# Programação por Objetos & Introdução à Linguagem Java

Programação II

2015/16

©2016 Lígia Ferreira, Salvador Abreu

# Programação por Objetos

- Apresentar o Paradigma da Programação Orientada Objectos com recurso à linguagem JAVA.
- Paradigma- Forma particular de estruturar e organizar conhecimento numa área específica.
- Paradigma de Programação (79 Floyd)- Modelo e abordagem organizacional que permitem definir o que se entende por computação.
- O paradigma da POO, centra-se no conceito de OBJECTO. (Simula-67). Trata-se duma abstracção que representa:
  - ➡ entidade única
  - ➡ estrutura
  - ➡ comportamentos

# Paradigma POO

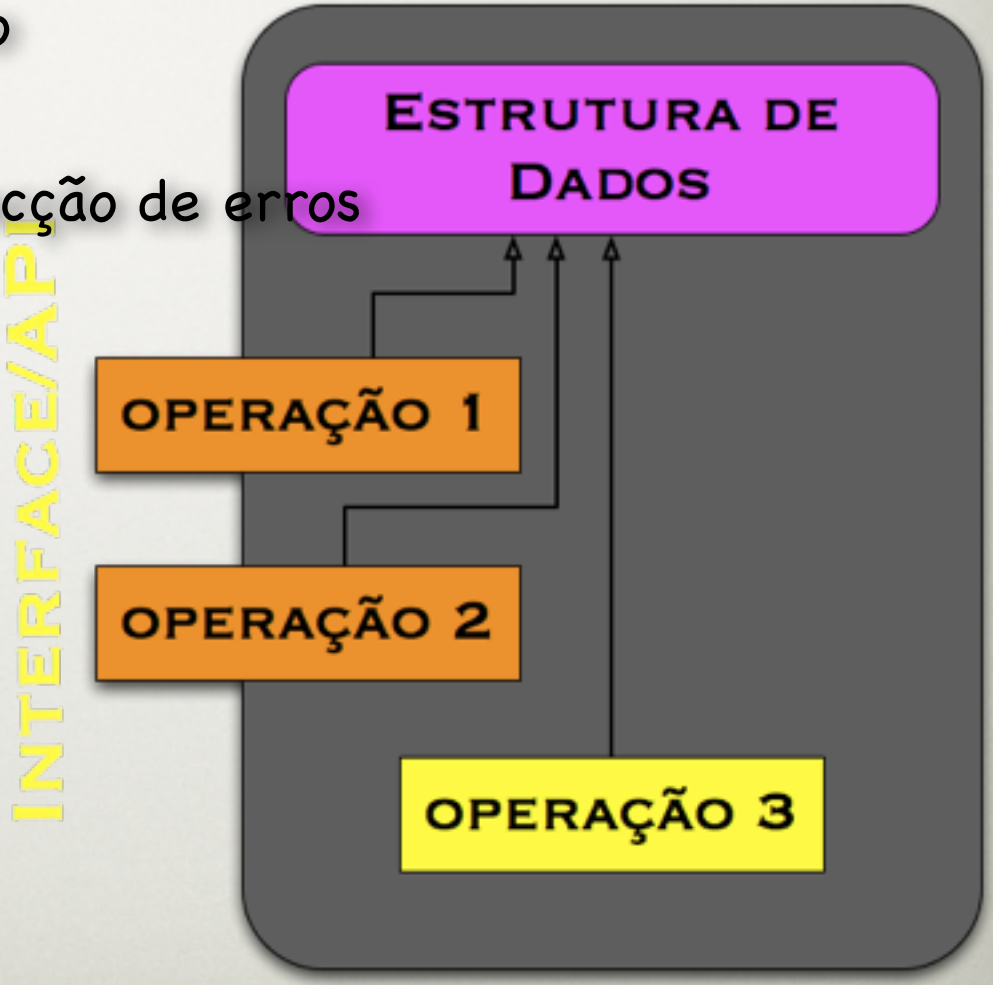
➡ Exemplo:

