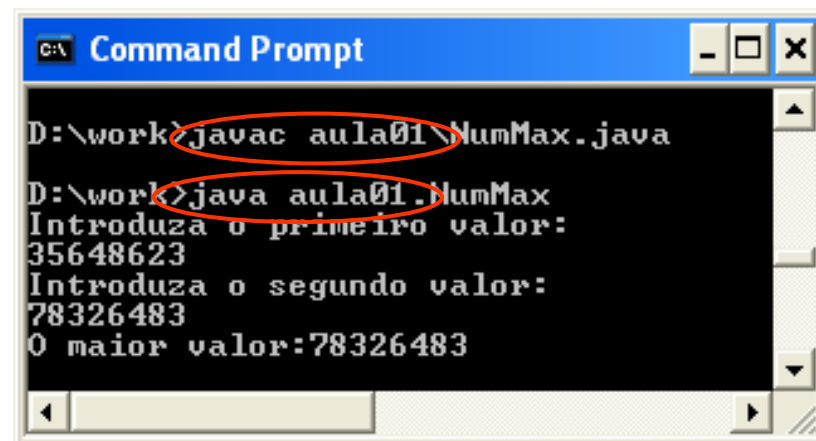
        int a, b, resultado;

        IO.cout.writeln("Introduza o primeiro valor:");
        a = IO.cin.readInt();
        IO.cout.writeln("Introduza o segundo valor:");
        b = IO.cin.readInt();
        resultado = NumMax.max( a, b);
        IO.cout.writeln("O maior valor: " + resultado);
    }
}

```

A classe **aula01.NumMax** disponibiliza as seguintes funções:

- **Função max(...)** – Retorna o máximo entre dois números inteiros passados como parâmetro à função.
- **Função main(...)** – Entry point, para a classe NumMax. Solicita ao utilizador a introdução de dois números inteiros e apresenta na consola o maior deles.



```

C:\> Command Prompt

D:\work>javac aula01\NumMax.java

D:\work>java aula01.NumMax
Introduza o primeiro valor:
35648623
Introduza o segundo valor:
78326483
O maior valor:78326483

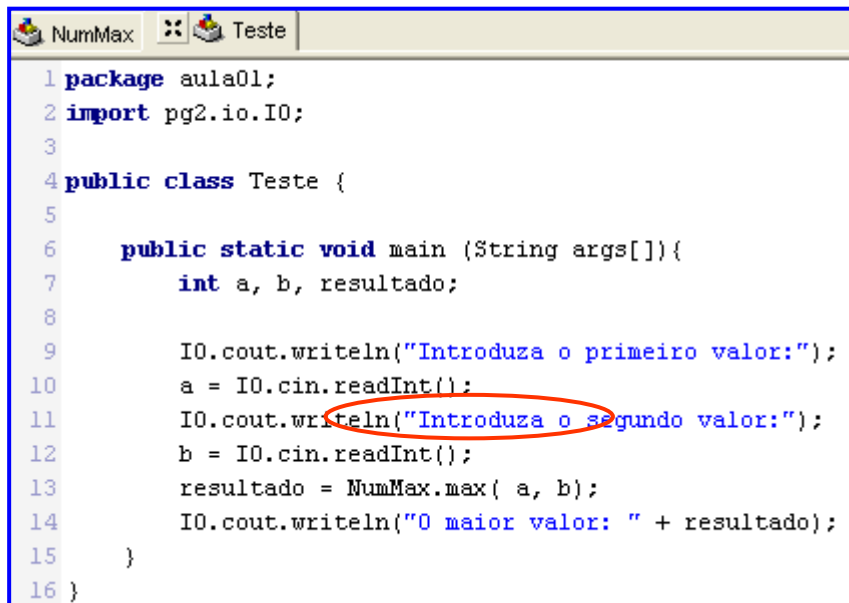
```

De salientar que sendo a classe **NumMax** definida dentro do package **aula01**, o caminho "**aula01**" e "**aula01.**" teve que ser indicado tanto em **compilação** como em **execução**, respectivamente.

... Máximo de dois números inteiros

O **pedido** e respectiva **execução** da função **max** da classe **NumMax** pode ser interpretada por um mecanismo de troca de mensagens.

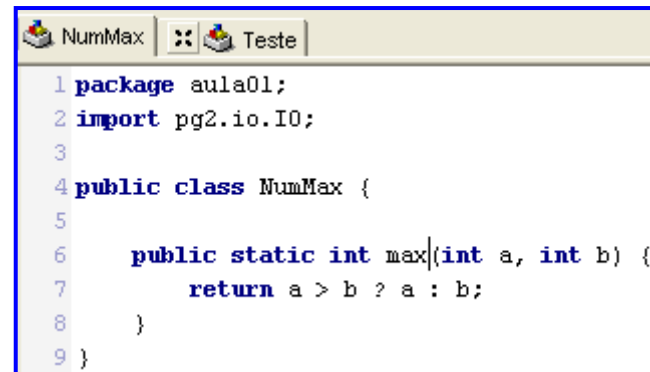
Para melhor esquematizarmos este processo vamos transferir o código de teste da função **main(...)**, para uma classe distinta designada **Teste**.



