



# Módulo II

## Introdução à Linguagem Java

### Assuntos

Histórico

Características da Linguagem

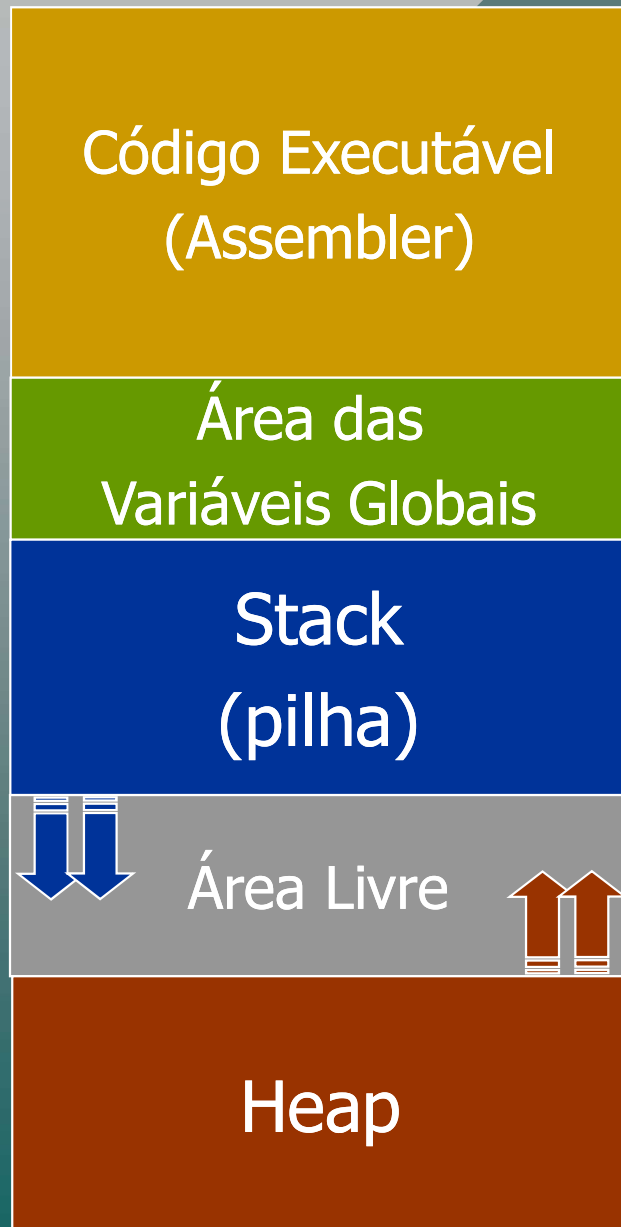
Primeiros Exemplos

# Créditos

## Autor

Prof. Alessandro Cerqueira  
([alessandro.cerqueira@hotmail.com](mailto:alessandro.cerqueira@hotmail.com))

# Uma Visão Geral da Estrutura Geral dos Programas na Memória



- Ao solicitarmos a execução de um programa `.exe`, ele ficará disposto na memória de uma forma semelhante a proposta pela figura ao lado.
- **Stack (pilha)** – Área onde são instaladas os parâmetros e variáveis locais das funções/ procedures/métodos. Quando chamamos uma função, é alocada memória para os seus parâmetros/variáveis locais no topo da pilha. Ao término da função estes são desempilhados.
- **Heap** – Área onde é feita a alocação dinâmica de memória.
- O sentido de crescimento da stack é inverso ao da heap para que se possa aproveitar ao máximo o espaço da área livre.

# Funcionamento da Stack

## Exemplo

```
main()
```

```
{  
    int x = 5;  
    int y = 2;  
    → x = funcPrim(x, y);  
    y = x - 10;  
}
```

```
int funcPrim(int p1, int p2)
```

```
{  
    int x = p1 + p2;  
    int y = p1 * p2;  
    x = funcDois(x, y);  
    return x;  
}
```

```
int funcDois(int a, int b)
```

```
{  
    return b * a / 2;  
}
```

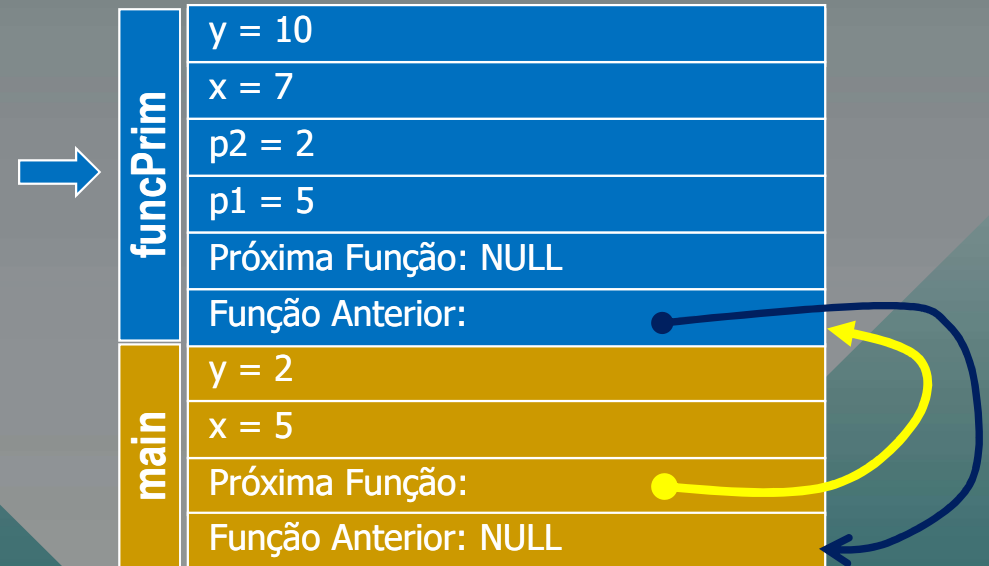


main	y = 2
	x = 5
	Próxima Função: NULL
	Função Anterior: NULL

# Funcionamento da Stack

## Exemplo

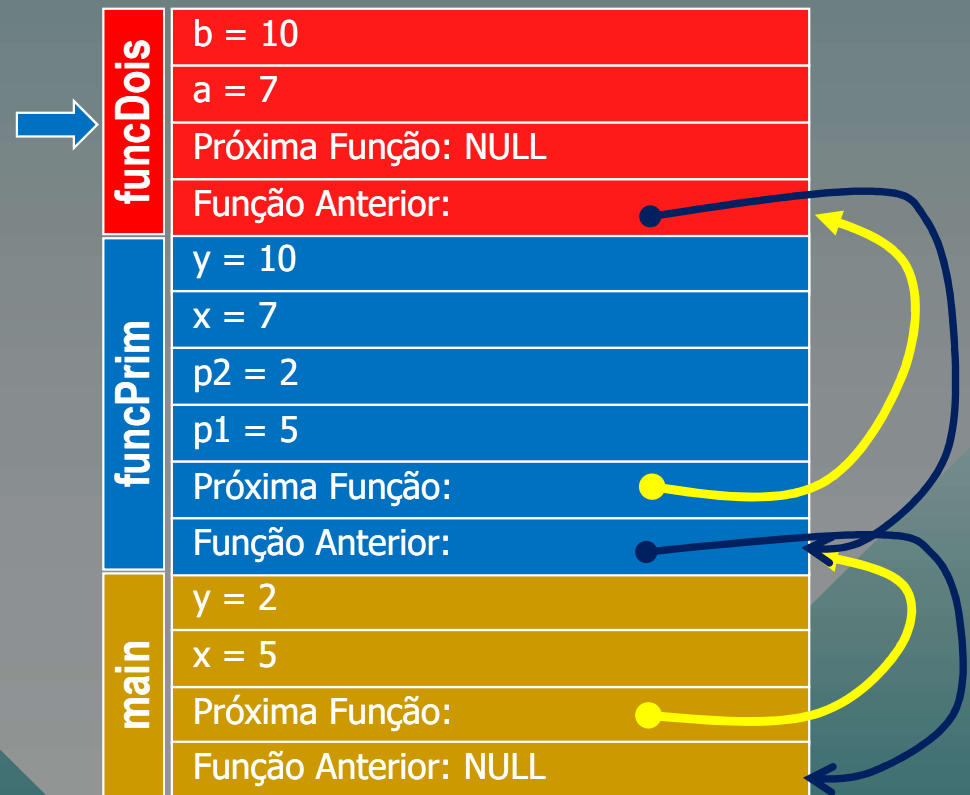
```
main()  
{  
    int x = 5;  
    int y = 2;  
    x = funcPrim(x, y);  
    y = x - 10;  
}  
  
int funcPrim(int p1, int p2)  
{  
    int x = p1 + p2;  
    int y = p1 * p2;  
    x = funcDois(x, y);  
    return x;  
}  
  
int funcDois(int a, int b)  
{  
    return b * a / 2;  
}
```