- Ponto do Plano
  - Abstração
  - Atributos /Características
  - Comportamento /operações passíveis de realizar sobre o objeto

- Ponto2D
  - coordenada em X;
  - coordenada em Y;
  - saber/modificar coordenada em X;
  - saber/modificar coordenada em Y;

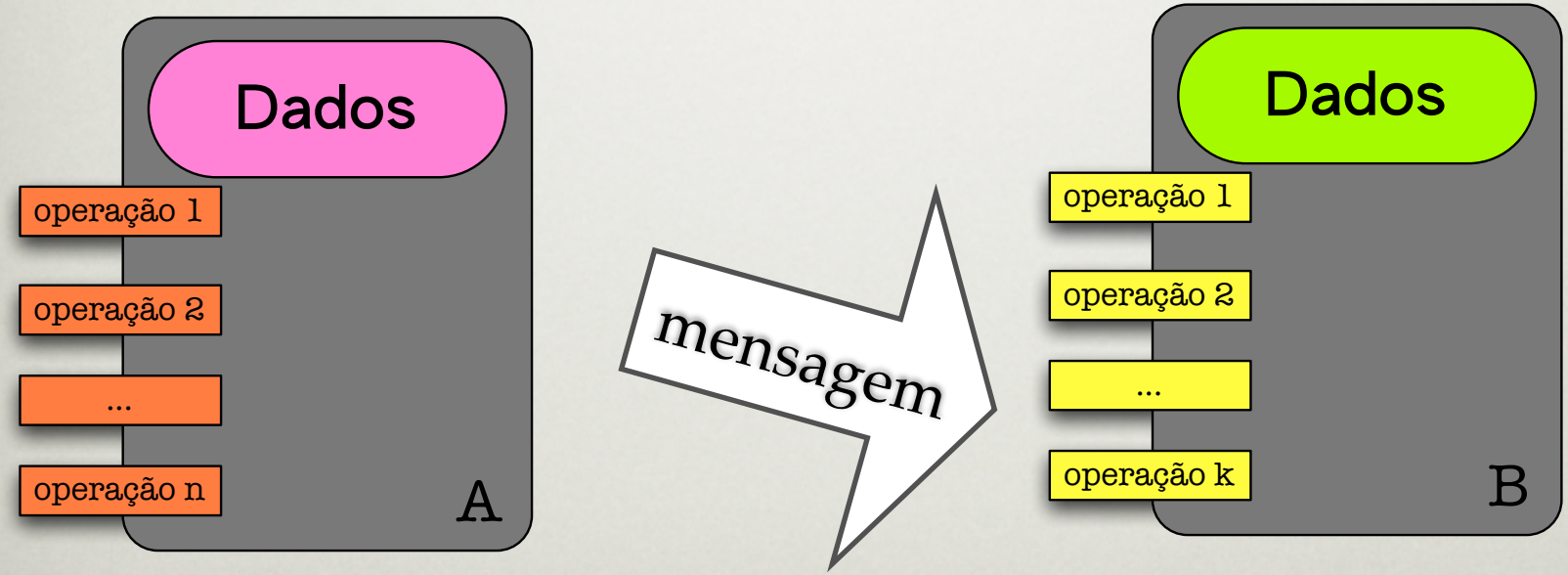
# Objeto: Encapsulamento

- Independente do contexto
- Reutilização de código
- Facilidade correcção/detecção de erros
- Modularidade



# Mensagens

- Se determinado objeto A, quer que o objeto B, execute determinado método tem de lhe enviar a mensagem adequada



# Mensagens

- A determinação da operação que será executada pelo recetor depende do identificador e número de parâmetros da messagem.
- Se existir no recetor uma operação de igual identificador e número de parâmetros será executada.