```

1 package aula01;
2 import pg2.io.IO;
3
4 public class Teste {
5
6     public static void main (String args[]){
7         int a, b, resultado;
8
9         IO.cout.writeln("Introduza o primeiro valor:");
10        a = IO.cin.readInt();
11        IO.cout.writeln("Introduza o segundo valor:");
12        b = IO.cin.readInt();
13        resultado = NumMax.max( a, b);
14        IO.cout.writeln("O maior valor: " + resultado);
15    }
16 }

```



```

1 package aula01;
2 import pg2.io.IO;
3
4 public class NumMax {
5
6     public static int max(int a, int b) {
7         return a > b ? a : b;
8     }
9 }

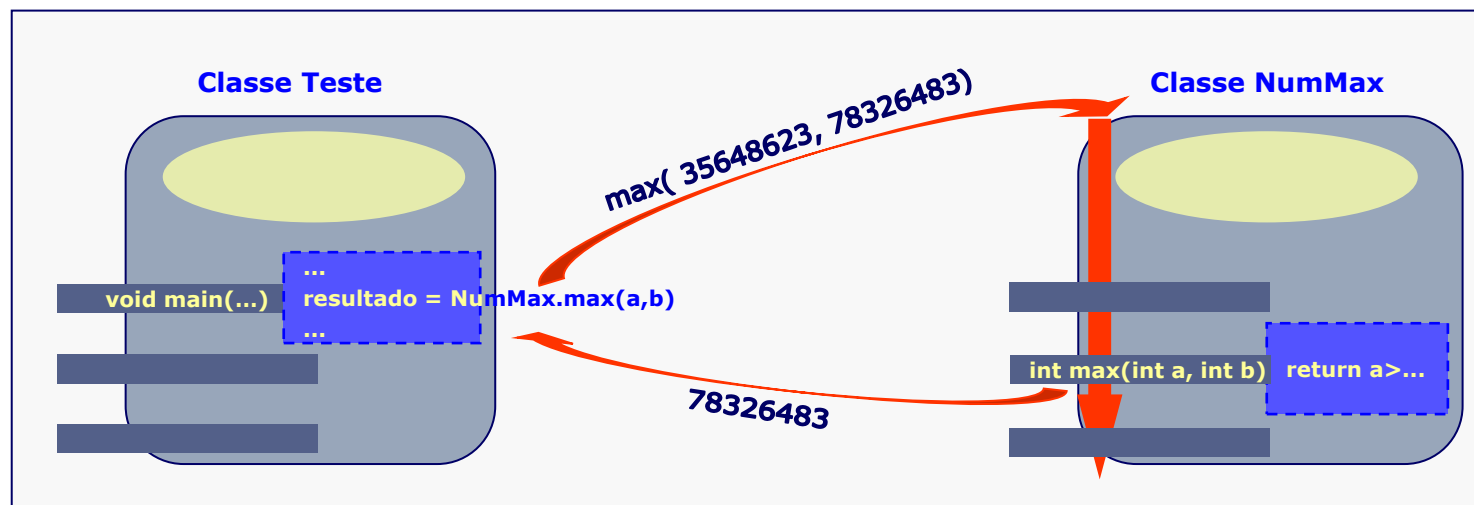
```

Dentro da classe **Teste** é possível fazer um pedido de execução da função **max**, que pertence a uma outra classe (**NumMax**), porque:

1. A função **max** está definida como **public**. Se estivesse definida como **private** só haveria acesso à função **max**, apenas dentro da classe onde foi definida, ou seja, **NumMax**.
2. As classes pertencem ao mesmo package, **aula01**. Caso contrário, seria necessário importar a respectiva classe, através da instrução: **import aula01.NumMax;**

... Máximo de dois números inteiros

O mecanismo de troca de mensagens entre classes pode ser representado da seguinte forma:



