

Material IsiFLIX para uso exclusivo de
Nelson de Campos Nolasco
nelsonnolasco@gmail.com



Java Fundamentals

Aula 1 - Introdução e Conceitos

Prof. Dr. Francisco Isidro

isidro@professorisidro.com.br

- Justificativa
 - Apresentar técnicas de programação Orientada a Objetos com implementações em uma linguagem de programação.






















- Objetivo
 - Capacitar o aluno a desenvolver aplicações através do paradigma Orientado a Objetos.
 - Capacitar o aluno a utilizar Java para resolução de problemas computacionais

- Java e Máquina Virtual
- Instalação e Configuração
- Fundamentos da Linguagem - Variáveis, tipos de dados, estrutura de um programa
- Fundamentos da Linguagem - Decisões, Repetições
- Orientação a Objetos - classes, objetos, atributos e métodos
- Resolvendo problemas e Pensando orientado a Objetos
- Herança e Polimorfismo
- Resolução de Problemas com Herança
- Tratamento Exceções
- Arquivos
- Lambdas e Streams

- Básica
 - DEITEL, H, DEITEL, P. Java - Como Programar. Ed. Pearson, 8a Edição, 2009
 - SIERRA, K, BATES, B. - Use a Cabeça! Java. Alta Books. 2005

Introdução

- A linguagem Java tem se consolidado como uma das mais utilizadas no meio corporativo e acadêmico atualmente

Language Rank	Types	Spectrum Ranking	Spectrum Ranking
1. Java	  	100.0	100.0
2. C	  	99.9	99.3
3. C++	  	99.4	95.5
4. Python	 	96.5	93.5
5. C#	  	91.3	92.4
6. R		84.8	84.8
7. PHP		84.5	84.5
8. JavaScript	 	83.0	78.9
9. Ruby	 	76.2	74.3
10. Matlab		72.4	72.8

fonte: IEEE

- Virtual Machine
 - Ambiente de execução para código gerenciado (Bytecode)
 - Atua como um Sistema Operacional real, gerenciando recursos, I/O, Memória, processos
- Orientação a Objetos
 - Recursos desse paradigma
 - Não possui herança múltipla (vamos discutir mais à frente)

- Distribuída
 - Comunicação facilitada através de redes de computadores
 - Execução remota de recursos (RMI)
 - Criação de aplicações server-side (Servlets, JSP, etc.)

- Segurança
 - Nada é seguro!!!
 - Sistemas de segurança na JVM
 - Assinatura digital
- Grande eficácia
 - Controle para evitar estouro de pilha
 - Controle para evitar corrupção de memória
 - Controle de Escrita ou leitura de arquivos locais

Características

- Múltiplas linhas de execução
 - Facilidade de construção de threads
 - Atraente para aplicações servidores
- Dinâmica
 - Adapta-se para o ambiente de execução
 - Adiciona classes, métodos, variáveis em tempo de execução