# Mensagens

- A computação resulta do facto do objecto que recebe a mensagem (receptor) executar a operação que está associada à mensagem.
- Consoante a mensagem poderá acontecer uma alteração no estado do receptor e/ou ser devolvido um resultado ao objecto que enviou a mensagem (emissor)

# Exemplos de Mensagens

- receptor.mensagem();
  - p2.print();
- receptor.mensagem(arg1,arg2,...,argn);
  - p2.setX(3);
- valor=receptor.mensagem();
  - X=p2.getY();
- valor=receptor.mensagem(arg1,arg2,...,argn)

# Classes

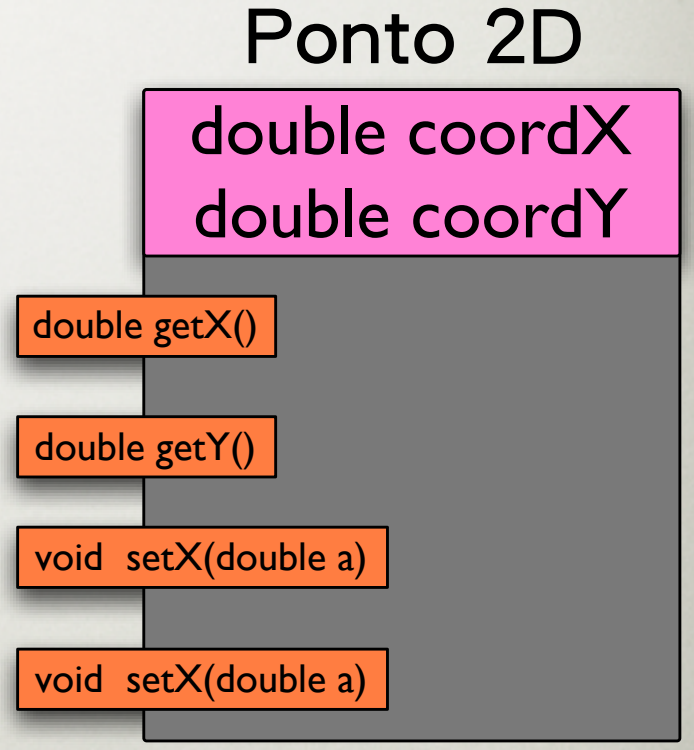
- Como garantir que objectos do mesmo tipo (Pontos2D) ao serem criados tenham a mesma estrutura interna e o mesmo comportamento?
  - ➡ Usar uma técnica "Copy-Paste"
  - ➡ Definir um objecto especial que armazene o padrão do tipo que representa (dados e operações) e que quando se criam objectos desse tipo se reproduza o padrão.