```

Command Prompt
D:\work>javac aula01\NumMax.java
D:\work>javac aula01\Teste.java
D:\work>java aula01.Teste
Introduza o primeiro valor:
35648623
Introduza o segundo valor:
78326483
O maior valor: 78326483

```

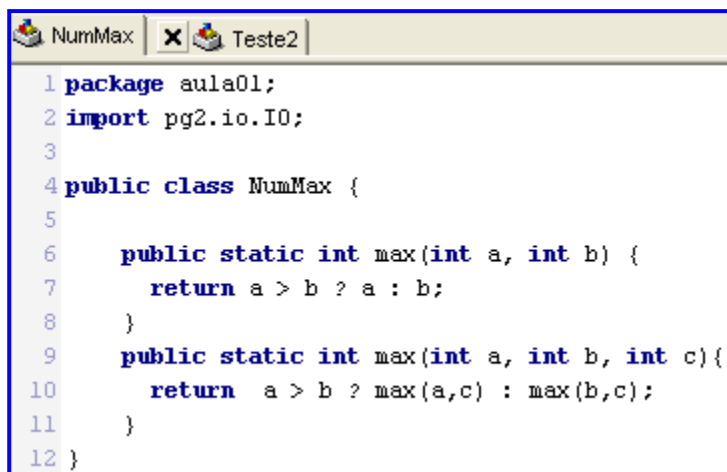
1. Envio da mensagem "max(35648623, 78326483)" à classe **NumMax**;
2. A classe **NumMax** procura a função que tenha na **assinatura** o **nome** max e que receba como **parâmetros** dois valores inteiros;
3. A função **max(int a, int b)** é executada assumindo a variável "a" o valor "35648623" e a variável "b" o valor 78326483;
4. Como retorno, a função **max** envia uma mensagem com o maior dos valores recebidos como parâmetro.

Máximo de três números inteiros

No próximo exemplo vai ser adicionado mais um “**serviço**” (**função**) à classe **NumMax**, para cálculo do maior valor entre 3 números inteiros, recebidos como parâmetro:

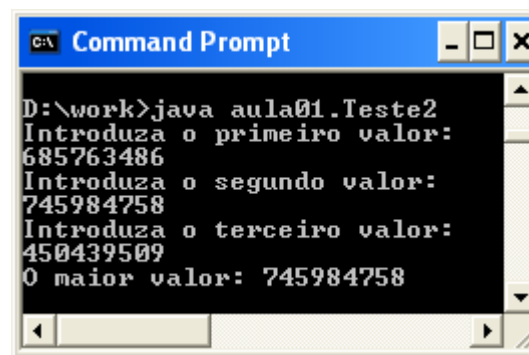
- Por coerência, o nome da nova função será **max**, o que não causa qualquer conflito, uma vez que o número de parâmetros recebidos é diferente da função anterior;
- A implementação desta função vai ser feita à custa da função anterior, ou seja, **reutilizando**.

A reutilização de funções já implementadas, vai ser um dos princípios a seguir sempre que possível.



```

1 package aula01;
2 import pg2.io.IO;
3
4 public class NumMax {
5
6     public static int max(int a, int b) {
7         return a > b ? a : b;
8     }
9
10    public static int max(int a, int b, int c){
11        return a > b ? max(a,c) : max(b,c);
12    }
13 }
  
```