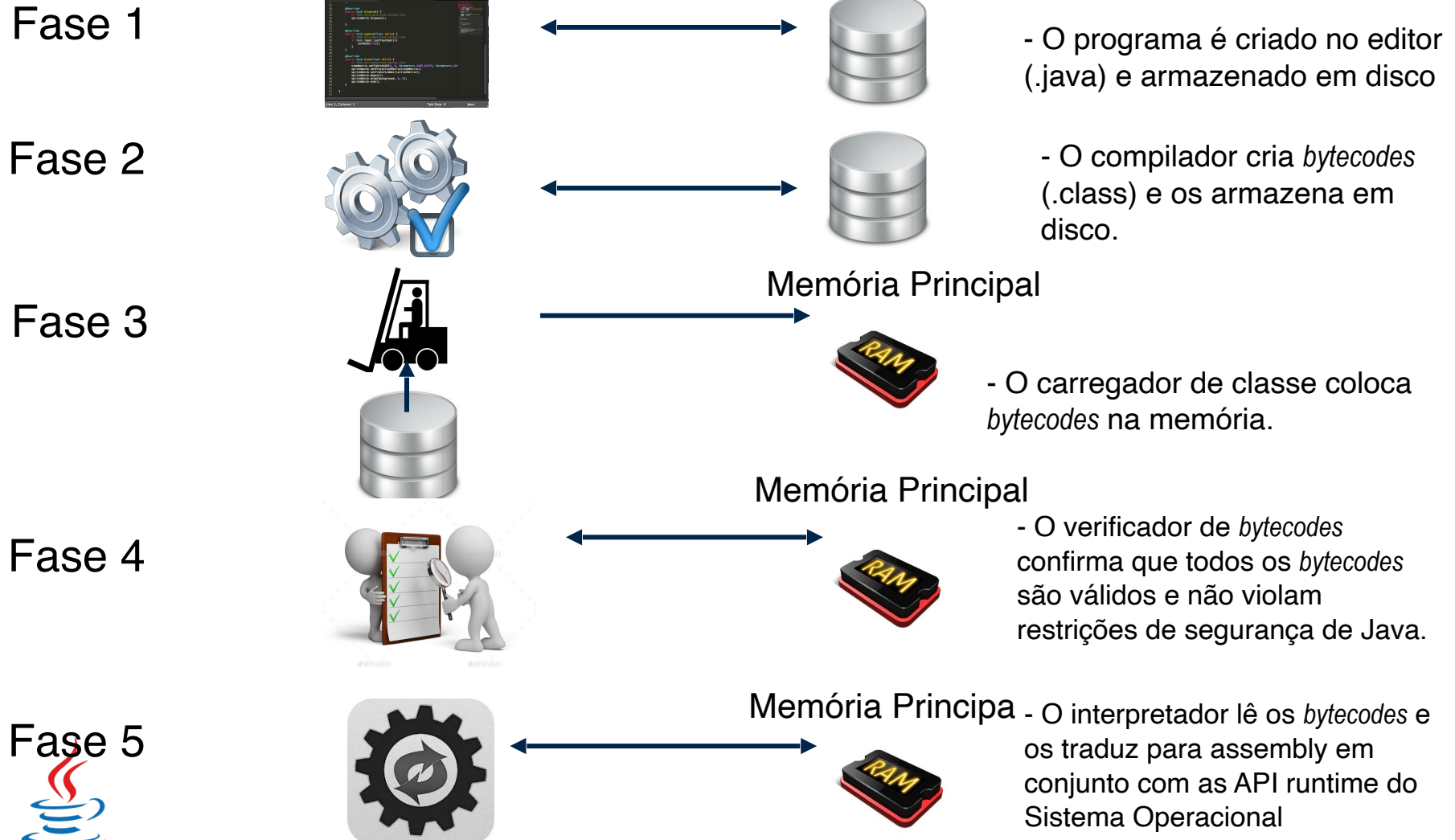
- Bytecode
 - Código intermediário, pré-compilado e pronto para ser interpretado
- Coletor de lixo (Garbage collection)
 - Recurso do Java para gerenciamento de memória, liberando memória que não é mais necessária

- JavaScript
 - Linguagem de scripts para páginas Web; apesar do nome e da sintaxe, não tem nada a ver com a programação em Java
- Lenda 1: Java é extensão do HTML
 - HTML é uma linguagem de marcação de hipertexto
 - Java é uma linguagem de programação
 - O que Java e Javascript tem em comum?
 - As 4 primeiras letras!

Cuidado!!!

- Lenda 2: Java é fácil
 - Nenhuma linguagem de programação é fácil
 - Implementar os conceitos de OO não é simples

Ambiente Java Típico



Java Software Development Kit

- O ambiente de desenvolvimento de *software* Java, Java SDK (antigamente, JDK), é formado essencialmente por:
 - um conjunto de aplicativos que permite, entre outras tarefas, realizar a compilação e a execução de programas escritos na linguagem Java.
 - um amplo conjunto de APIs que compõem o núcleo de funcionalidades da linguagem Java.
 - Uma API (*Application Programming Interface*) é uma biblioteca formada por código pré-compilado, pronto para ser utilizado no desenvolvimento de suas aplicações.

Java Software Development Kit

- As ferramentas básicas do kit de desenvolvimento Java são:
 - o compilador Java, javac,
 - o interpretador de aplicações Java, java e
- Tudo em Java está organizado em classes.
 - Algumas classes são desenvolvidas pelo programador da aplicação e outras,
 - Já estão disponíveis através do núcleo de funcionalidades da plataforma Java.

Núcleo de funcionalidades

- As classes que compõem o núcleo de funcionalidades Java estão organizadas em pacotes.
 - Um pacote (**package**) Java é um mecanismo para agrupar classes de finalidades afins ou de uma mesma aplicação.
 - Além de facilitar a organização conceitual das classes, o mecanismo de pacotes permite localizar cada classe necessária durante a execução da aplicação.
 - Principal funcionalidade de um pacote Java:
 - Evitar a explosão do espaço de nome, ou seja, classes com o mesmo nome em pacotes diferentes podem ser diferenciadas pelo nome completo, pacote.classe.