# Funcionamento da Stack

## Exemplo

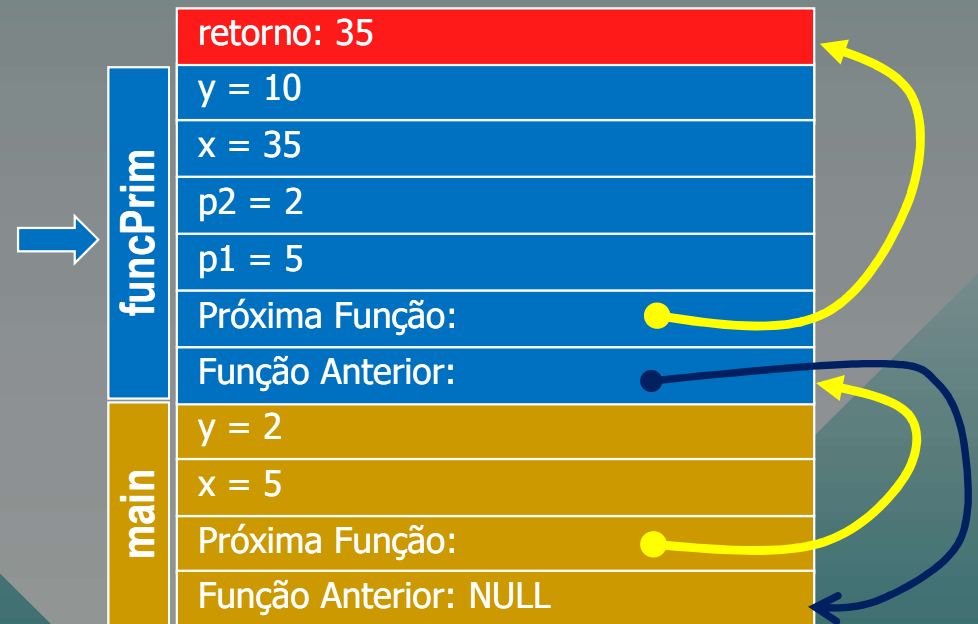
```
main()  
{  
    int x = 5;  
    int y = 2;  
    x = funcPrim(x, y);  
    y = x - 10;  
}  
  
int funcPrim(int p1, int p2)  
{  
    int x = p1 + p2;  
    int y = p1 * p2;  
    x = funcDois(x, y);  
    return x;  
}  
  
int funcDois(int a, int b)  
{  
    return b * a / 2;  
}
```



# Funcionamento da Stack

## Exemplo

```
main()  
{  
    int x = 5;  
    int y = 2;  
    x = funcPrim(x, y);  
    y = x - 10;  
}  
  
int funcPrim(int p1, int p2)  
{  
    int x = p1 + p2;  
    int y = p1 * p2;  
    x = funcDois(x, y);  
    return x;  
}  
  
int funcDois(int a, int b)  
{  
    return b * a / 2;  
}
```



# Funcionamento da Stack

## Exemplo

```
main()
```

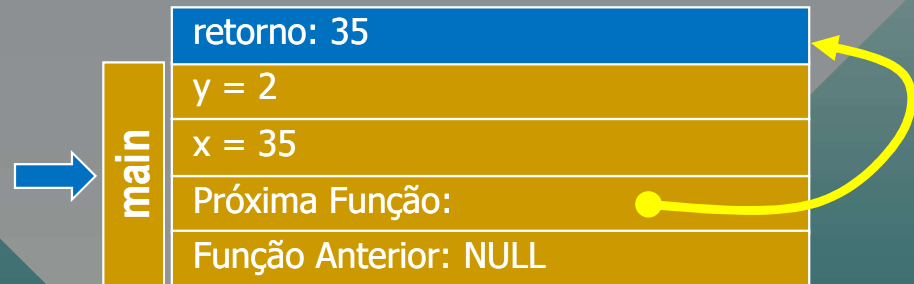
```
{  
    int x = 5;  
    int y = 2;  
    x = funcPrim(x, y);  
    y = x - 10;  
}
```

```
int funcPrim(int p1, int p2)
```

```
{  
    int x = p1 + p2;  
    int y = p1 * p2;  
    x = funcDois(x, y);  
    return x;  
}
```

```
int funcDois(int a, int b)
```

```
{  
    return b * a / 2;  
}
```






# Funcionamento da Stack

## Exemplo

```
main()
```

```
{  
    int x = 5;  
    int y = 2;  
    x = funcPrim(x, y);  
    y = x - 10;  
}
```



```
int funcPrim(int p1, int p2)
```

```
{  
    int x = p1 + p2;  
    int y = p1 * p2;  
    x = funcDois(x, y);  
    return x;  
}
```

```
int funcDois(int a, int b)
```

```
{  
    return b * a / 2;  
}
```



main	y = 25
	x = 35
	Próxima Função: NULL
	Função Anterior: NULL

# Histórico da Linguagem Java

- Java é um projeto da **Sun Microsystems** que foi chefiado por **James Gosling** e concebido inicialmente para aparelhos que possuísem processadores (TVs, microondas, etc).
  - **Linguagem Oak**
- Com o advento da **Web**, o projeto teve seu desenvolvimento reorientado.
- Java apresenta características adequadas para a criação de aplicações para ambientes corporativos.
- Site oficial: **[java.sun.com](http://java.sun.com)**

# Características da Linguagem Java

- *“Simples”*
- *Orientada a Objetos*
- *Distribuída*
- *Interpretada*
- *Robusta*
- *Neutra*
- *Segura*
- *“Alto desempenho”*
- *Portátil*
- *Paralelizável*
- *Dinâmica*

# Características da Linguagem Java

## “Simplicidade”

- Aprender Java não é tarefa simples. Porém torna-se mais fácil para quem domina Orientação a Objetos e/ou já programa com C ou C++, já que sua sintaxe possui alguns pontos em comum com estas linguagens.
- Em geral, **desenvolver um sistema em Java torna-se mais simples do que desenvolver em C++** pois não apresenta alguns de seus recursos “desaconselháveis” como:
  - **Aritmética de ponteiros** (ex: `*--pt = vet+5`)
  - **Estruturas** (structs)
  - **Definição de tipos** (typedef)
  - **Macros** (#define)
  - **Liberação explícita de memória** (free)
- Com isto, pode-se eliminar boa parte dos erros mais comuns e simplifica a codificação.

# Características da Linguagem Java

## Baseada em OO, Distribuída, Robusta

- **(OO)** Utiliza conceitos do **Modelo Orientado a Objetos**.
- **(Distribuída)** Aplicações que envolvem a **comunicação entre computadores através da rede** são chamadas de **Aplicações distribuídas**.

Java apresenta recursos para programação com **TCP/IP** (sockets), além disponibilizar outras tecnologias para criação de aplicações distribuídas como: **RMI, CORBA, Jini**, etc.

- **(Robusta)** Possui verificação em tempo de compilação e execução com o objetivo de gerar códigos confiáveis.
- Gerenciamento automático de memória c/ **Garbage Collection**

# Características da Linguagem Java

## Trecho de Código em Java (Exemplo de Garbage Collection)

- Considere o seguinte trecho de código:

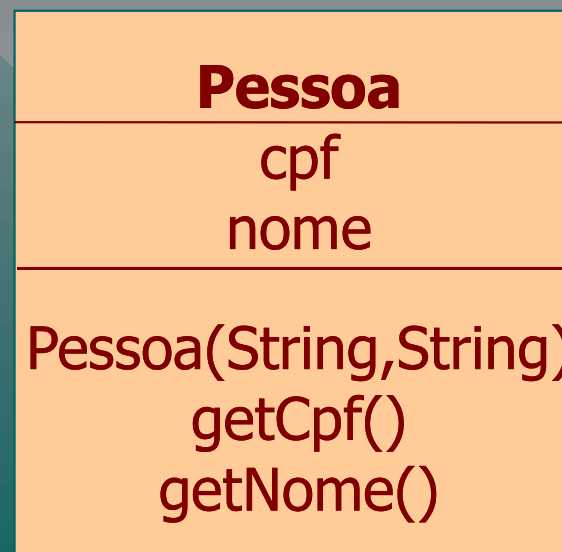
(1) **Pessoa** p1, p2, p3;

(2) p1 = **new** **Pessoa**("12345678-90", "José da Silva");

(3) p2 = **new** **Pessoa**("09876543-21", "Maria de Souza");

(4) p3 = p1;


- Também considere a classe Pessoa com a seguinte interface:





# Características da Linguagem Java

## Declaração de Variáveis (Exemplo de Garbage Collection)

- Na primeira linha estamos declarando **três variáveis locais** *p1*, *p2* e *p3* que são ponteiros para objetos da classe *Pessoa*. Como são **variáveis locais**, não podemos determinar o valor presente dentro delas. Entretanto, veremos à frente que em Java tudo que for alocado na **Heap** terá uma **inicialização default**; ou seja, tudo recebe automaticamente um valor padrão\*.

p1: 

p2: 

p3: 

Toda variável/atributo/parâmetro  
cujo tipo não for um tipo primitivo  
Na realidade é um ponteiro em Java