```

D:\work>java aula01.Teste2
Introduza o primeiro valor:
685763486
Introduza o segundo valor:
745984758
Introduza o terceiro valor:
450439509
0 maior valor: 745984758
  
```

Notar que a implementação do algoritmo de cálculo do máximo entre três inteiros, seria muito mais complexa, caso não fosse reutilizada a função “**max(int a, int b)**”.

Classe Math



Um dos principais instrumentos de estudo na implementação de programas em Java, são os **documentos de especificação da API** (*Application Programmer's Interface*) **Java**.

Neles encontramos a descrição funcional de um conjunto de **packages** já integrados e disponibilizados pela plataforma Java, cujas **Classes** oferecem diversos **serviços**.

Estas páginas podem ser consultadas nos links:

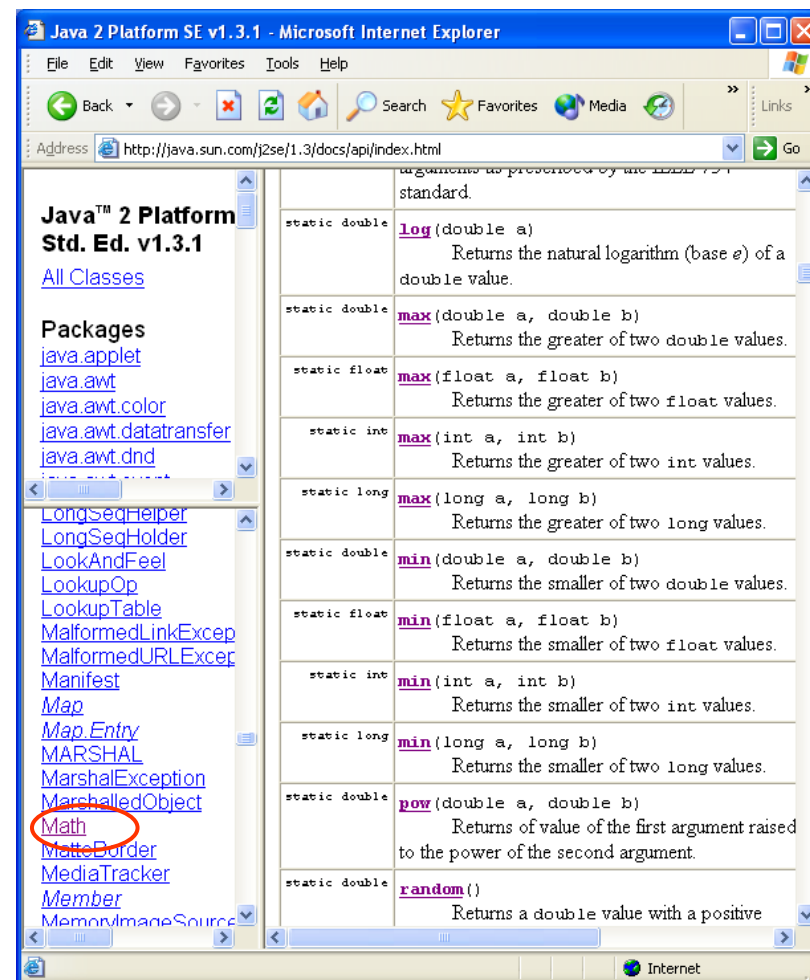
- <http://java.sun.com/j2se/1.3/docs/api/index.html>
- <http://java.sun.com/j2se/1.4.1/docs/api/index.html>

(Existem diversos documentos de acordo com a versão Java.)

Entre as classes disponíveis, existe a classe **Math** que além de uma série de outras funções, já inclui a função **max(...)**, para cálculo do maior de dois números.

Como podemos verificar na figura, existem diversas funções com a mesma designação "**max**", mas que recebem parâmetros diferentes consoante o tipo de números a manipular.

Assim, a função anterior também poderia ser implementada por reutilização desta classe. Nesse caso não seria necessário fazer a respectiva importação, uma vez que a classe pertence ao package **java.lang**, cujas as classes são directamente acessíveis.



Os Packages da API 1.3

Na API 1.3 existem 3 *packages* principais:

- java
- javax
- org.omg

Na cadeira de PG2, serão estudados os packages assinalados na tabela ao lado.

java	(36)	Biblioteca principal
java.applet		Applets
java.awt	+11	Ambiente de janelas
java.beans	+1	Componentes Beans
java.io		input & output
java.lang	+2	Classes principais
java.lang.reflect		Introspecção
java.math		Aritmética
java.net		Comunicação
java.rmi	+4	Remote Method Invocation
java.security	+4	Mecanismos de segurança
java.sql		JDBC
java.text	+1	Internacionalização
java.util	+2	Contentores & Algoritmos
javax	(29)	Extensão à biblioteca (1.2)
javax.swing	+15	Ambiente de janelas (plaf)
org.omg	(11)	Objectos Distribuídos
org.omg.CORBA	+6	Objectos CORBA