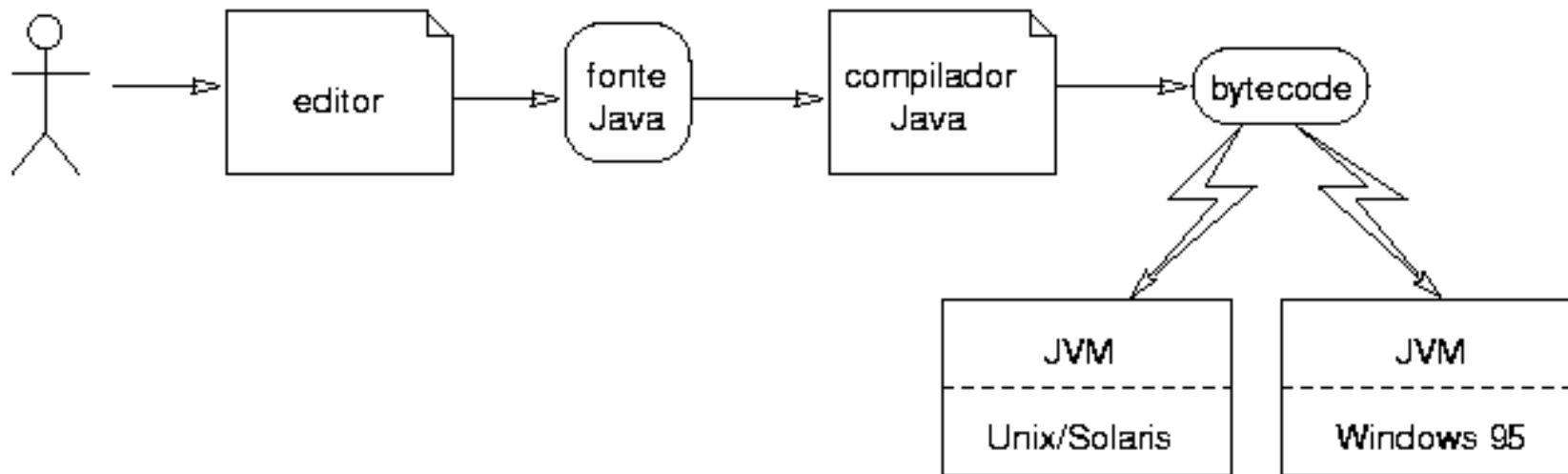
Principais Pacotes Java

- Entre os principais pacotes oferecidos como parte do núcleo Java estão:
 - `java.lang`
 - `java.util`
 - `java.io`
 - `java.net`
- Observe que esses nomes seguem a convenção Java, pela qual nomes de pacotes (assim como nomes de métodos) são grafados em letras minúsculas, enquanto nomes de classes têm a primeira letra (de cada palavra, no caso de nomes compostos) grafada com letra maiúscula.
- Além dessas funcionalidades básicas, há também APIs definidas para propósitos mais específicos compondo a extensão padronizada ao núcleo Java.

- Um dos grandes atrativos da plataforma tecnológica Java é a portabilidade do código gerado.
- Esta portabilidade é atingida através da utilização de *bytecodes*.
 - *Bytecode* é um formato de código intermediário entre o código fonte, o texto que o programador consegue manipular, e o código de máquina, que o computador consegue executar.
- Na plataforma Java, o *bytecode* é interpretado por uma máquina virtual Java (JVM).

Bytecodes

- A portabilidade do código Java é obtida à medida que máquinas virtuais Java estão disponíveis para diferentes plataformas.
- Assim, o código Java que foi compilado em uma máquina pode ser executado em qualquer máquina virtual Java, independentemente de qual seja o sistema operacional ou o processador que executa o código:



Máquina Virtual Java (JVM)

- É uma máquina de computação abstrata e um ambiente de execução independente de plataforma.
- Programas escritos em Java e que utilizem as funcionalidades definidas pelas APIs dos pacotes da plataforma Java executam nessa máquina virtual.