**Obs\*:** Para as variáveis locais, não podemos contar que a inicialização default (null) será aplicada.

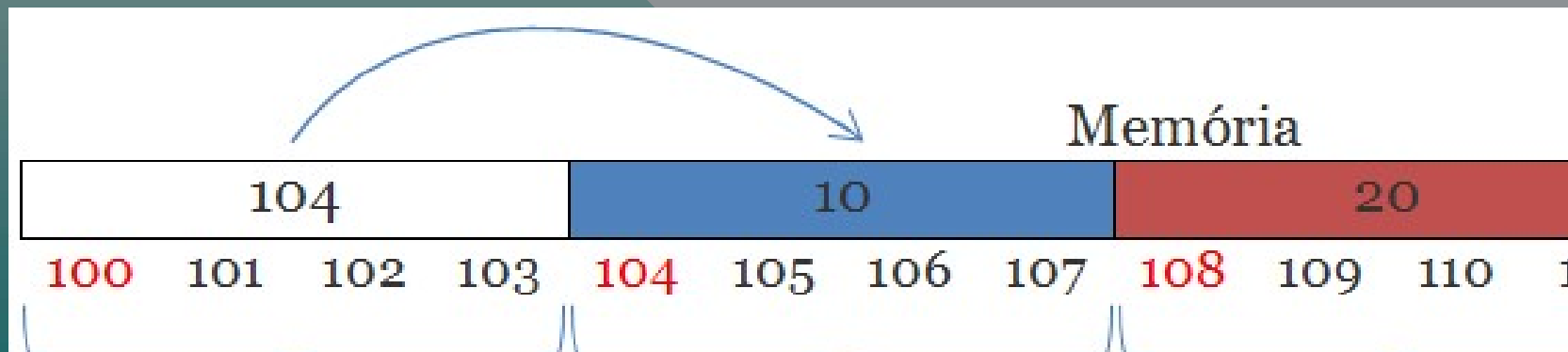
# Características da Linguagem Java

## Declaração de Variáveis (Exemplo de Garbage Collection)

- *Ponteiro (recordando...)*

- É um *tipo de dado* semelhante aos inteiros;
- Contudo o seu valor representa o *endereço de uma área* na memória RAM do computador.
- O *tipo associado ao ponteiro* representa *o tipo de dado que vamos encontrar na posição especificada* pelo ponteiro
- Exemplo na Linguagem C:

```
int* pt = &x; // pt é uma variável do tipo 'ponteiro para int' e que  
              // armazenará o endereço da variável x. Assim pt está  
              // apontando para x.  
  
int x = 10;
```





# Características da Linguagem Java

## Declaração de Variáveis (Exemplo de Garbage Collection)

- A declaração de variáveis e atributos segue a estrutura:

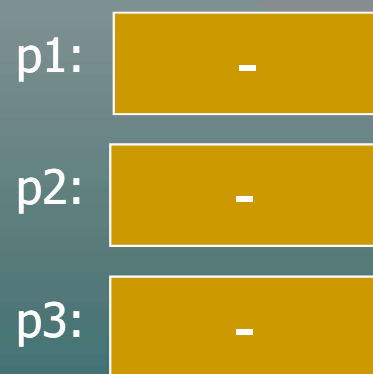
**<TIPO>** <nomeVar>, <nomeVar2>, ...;

- Os tipos primitivos de Java são: **byte**, **short**, **int**, **long**, **float**, **double**, **char** e **boolean**. Tudo o que não for rigorosamente tipo primitivo é um ponteiro para **<TIPO>**.

# Características da Linguagem Java

## Operadores (Exemplo de Garbage Collection)

- Na segunda linha temos dois operadores `=` e `new`. Pela **tabela de precedência**, o primeiro a ser executado é o `new`. Este é responsável pela **instanciação de um novo objeto** da classe `Pessoa`. Para isto, o operador realiza duas operações:
  - (1) O operador `new` aloca memória na **Heap** para o novo objeto, promovendo a **inicialização default** para os seus atributos.



Endereço de  
Memória  
**82AE4C**

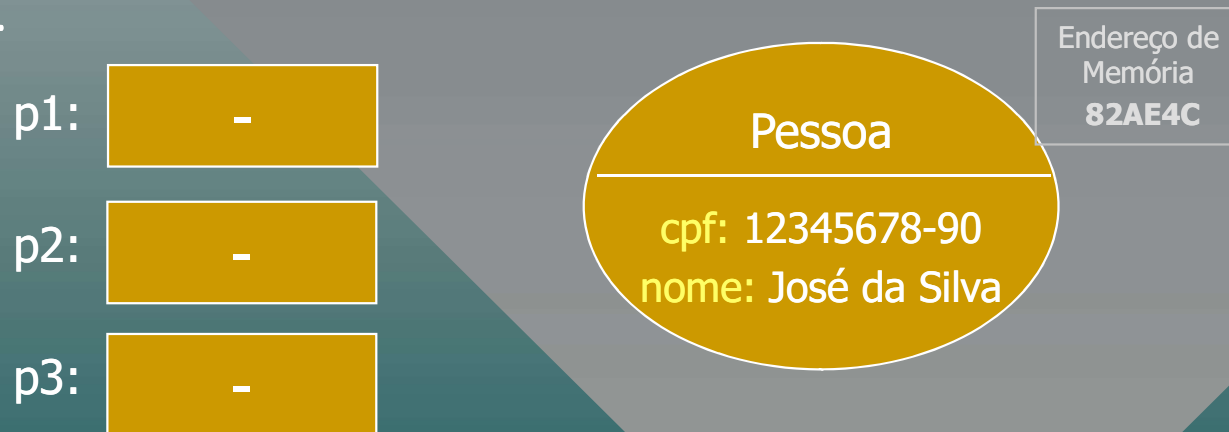
Observe que foi criado na memória (Heap) um novo objeto da classe `Pessoa` e que seus atributos foram inicializados com `"null"`.  
Considere que o endereço de memória onde este objeto foi criado é o `82AE4C`

# Características da Linguagem Java

## Operadores (Exemplo de Garbage Collection)

(2) O **new** envia uma mensagem para o novo objeto solicitando a execução do **método construtor** da classe Pessoa.

- **Método construtor** é um método que o programador coloca na classe cuja responsabilidade é inicializar o novo objeto. (deveria se chamar **método inicializador**)
- Quem executa o método construtor é o objeto recém-criado.
- Em Java, o método construtor é aquele que possui o mesmo nome da classe.



Observe que o objeto Pessoa agora apresenta os seus atributos com valores. Quem promoveu esta alteração foi o método construtor através dos parâmetros enviados pelo **new**

# Características da Linguagem Java

## Operadores (Exemplo de Garbage Collection)

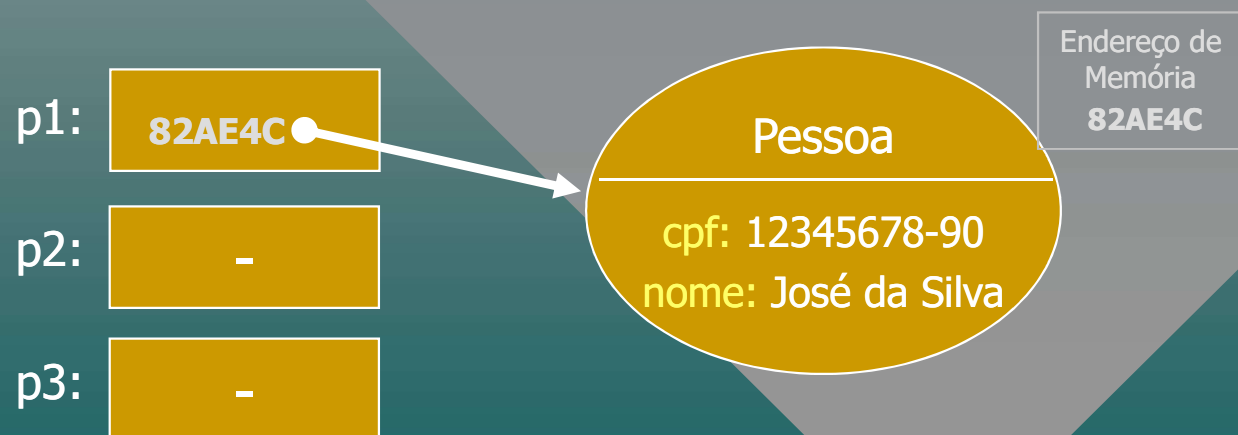
- **Dicas Importantes:**

- Imagine que cada objeto possui um **processador** capaz de executar os métodos definidos em sua classe. **Quando ele recebe uma mensagem**, o objeto **pega o código do método** associado à mensagem **e o executa** como se fosse um “**script**”.
- **Um método deve ser escrito de tal forma que qualquer objeto da classe possa executá-lo.** Veremos à frente que para escrevermos métodos é necessário uma forma de referenciar os atributos do objeto que estiver executando o método num determinado instante.
- Devemos considerar que o operador **new** **retorna ao final da sua execução o endereço de memória** onde foi alocado o novo objeto.