# Classes

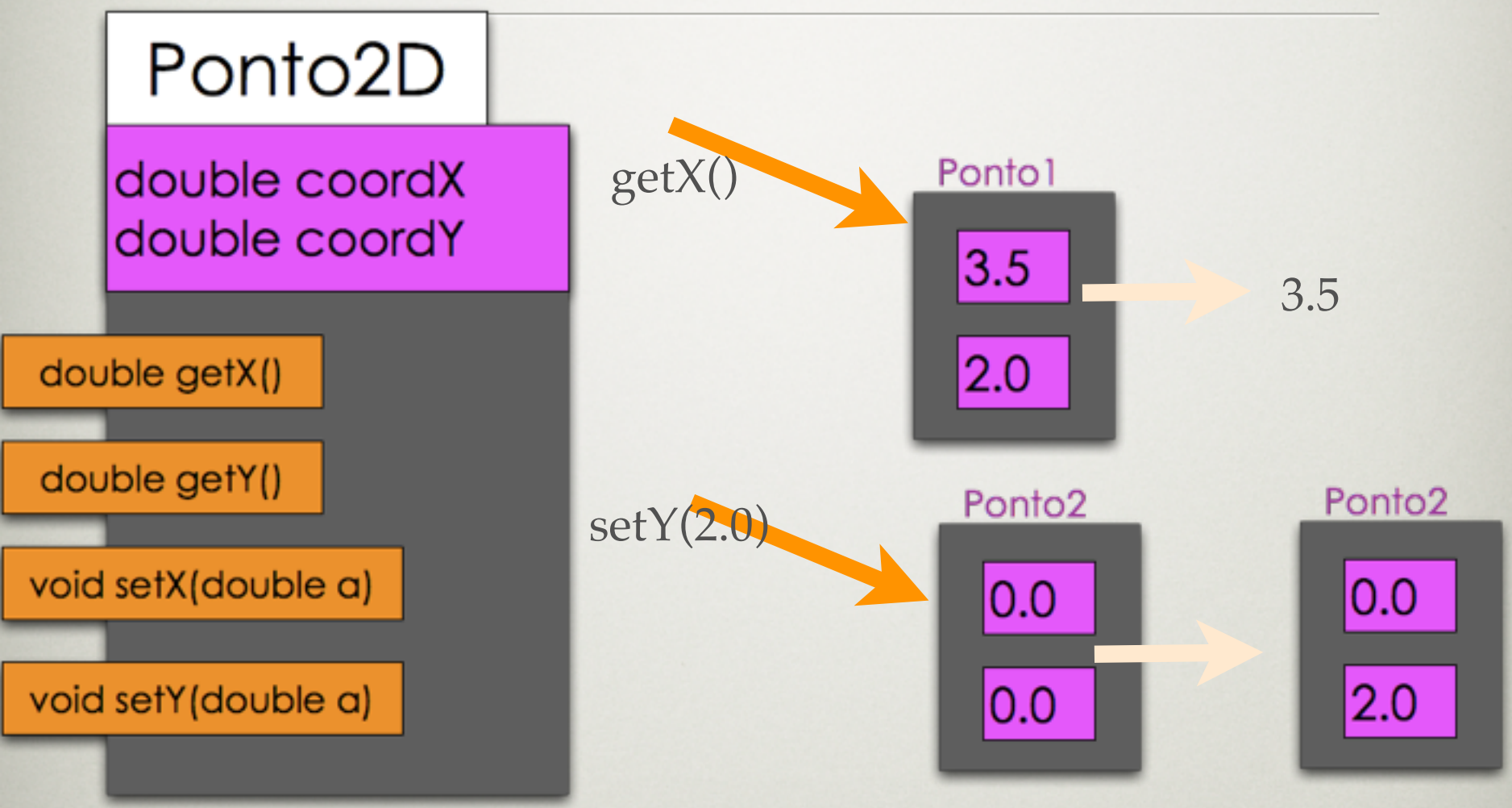
- É o papel das classes.
- Para que servem?
  - ➡ Definir estrutura e comportamento dos seus objectos
  - ➡ Providenciar um mecanismo que permita a criação de novos objectos (instâncias) dessa classe

# Classes

- Ao definirmos esta classe ficamos a saber:
  - Os pontos2D são caracterizados por:
    - coordenada em X
    - coordenada em Y
  - Para Pontos2D podemos:
    - Saber suas coordenadas
    - Alterar as coordenadas