# Características da Linguagem Java

## Operadores (Exemplo de Garbage Collection)

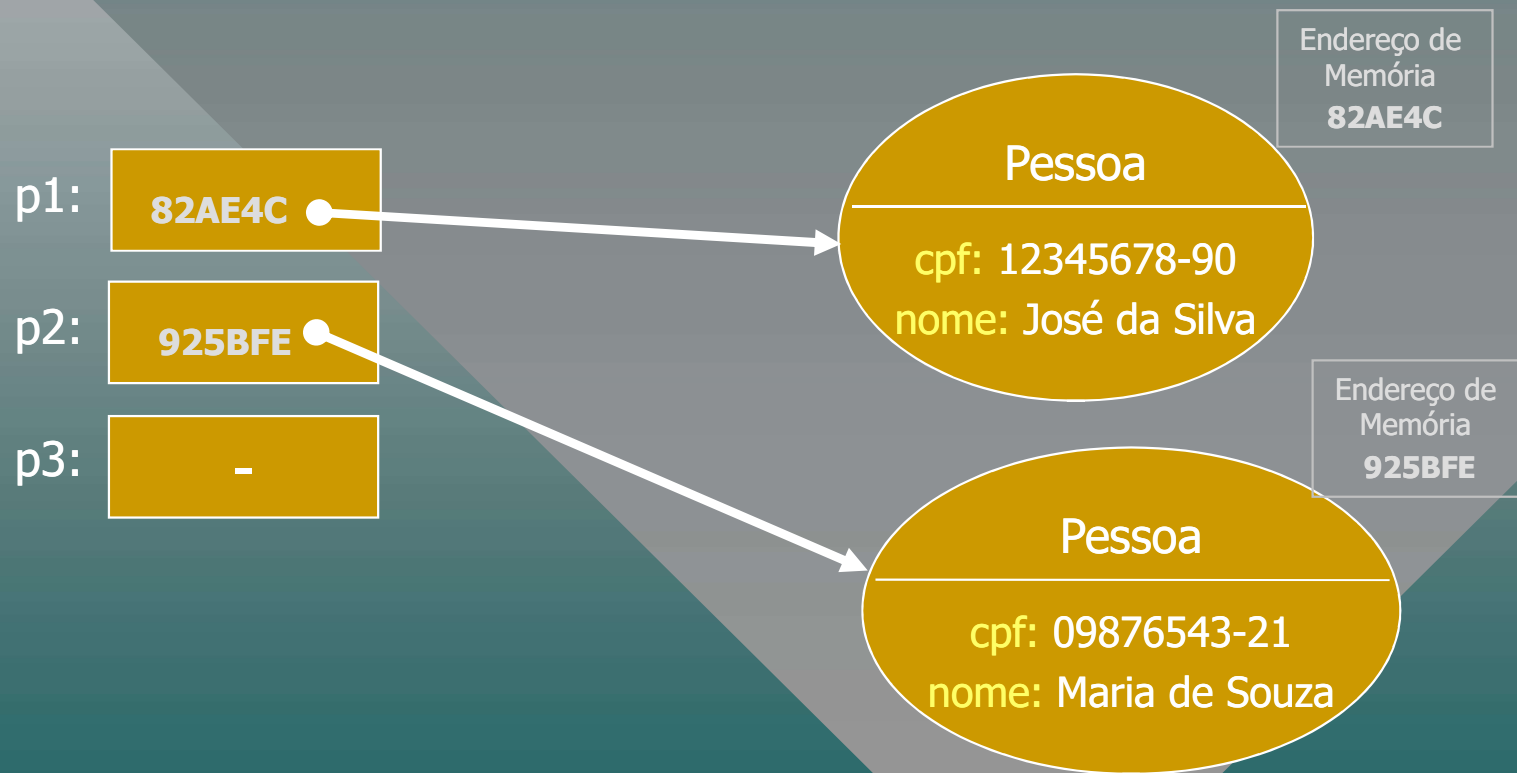
- Depois de instanciado, o **operador de atribuição (=)** será executado. Este fará com que **p1 passe a apontar** para o novo objeto.
  - Sempre que fizermos uma atribuição a um ponteiro (ou seja, quando o ponteiro estiver do lado esquerdo da atribuição), queremos que este **passe a apontar** para o endereço indicado ao lado direito.
  - **Uma variável ponteiro é semelhante a uma variável inteira.** Porém o número guardado faz referência a um endereço de memória e as operações com este tipo de variável sempre dizem respeito ao objeto apontado.



# Características da Linguagem Java

## Operadores (Exemplo de Garbage Collection)

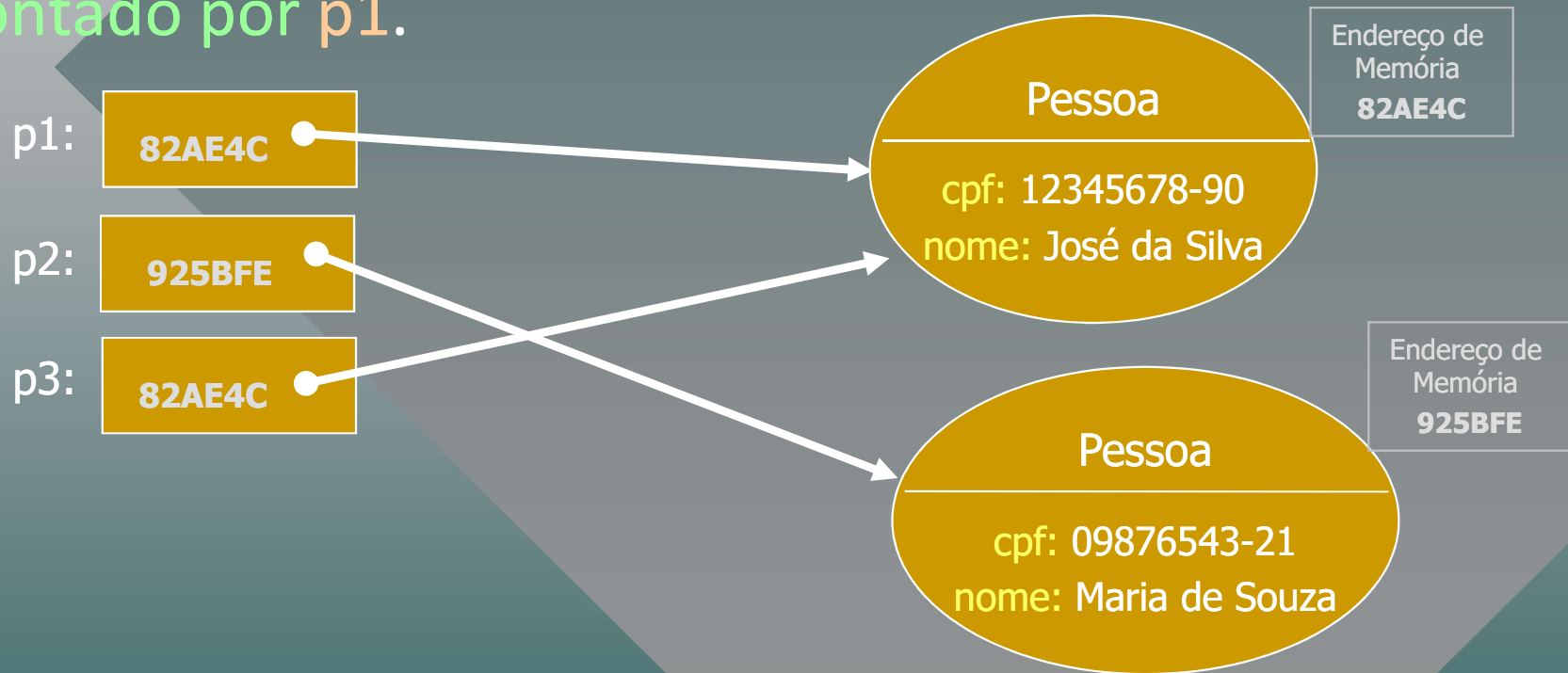
- Na terceira linha ocorre a mesma coisa que ocorreu na segunda linha, porém **p2** passa a apontar para o novo objeto.



# Características da Linguagem Java

## Operadores (Exemplo de Garbage Collection)

- Na quarta linha, *p3* passa a apontar para o objeto apontado por *p1*.

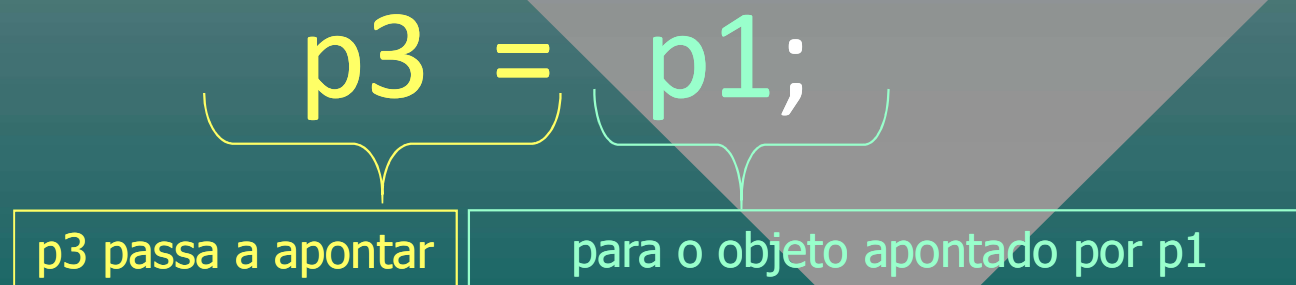


- Observe que todos os objetos alocados podem ser acessados através das variáveis declaradas.

# Características da Linguagem Java

## Operadores (Exemplo de Garbage Collection)

- Sempre que encontrarmos um ponteiro em um código Java, a semântica de seu uso será:  
“para o objeto apontado por *<ponteiro>*”.
- A única exceção é quando o ponteiro está do lado esquerdo da atribuição, cuja semântica é:  
“*<ponteiro>* passa a apontar”.
- Ex:





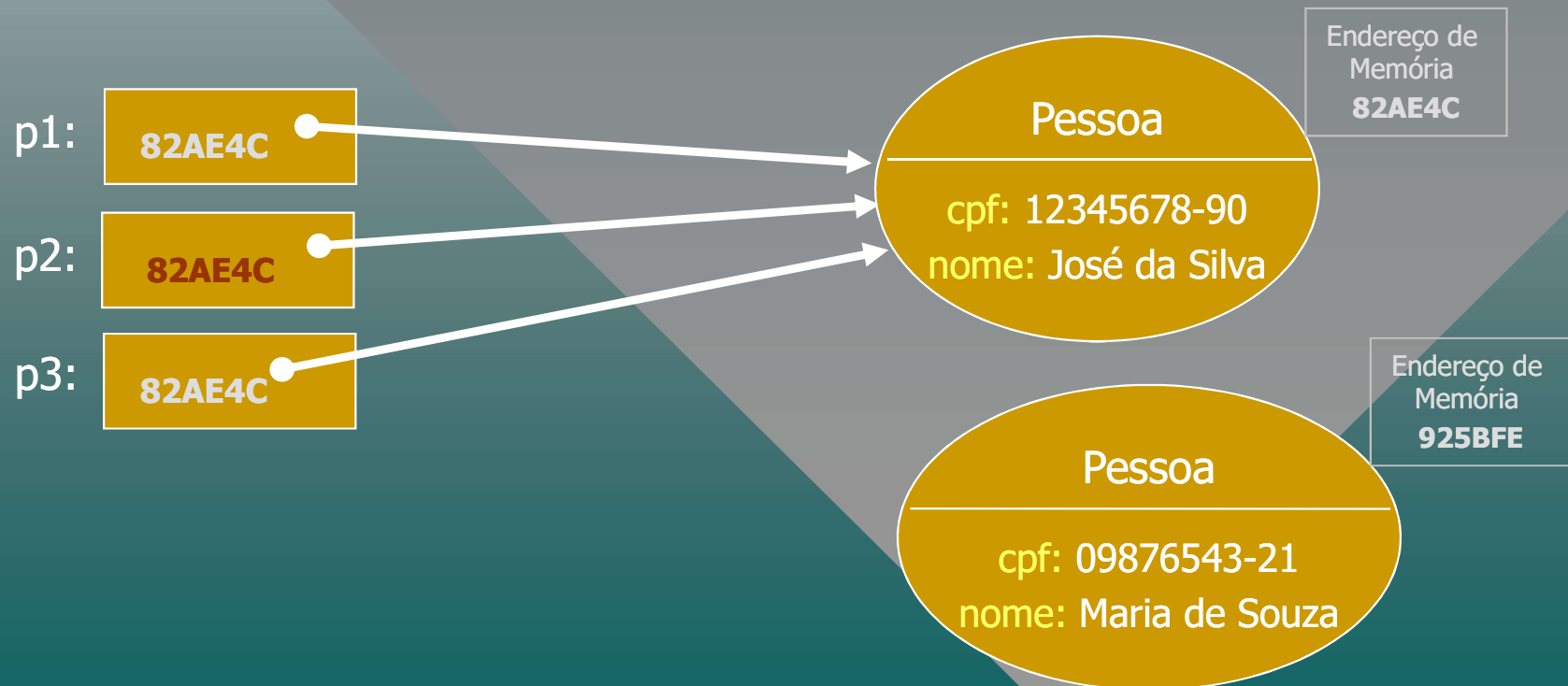
# Características da Linguagem Java

## Garbage Collection (Coletor de Lixo)

- Se adicionarmos uma quinta linha que faça:

**(5)** `p2 = p1;`

Veremos que estaremos perdendo a referência que tínhamos para o segundo objeto que foi alocado, e nunca mais poderemos ter acesso a este objeto (**LIXO**)



# Características da Linguagem Java

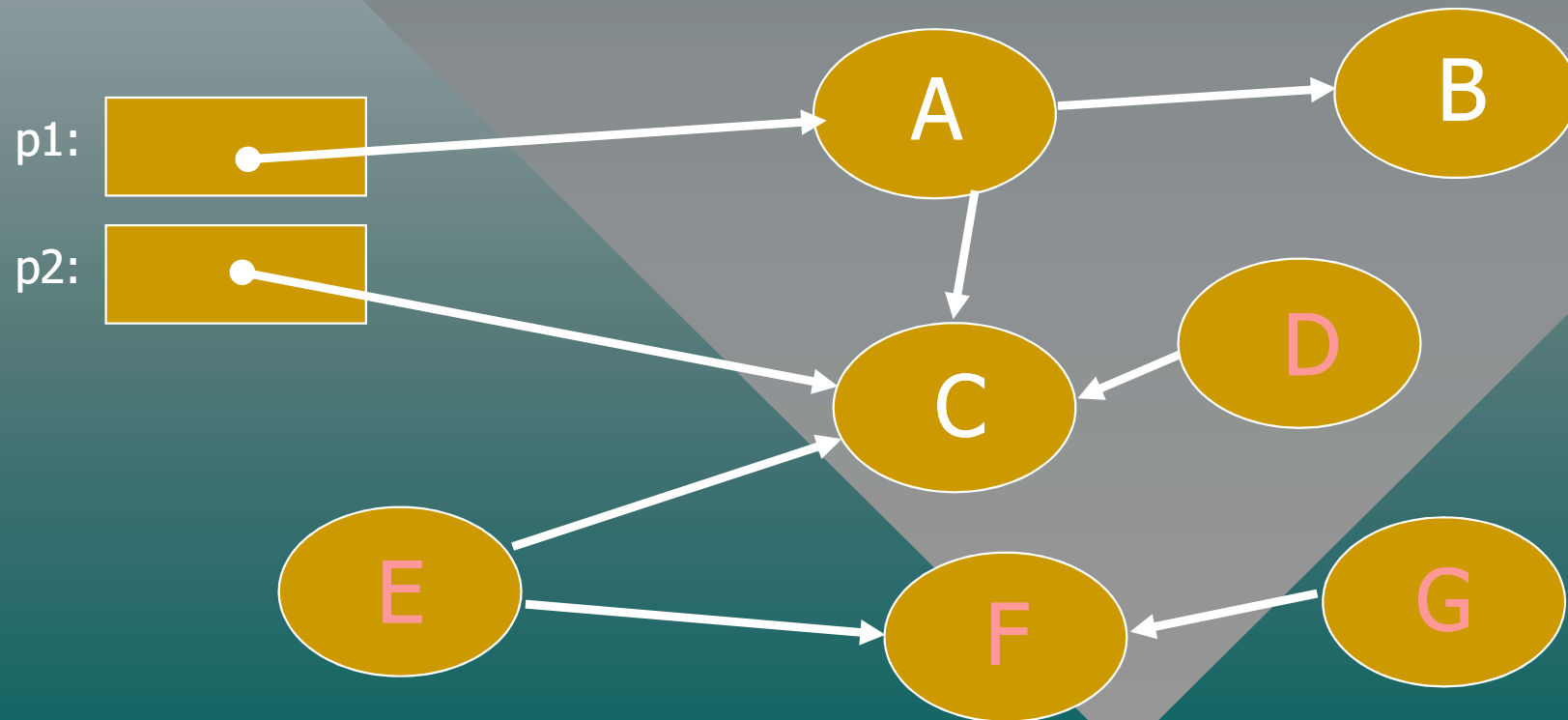
## Garbage Collection (Coletor de Lixo)

- Em linguagens como C++, é dever do programador codificar os procedimentos para **desalocação de memória**. Já em Java, não há esta necessidade pois esta tarefa é feita automaticamente pelo *Garbage Collection*.
- Este processo é executado por uma **thread** que roda em paralelo ao programa recolhendo os objetos “perdidos” e desalocando a memória destinada a eles.
- **Não sabemos quando exatamente o coletor irá rodar.** Porém quando for executado, o processo removerá todos os objetos considerados lixo.

# Características da Linguagem Java

## Garbage Collection (Coletor de Lixo)

- Consideramos um objeto **perdido** se não for possível recuperá-lo direta ou indiretamente a partir das variáveis
  - Ou seja, não há um caminho que ligue alguma variável ao objeto.
- Observe a situação abaixo. Após a execução do garbage collection, os objetos **D**, **E**, **F** e **G** seriam removidos.



# Características da Linguagem Java

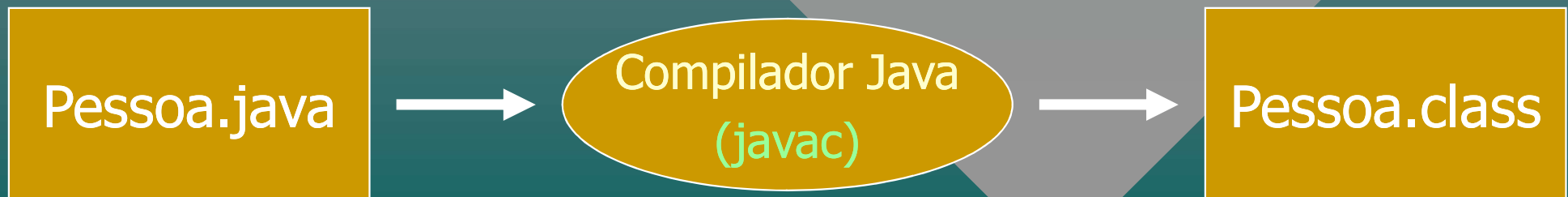
## Considerações

- Apesar do exemplo ter sugerido os endereços de memória, em Java não é possível saber qual é o endereço de memória onde um objeto está (aspecto de segurança).
- Mais a frente, veremos a representação das *Strings* na forma de objetos. Assim, os desenhos empregados simplificaram a realidade com o objetivo de clareza.