# Classes e Instâncias



# Uma implementação do paradigma

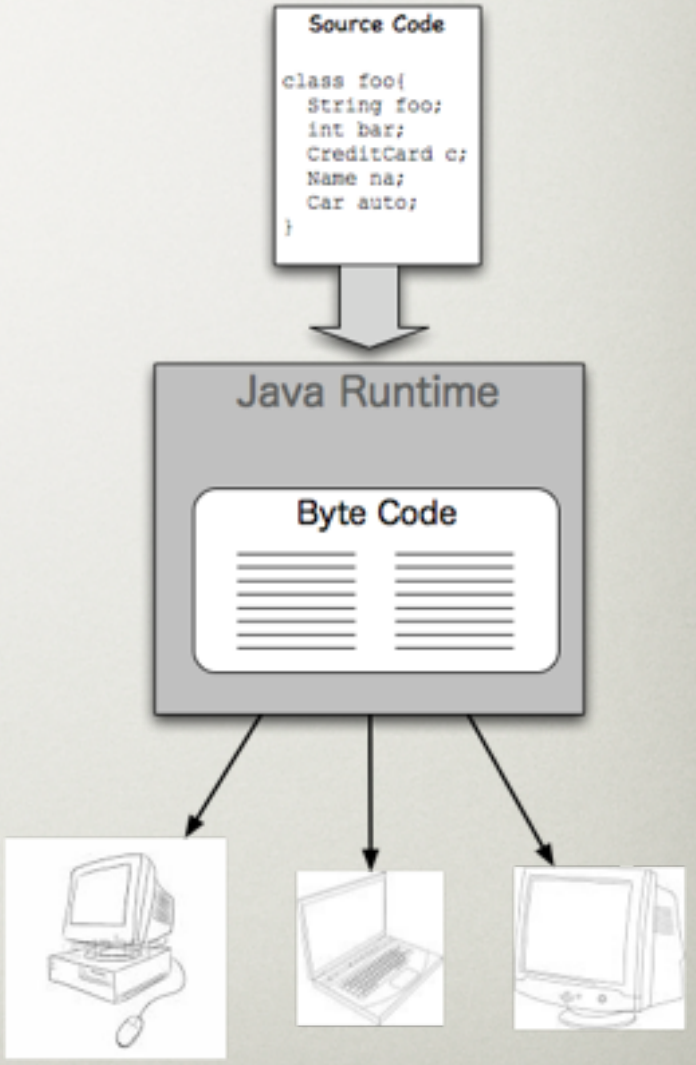
- Como criar objetos ?
- Como especificar classes ?
- Como definir características/dados dos objectos?
- Como enviar mensagens aos objetos criados?
- Como criar operações correspondentes às mensagens?
- Como obter  computações usando este paradigma?