# Características da Linguagem Java

## Neutra, Compilada e Interpretada

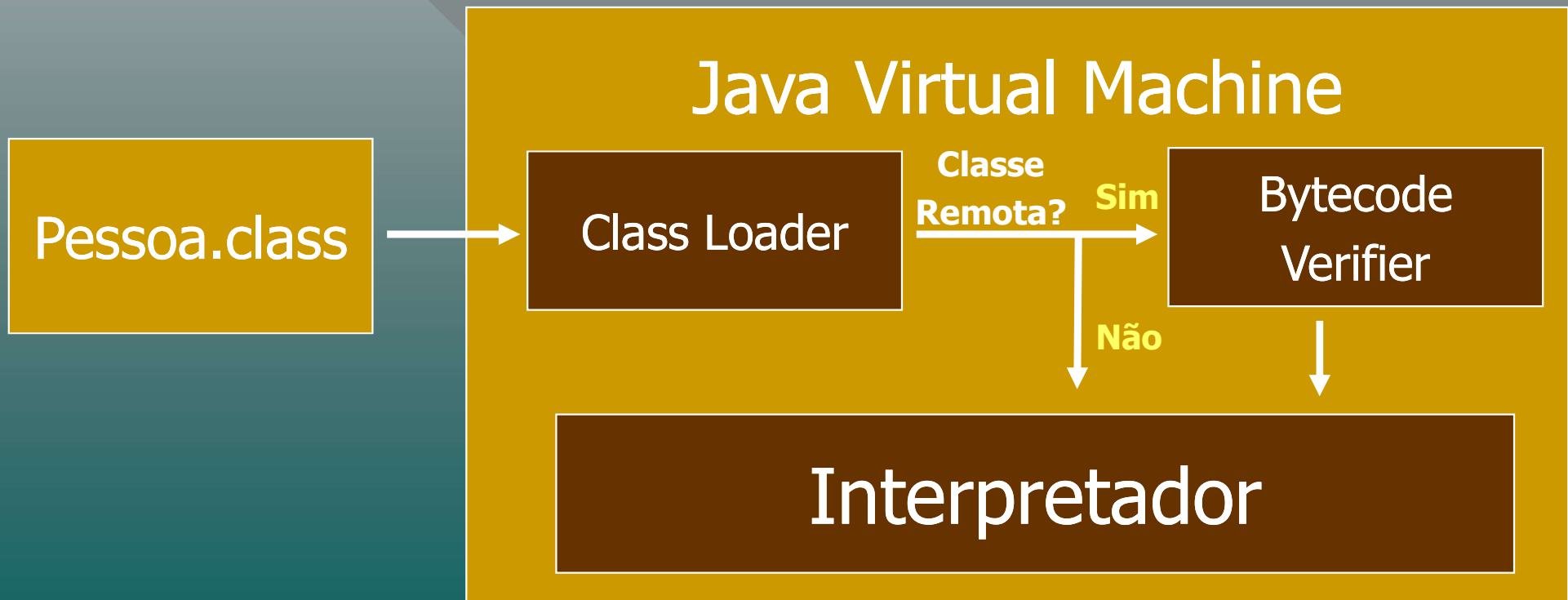
- Cada classe escrita em Java deve ficar em um arquivo cujo nome é formado pelo nome da classe + extensão **.java** (ex: Classe Pessoa → Pessoa.java)
- Ao submetermos este código ao compilador, ele não gera código de máquina, mas gera um código chamado de **Bytecode**, que é o código de máquina que só pode ser executado por um processador virtual (programa) chamado **JVM (Java Virtual Machine)**.
- Para cada classe compilada é gerado um arquivo com o bytecode produzido pelo compilador. O seu nome é formado pelo nome da classe + extensão **.class**.



# Características da Linguagem Java

## Neutra, Compilada e Interpretada

- Para executarmos o programa precisamos de uma JVM (software) que:
  - Carrega o bytecode com o *Class Loader*;
  - Se o bytecode veio de fonte externa (ex.Applet), ele passa por um *Verificador de ByteCode* (segurança!)
  - O interpretador repetidamente lê uma instrução e a executa.



# Características da Linguagem Java

## Neutra, Compilada e Interpretada

- Genericamente, um **Interpretador** é um programa capaz de ler e executar um programa; para isto, ele continuamente **lê uma linha, verifica aspectos léxicos/sintáticos/semânticos, examina quais instruções estão associadas à linha e as executa**.
  - Ex: **JavaScript** (Cada navegador web possui um interpretador desta linguagem capaz de ler e executar as funções embutidas na página HTML)
- Em Java, **o processo de interpretação é muito mais simples que o relatado acima**, pois muitas das etapas já foram feitas pelo compilador Java (ex. verificação da sintaxe) e as instruções presentes no bytecode são de fácil entendimento.
- Podemos afirmar que **o processo de execução de programas realizado pelo Interpretador Java é muito semelhante ao que uma CPU faz para executar um programa .EXE em memória**.

# Características da Linguagem Java

## (Curiosidade) Tecnologias das JVMs

- Em JVMs com tecnologia **HotSpot** (mais comum), há a presença do interpretador.
- Entretanto há uma **exceção** que são as JVMs com tecnologia **JIT (Just in Time)**, o bytecode é convertido para código nativo em tempo de execução objetivando-se melhorar o desempenho do programa. Somente neste caso não há processo de interpretação.
- Estudos mostram que o desempenho de JVMs JIT são praticamente similares ao das JVMs HotSpot.
- Podemos dizer que Java é uma linguagem **neutra** pois o **bytecode produzido não está vinculado a nenhum processador/sistema operacional específico**.



# Características da Linguagem Java

## Neutra, Compilada e Interpretada

- Sumarizando:
  - Java é uma linguagem tanto **compilada** quanto **interpretada** pois cada classe em Java deve ser submetida ao **Compilador Java** que, ao invés de gerar um código “assembler” para um processador específico, gera um código chamado **bytecode** que não está vinculado a nenhum processador ou SO (por isso dizemos que Java é uma linguagem **neutra**).
  - Para executarmos um programa em Java, necessitamos um **Interpretador Java** que leia cada código do bytecode e o execute.
  - A grande vantagem de Java apresentar estas características é a **PORTABILIDADE**

# Características da Linguagem Java

## Segura, Portátil e “Alto Desempenho”

- **(Segura)** Apesar de termos variáveis que são ponteiros, Java não nos deixa manipular (ler ou alterar) com o valor do endereço.
  - Extensiva verificação do bytecode (presença do mecanismo CRC).
- **(Portátil)** Roda em qualquer plataforma.
  - Além de neutra, implementa tipos de dados com tamanho fixos (ex. *Int* 's têm 32 bits).
- **(“Alto desempenho”)** Apesar de Java ser interpretada, os programas rodam rapidamente pois o processo de interpretação é simples.

# Características da Linguagem Java

## Paralelizável e Dinâmica

- **(Paralelizável)** Suporta programação de *threads* (subprocessos que rodam “paralelamente” ou “concorrentemente”, dependendo da plataforma).
- **(Dinâmica)** As classes são carregadas em tempo de execução, de acordo com a necessidade (ou seja, a JVM só carrega uma classe quando houver necessidade).
  - Estas classes podem ser atualizadas separadamente (**alta coesão e baixo acoplamento**)
- Além destas características, encontramos em Java uma série de tecnologias que permitem criar sistemas para uma série de aplicações.

# Nomenclaturas

- **JDK – Java Development Kit**
  - Conjunto de ferramentas (ex. Compilador, Interpretador, Jar, Javadoc, etc.), classes (Java Platform Core Classes) e documentação para fazer o desenvolvimento em Java.
- **JFC – Java Foundation Classes**
  - Conjunto de classes que compõem a base para implementação de interfaces gráficas.
- **Java 2**
  - Nome dado à versão 1.2 do JDK e referência a um padrão de Core Classes.
- **Java 5 (Tiger)**
  - Nome dado à versão 1.5 (agora chamada de 5.0) do JDK e referência a um padrão de Core Classes. Nesta versão foram acrescentados novos recursos à linguagem. A versão
- **SDK – Java Software Development Kit**
  - Sinônimo para JDK
- **Java Platform Core Classes**
  - Conjunto de classes que compõem a base de uma versão de Java.
- **JVM – Java Virtual Machine**
  - Máquina Virtual Java (“Interpretador” e demais itens)

# Nomenclaturas

- **JSE – Java Platform Standard Edition**
  - Centro da tecnologia Java
  - Conjunto básico para o desenvolvimento de aplicações Java.
- **JEE – Java Platform Enterprise Edition**
  - Conjunto de especificações e práticas que apoiam o desenvolvimento e implantação de aplicações “multi-tier”
  - Construído sob o JSE (Standard Edition);
  - **Tecnologias:** EJB, Servlets, JAXR (XML), Corba, etc.
- **JME – Java Platform Micro Edition**
  - Plataforma para desenvolvimento de aplicações para pequenos equipamentos eletrônicos.
- **JRE – Java Runtime Environment**
  - Interpretador simplificado (ex. sem apoio à depuração) para fazer implantação de aplicações + Java platform core classes.

# Ferramentas para Desenvolvimento

- A Sun Microsystems fornece gratuitamente o **JDK (Java Software Development Kit)** que é composto vários utilitários como: compilador, interpretador e bibliotecas (pacotes).
- A interface do JDK é **baseada em linha de comandos** de Shell
- Existem outras ferramentas como:
  - Eclipse
  - NetBeans
  - BlueJ
  - Kawa
  - Borland JBuilder
  - WSAD (WebSphere Application Development)

# Primeiro Exemplo

## Trabalhando somente com o JDK

- Crie um diretório **C:\Java\AloMundo** (pasta do projeto)
- Crie o diretório **C:\Java\AloMundo\controle** (pacote)
- Crie o arquivo **Programa.java** (Cuidado: case-sensitive!)

```
package controle;  
public class Programa  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Alo, mundo!");  
    }  
}
```

- Para compilar, vá para C:\Java\AloMundo:

C> **javac controle\Programa.java** (gera o arquivo Programa.class)

- Para Executar:

C> **java controle.Programa**  
Alo, mundo!



# Primeiro Exemplo

## Introdução ao Eclipse

- Ambiente **extensível**, **portável** e **aberto** que permite a integração de várias ferramentas de desenvolvimento (não somente codificação!).
- **Plug-in**
  - Módulos que permitem a extensão das funcionalidades do Eclipse
- **Perspectiva**
  - Conjunto de visões e editores organizados para apoiar um certo tipo de atividade.
  - Principais perspectivas: **Java**, **Java Browsing**, **Debug**, **Resources**, **CVS** (Sistema de Controle de Versões).
  - Ao instalar novos plug-ins, novas perspectivas podem ficar disponíveis.
- **Workspace**
  - Pasta que contém um ou mais projetos
- **Projeto**
  - Pasta dentro do workspace que representa um programa ou um módulo. Pode conter código e outros recursos.
- **Package**
  - Subpasta do projeto que contém um conjunto de classes que tratam de um mesmo assunto (a princípio)



# Primeiro Exemplo

## Introdução ao Eclipse

The screenshot shows the Eclipse IDE with the Java perspective. The interface includes a menu bar, a toolbar, and several views. The Package Explorer on the left shows the project structure, and the Editor on the right shows the code for 'ModeloDAO.java'.

Labels and their corresponding elements in the IDE:

- Projeto Java**: Points to the 'MeuProjeto' project in the Package Explorer.
- Pacote**: Points to the 'trabalho.model' package in the Package Explorer.
- Classe**: Points to the 'ModeloDAO.java' file in the Package Explorer.
- Atributo/Constante**: Points to the 'DRIVER' attribute in the 'ModeloDAO' class in the Package Explorer.
- Método**: Points to the 'criarConexao' method in the 'ModeloDAO' class in the Package Explorer.
- Editor**: Points to the code editor window showing the 'ModeloDAO.java' file.
- Barra de Perspectivas**: Points to the perspective bar at the top right of the IDE.

The code in the Editor is as follows:

```
/**
 * Cria uma nova conexão usando os parâmetros.
 */
public static Connection criarConexao(String url, Properties
{
    Connection conexao = null;

    try
    {
        conexao = DriverManager.getConnection(url, parametros);
    }
    catch (Throwable e)
    {
        e.printStackTrace();
        conexao = null;
    }

    //devolve a resposta, se conectou ou não
    return conexao;
}
```

## Eclipse na Perspectiva Java

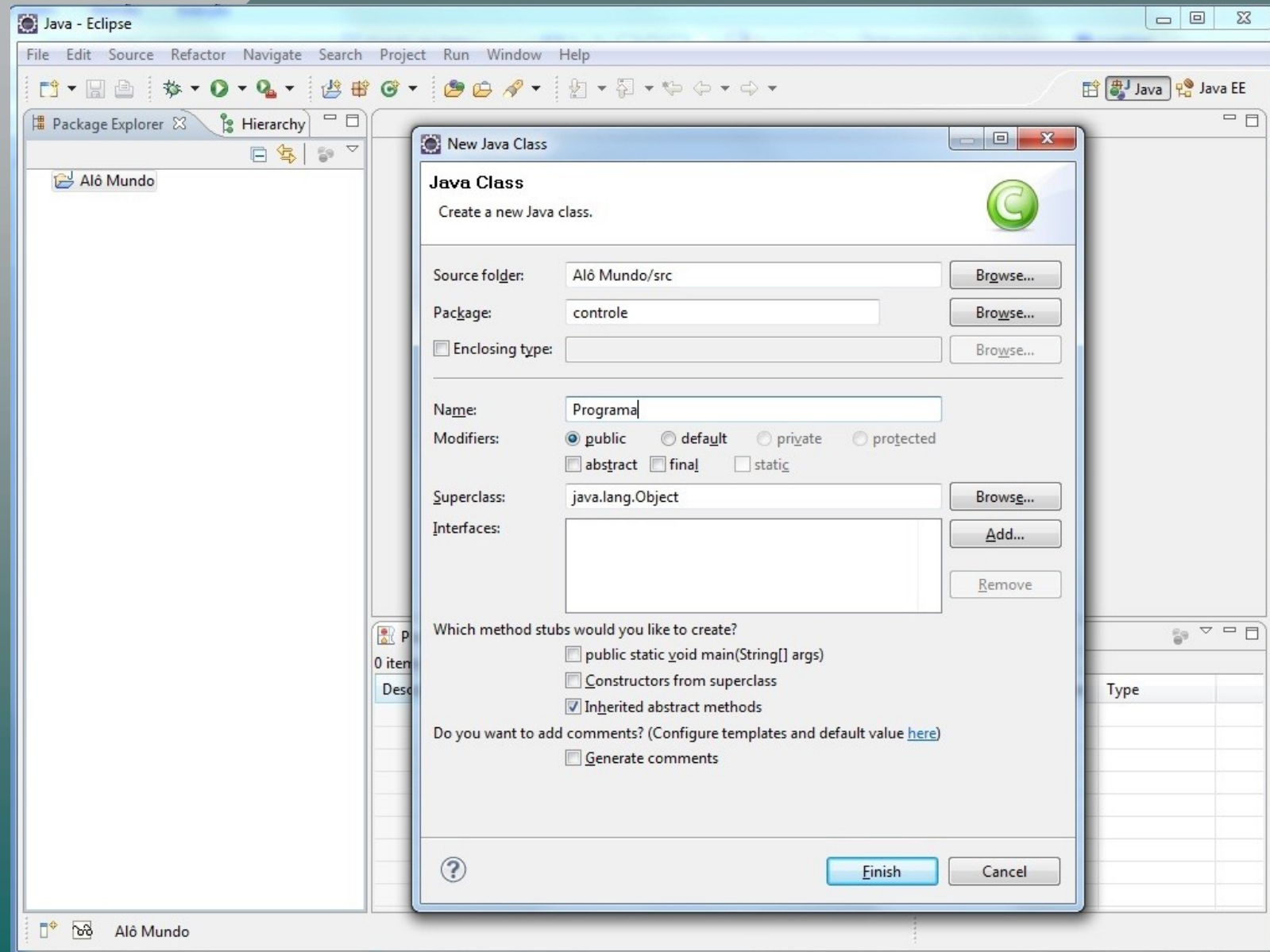
# Primeiro Exemplo

## Projeto no Eclipse

- Abra o workspace **C:\Java\workspace** e habilite a a perspectiva **Java**
- Criando um Projeto
  - **[File][New][Project]** e no assistente marcar **[Java Project]** e **[Next]**
  - Project Name: **Alô Mundo** e **[Finish]**
- Crie a classe Programa
  - **[File][New][Class]**
  - Package: **controle** (em letras minúsculas)
  - Name: **Programa** (P Maiúsculo)
- Ao salvar a classe (Ctrl+S), o Eclipse faz a compilação automática.

# Primeiro Exemplo

## Projeto no Eclipse



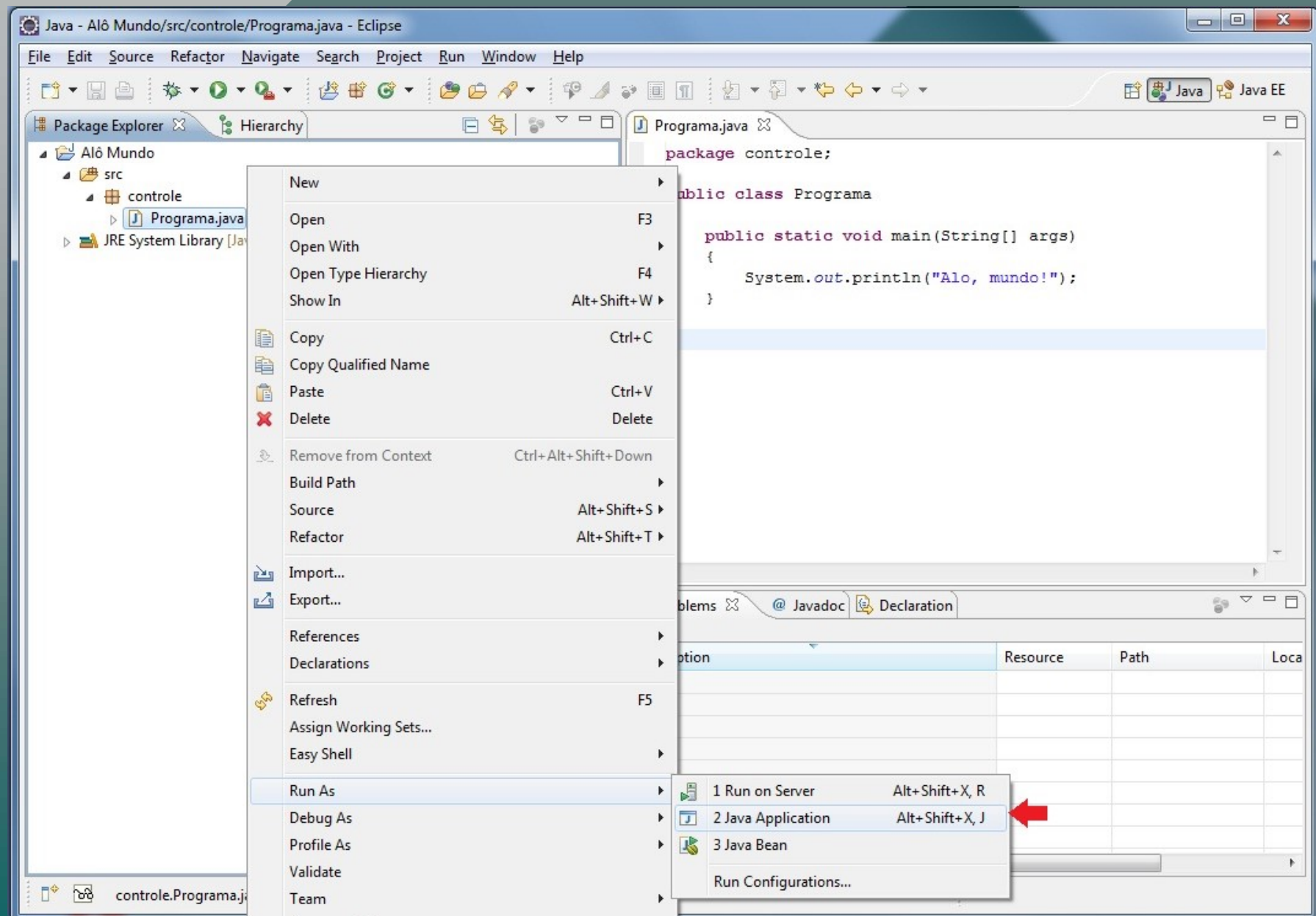
# Primeiro Exemplo

## Projeto no Eclipse

- Para Executar:
  - Clicar com o botão direito sobre a classe *Programa* na área do *Package Explorer*
  - Selecionar a opção *[Run As][Java Application]*
  - A entrada/saída da execução se dá através da *[View][Console]*