# Linguagem Java

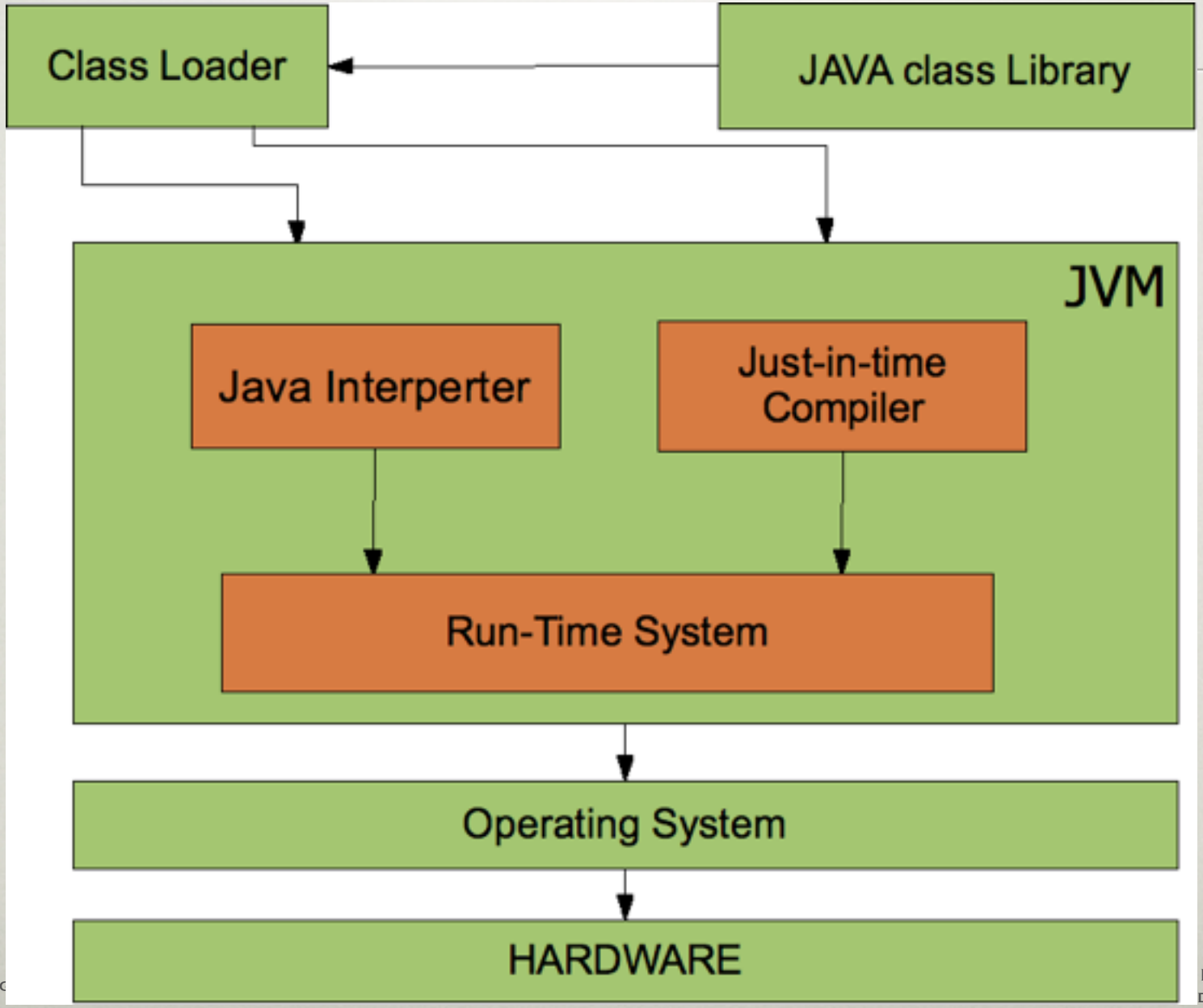
- Simples
  - Relativamente a outras linguagens (C, C++), foram eliminadas construções que comprometem a transparência (apontadores, alocações explícitas de memória,...)
- Orientada a Objectos
  - Foi criada desde o início como linguagem que implementa o paradigma, não foi à "posteriori" adaptada para o incorporar.

# Linguagem Java

- É uma linguagem pré-compilada
- e também interpretada
  - ➡ o byte-code gerado pelo compilador é interpretado pelas JVMs:



# Ambiente de Execução



# Linguagem Java

- Desempenho
  - ➡ É uma linguagem interpretada, logo não comparável a linguagens como Pascal, C, C++ para certas aplicações. Melhores compiladores Just-in-time são uma possibilidade para melhorar o desempenho.
- Multithreaded
  - ➡ Permite a execução simultânea de pequenos processos (light processes);
- Distribuída (RPC's e RMI)
  - ➡ Permite invocação remota de métodos, e um conjunto de classes (classes de Rede) que implementam os mecanismos de acesso remoto;
- Robusta
  - ➡ fortemente tipada
  - ➡ possui mecanismos de tratamento de exceções
- Segura
  - ➡ Possui mecanismos para verificação interna de byte-code ("sand box model")

# Linguagem Java


- Dois níveis de tipos/ duas perspectivas

| Tipo | valores | default | bits | gama |
|---|---|---|---|---|
| boolean | true, false | FALSE | 1 | |
| char | caracteres unicode | \u0000 | 16 | \u0000 a \uFFFF |
| byte | inteiro com sinal | 0 | 8 | -128 a 127 |
| short | inteiro com sinal | 0 | 16 | -32768 a 32767 |
| int | inteiro com sinal | 0 | 32 | -2147483658 a |
| long | inteiro com sinal | 0 | 64 | ≈-1E+20 a 1E+20 |
| float | IEEE 754 FP | 0 | 32 | ≈±3.41E+38 a ±1.4E-44 |
| double | IEEE 754 FP | 0 | 64 | ≈±1.8E+308 a ±5E-324 |