# Primeiro Exemplo

## Projeto no Eclipse

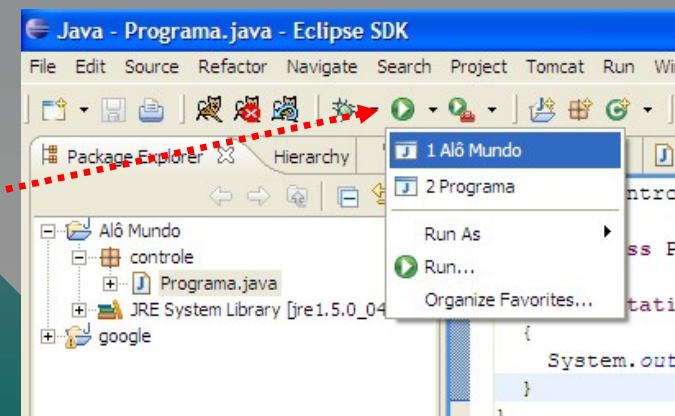
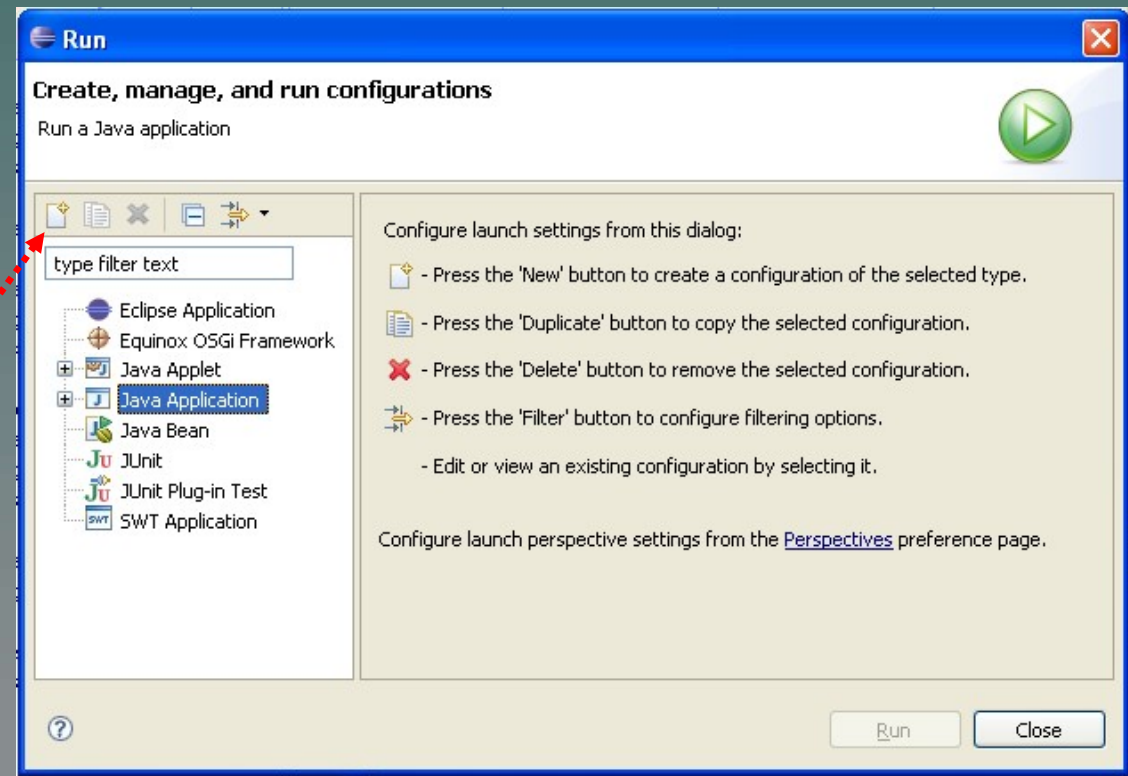




# Primeiro Exemplo

## Projeto no Eclipse

- Para executar, também podemos criar uma “**Configuração de Execução**”:
  - [Run][Run]
  - Marcar do lado esquerdo [Java Application] e pressionar o botão [New]
  - Name: **Alô Mundo** (nome da configuração)
  - Project: **Alô Mundo**
  - Main Class: **controle.Programa** (classe que contém o método main)
  - [Run] para executar
- Para executar novamente, basta utilizar o botão [Run] na barra de ferramenta e selecionar a configuração desejada.



# Tipos de Programas Java

## *Applets*

- **Applets** são programas voltados para a Web que são executados (geralmente) a partir de navegadores WWW sob alguma página HTML.
  - Algo que se assemelha ao funcionamento das **Applets** nas páginas HTML são os programas escritos em **Flash**. Entretanto, **Flash** é muito mais utilizado para animações, enquanto **Applets** são mais utilizadas em outras situações (ex. teclado virtual de home banking).
- Applets possuem algumas **limitações por razões de segurança**.
- Para rodar uma applet, o navegador Web precisa ter ao seu alcance uma **JVM** ou **JRE**.
- Disponibilizam dinamismo e interatividade em páginas da Web.
- Não possuem o método *main*.

# Primeiro Exemplo em Applet

```
import java.awt.Graphics;  
import java.applet.Applet;  
  
public class Ex01Applet extends Applet  
{  
    public void paint (Graphics g)  
    {  
        g.drawString("Alo, mundo!", 25,25);  
    }  
}
```

- Observe:
  - Inexistência do método **main** e sim o método **paint**
  - Criação da classe Ex01Applet a partir da especialização da classe Applet (**extends**).
  - Uso do modo gráfico.



# Páginas HTML com Applets

- Uso do cláusula **<APPLET>**
  - Exemplo:  
`<APPLET CODE=Ex01Applet.class WIDTH=100 HEIGHT=100>`  
Texto exibido se o navegador não suportar Java  
`</APPLET>`
- Para testar applets sem um navegador, utilizar o utilitário *appletviewer*:

```
C> appletviewer Ex01Applet.html
```

O *appletviewer* não interpreta o restante das cláusulas HTML (somente a tag `<Applet>`)

# Primeiro Exemplo em Applet

## Applets no Eclipse

- Criar o projeto e a classe da mesma forma que um projeto tradicional.
- Ao se criar a configuração de execução, a única diferença consiste em selecionar a opção **[Java Applet]** ao invés de **[Java Application]**

# Instalações Necessárias

## JDK

- Primeiramente, deveremos instalar um JDK. Basta pegar o programa instalador e clicar em [Next] todas as vezes que for solicitado.
- Num processo normal, o JDK será instalado na pasta  
C:\Arquivos de Programas\Java\jdk1.5.0\_06
- Também é instalado uma versão do JRE na pasta  
C:\Arquivos de Programas\Java\jre1.5.0\_06

# Instalações Necessárias

## Configuração das Variáveis de Ambiente

- Após a instalação do JDK, é necessário fazer as configurações das variáveis de ambiente **JAVA\_HOME** e **PATH**
- Uma **Variável de Ambiente** é uma informação que fica disponível no Sistema Operacional e que pode ser recuperada pelos programas em execução.
- A variável **JAVA\_HOME** indica onde está instalado o JDK
- A variável **PATH** informa para o *Prompt de Comando* quais são as **pastas onde um determinado executável pode se encontrar** caso não esteja na pasta corrente.

# Instalações Necessárias

## Configuração das Variáveis de Ambiente

- Ir em [Painel de Controle], ícone [Sistema], guia [Avançado], botão [Variáveis de Ambiente].
- Para as configurações possam valer para qualquer usuário do computador, trabalhe com o painel [Variáveis do Sistema].
- Clique no botão [Nova]. No *nome da variável* preencha “**JAVA\_HOME**” e no *valor da variável* coloque a pasta onde o JDK está instalado (provavelmente deve se encontrar em **C:\Arquivos de Programas\Java\jdk1.5.0\_06**)
- Procure agora a variável “**PATH**” e a selecione. Clique no botão [Editar]. No *valor da variável*, acrescente o seguinte conteúdo:  
**;%JAVA\_HOME%\bin**

# Instalações Necessárias

## Eclipse

- Para instalar o Eclipse, basta copiar o conteúdo do CD de instalação para **c:\bin\eclipse**.
- Todos os plugins necessários já estão configurados.
- Crie um atalho na **Área de Trabalho** apontando para **c:\bin\eclipse\eclipse.exe**.
- Para verificar se as configurações estão corretas, basta executar o Eclipse.