# Linguagem Java

- Literais
  - inteiros: Especificados em octal, decimal ou hexadecimal, para octal usa-se prefixo 0 (zer0) para hexadecimal usa-se o prefixo 0x (zero x)
    - 
```
int x=23;
int i=0123; // octal 123 decimal 83
int a=0x00ff; // hexadecimal ff decimal 255
```

- literais inteiros são sempre do tipo int excepto quando são sucedidos pelo prefixo L
    - 
```
long r=13L;
long r=13; // inteiro 13  e promovido a long
int i=13;
byte b=i;  //erro
byte b= (byte) i;
```

# Linguagem Java

- vírgula flutuante: Podem ser especificados numa notação decimal ou científica. São do tipo double a menos que tenham o sufixo f (de float);
  - double d=7.72;

    double e=5.45e+8;

    float f=7.1f;

    float g=3.00e+5f;

# Linguagem Java

- caracteres: Podem ser expressos usando o caracter entre ' ', pela sequência de escape ASCII ou pelo caracter unicode também entre ' ':

```
char ch1='a';
char ch2='\n';
char ch3='\u00a7' //o caracter unicode 167   §
```

- Coerção (Casting): conversão forçada entre tipos diferentes

```
int x=23;
float f=(float) x;
char ch=(char) x;
int k= (int) ch +100;
```

# Linguagem Java

- Outras instruções básicas:
  - EXPRESSÕES: instruções que produzem um resultado (quando avaliadas) passível de ser usado noutra instrução; expressões matemáticas e chamadas de métodos são exemplos de expressões; blocos de código ( várias instruções englobadas por chavetas) são uma instrução. O valor retornado pode ser um tipo primitivo ou...

# Operadores

| Prioridade | Símbolo | Tipo | Significado |
|---|---|---|---|
| 1 | ! | booleano | negação |
| 1 | (tipo) | qualquer | cast |
| 2 | * / % | aritmético | multiplicação divisão resto |
| 3 | + - | aritmético | soma subtração |
| 5 | > >= < <= | aritmético | comparação |
| 6 | == != | Objectos / tipos | igualdade desigualdade |
| 7 | & | booleano | and |
| 8 | ^ | booleano | xor |
| 9 | \| | booleano | or |
| 10 | && | booleano | and condicional |
| 11 | \|\| | booleano | or condicional |
| 12 | ? : | _ | opertador ternário |
| 13 | = | qualquer | atribuição |
| 13 | *= /= %= | qualquer | atribuição com operação |
| 13 | += -= | qualquer | atribuição com operação |
| 13 | &= \|= | booleano | atribuição com operação |

# Linguagem Java - condicionais

- Condicionais:
  - IF-THEN-ELSE
  - if (exp-booleana) instrução [else instrução];

```
if (a>b)
   c=a;
else
   c=b;
```

```
if (x%2==0)
   i++;
```

```
if (x!=3 && x!=9){
   i+=10;
   j=i;
}
else {
   i+=7;
   j=-i;
}
```

# Linguagem Java - switch

- Condicionais;
  - SWITCH- permite selecionar as instruções a realizar por avaliação duma expressão inteira ou que possa ser promovida a este tipo;

```
switch (expressão){
case valor: instruções ;
[case valor: instruções ;]...
[default: instruções ;]
}
```

```
switch(mes){
   case 2:
        dias=28;
        break;
   case 11:
   case 4:
   case 6:
   case 9:
        dias=30;
        break;
   default:
        dias=31;
}
```

# Linguagem Java - ciclos for

- Repetitivas
  - FOR - Permite repetir uma instrução (bloco!), enquanto a condicional for verdadeira. A iteração descreve a evolução da(s) variáveis da inicialização nas iterações sucessivas.
  - for(inicialização; condicional; iteração) instrução;

âmbito ???

```
for (int a=0, b=10; a<b; a++,b--){
  dias=a*b;
  System.out.println(dias);
}
```

```
for(i=-6;i!=0; i+=2)
   System.out.println(i);
```

```
for(i=-6;i!=0; i+=4)
   System.out.println(i);
```

```
for(i=-6;i>0; i+=4)
   System.out.println(i);
```

# Linguagem Java - ciclo while

- Repetitivas

➡ WHILE – Permite repetir uma instrução (bloco!), enquanto a condicional for verdadeira.

while( condicional) instrução;

```
boolean encontrou=false;
int dig=5;
while (!encontrou && d!=0){
  encontrou= (d%10==dig);
  d /= 10;
}
```

```
boolean encontrou=true;
int dig=5;
while (!encontrou && d!=0){
  encontrou= (d%10==dig);
  d /= 10;
}
```

# Linguagem Java - ciclos

- Repetitivas

➡ DO/WHILE - Permite repetir uma instrução (bloco!), enquanto a condicional for verdadeira. Teste condicional está à saída do ciclo logo...

do {instrução;} while (condicao);

```
int i=2;
int j=10;
int soma=0;
do{
   soma=soma+i;
   j--;
   }
while (j>0);
System.out.println(soma);
```

resultado?

# Linguagem Java - Arrays

- Arrays
- Em java tipos não primitivos(tipos de referência) são os Arrays e os Objectos. A criação dinâmica (criados em tempo de execução) dum array faz-se com new (mas não são objectos) indicando o tipo dos seus elementos e a respectiva dimensão;

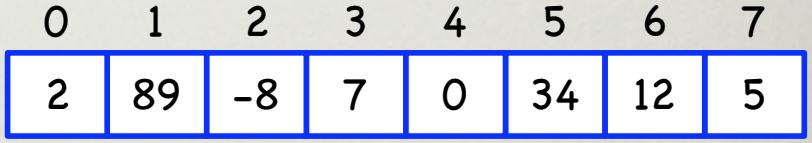
```
int x[];

int[] x;

int[] x={2,89,-8,7,0,34,12,5};

int[] x=new int[8];

int[][] tabela=new int[5][5];
```



# Linguagem Java - Arrays

- Acesso: Faz-se usando identificador [indice], sendo índice um inteiro entre 0 e dimensão array -1
- A dimensão dum array é dada por identificador.length;

```
int s=0;
for (int i=0;i<x.length;i++)
   s=s+x[i];
```

```
int max=x[0];
int i=2;
do {
   if(x[i]>max)
        max=x[i];
   i++;
} while(i<x.length);
System.out.println(max);
```

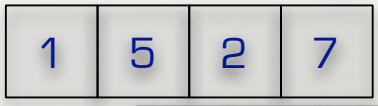
# Linguagem Java

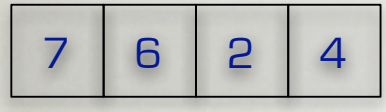
- Qualquer dos exemplos de Java apresentados pode ser testado colocando o código dado, dentro dum programa, mas como não sabemos criar programas vamos usar o painel "interactions" do Dr.Java.

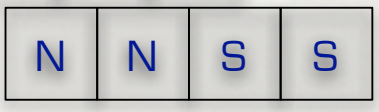
# Exercícios

- Escrever código Java, para:
  - Dado um número, saber se é divisível por 5
  - Dado um array:
    - Contar quantos números impares tem um array de shorts
    - Saber qual o maior elemento do array
    - Somar todos os elementos do array
    - Somar só os positivos
  - Dada uma chave (Array com 4 dígitos sem repetições) e outro número expresso como um array de 4 dígitos, saber se o 2º numero dado tem os mesmos números da chave, mesmo que por outra ordem.

# Roda da Sorte

- Por exemplo, se for dado 

sendo a chave 

deve ser retornado 


Os exercícios são para trabalho de casa e na próxima aula devam ser apresentados aos professores