Algumas Palavras-chave Java

abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	false
final	finally	float	for	if
implements	import	instanceof	int	interface
long	native	new	null	package
private	protected	public	return	short
static	super	switch	synchronized	this
throw	throws	transient	true	try
void	volatile	while	enum	

- Aritméticos

- +
- -
- *
- /
- % (módulo) “resto da divisão”

- Igualdade

- ==
- !=

- Relacionais

- > (Maior que)
- < (Menor que)
- >= (Maior ou igual)
- <= (Menor ou igual)

- Lógicos

- && (E lógico)
- || (OU lógico)
- ! (NÃO lógico)

- Bitwise (bit a bit)

- & (E bit a bit)
- | (OU bit a bit)

Precedência e Associatividade de Operadores

Operadores	Associatividade	Tipo
()	Da esquerda para direita	parênteses
* / %	Da esquerda para direita	multiplicativo
+ -	Da esquerda para direita	aditivo
< <= > >=	Da esquerda para direita	relacional
= = ! =	Da esquerda para direita	igualdade
&&	Da esquerda para direita	E lógico
	Da esquerda para direita	Ou lógico
(teste)? se V: se F	Da direita para esquerda	condicional
= += -= *= /= %=	Da direita para esquerda	de atribuição

- $2 + 3 * 5 = 17$
- $(2+3) * 5 = 25$

Operadores de Incremento e Decremento

Operador	Chamado de	Expressão de exemplo	Explicação
++	Pré-incremento	++a	Incrementa a por 1, depois utiliza o novo valor de a na expressão em que a reside.
++	Pós-incremento	a++	Utiliza o valor atual de a na expressão em que a reside.
--	Pré-decremento	--b	Decrementa b por 1, depois utiliza o novo valor de b na expressão em que b reside.
--	Pós-decremento	b--	Utiliza o valor atual de b na expressão em que b reside.



Estrutura de um Programa em Java

```
1 //um primeiro programa em Java
2 //Welcome.java
3 public class Welcome {
4     // o método main inicia a execução do aplicativo Java
5     public static void main (String args [ ] )
6     {
7         System.out.println("Welcome to Java Programming!");
8     } // fim do método main
9 } // fim da classe Welcome
```



Estrutura de um Programa em Java

- *//* (comentário de única linha)
 - indica que o restante da linha é um comentário
- */** (comentário de múltiplas linhas) **/*
 - pode ser dividido em várias linhas
- */*** (comentário de documentação) **/*
 - programa utilitário **javadoc** - lê esses comentários e usa os mesmos para preparar a documentação do programa
- Documentar programas e melhorar a legibilidade
- Dica: todo programa deve inciar com um comentário indicando o propósito do programa.

Estrutura de um Programa em Java

- `public class Welcome {`
 - Inicia a definição de classe para a classe Welcome
 - `class` palavra reservada (sempre escrita em minúsculo)
 - Convenção: nome de classes iniciam com uma letra maiúscula e têm uma letra maiúscula para cada palavra do nome de classe (ExemploNomeClasse) - identificador
 - O identificador é uma série de caracteres que consistem em letras, dígitos, sublinhados (_) e sinais de cifrão (\$) que não iniciem com um dígito e não contenham nenhum espaço
 - Java faz distinção entre letras maiúsculas e minúsculas - erro de sintaxe
 - Todas as definições de classe Java são armazenadas em arquivos que terminam com a extensão nome de arquivo “.java”

Estrutura de um Programa em Java

- Dica: erro comum para uma classe public
 - Se o nome do arquivo não for idêntico ao nome da classe (ortografia e letras maiúsculas e minúsculas)
 - Um arquivo contenha duas ou mais classes public
- {
 - inicia o *corpo* de cada definição de classe
- }
 - termina cada definição de classe
- Observe que as linhas de 4 a 8 estão recuadas - convenção de espaçamento

Estrutura de um Programa em Java

- `public static void main (String args [])`
 - Faz parte de todo aplicativo Java
 - Responsável pelo início da execução dos aplicativos
 - () indicam que *main* é um método
 - Palavra-chave *void* indica que esse método realizará uma tarefa (exibindo uma linha de texto nesse programa), mas não retornará nenhuma informação
 - Dica: recue o corpo inteiro de cada definição de método um “nível” de recuo entre { e } que definem o corpo do método.

- É um método associado à classe e não a um objeto específico da classe
 - é definido como um método estático
 - deve ser um método público para permitir sua execução a partir da máquina virtual Java.
 - Não tem valor de retorno, mas recebe como argumento um arranjo de *strings* que corresponde aos parâmetros que podem ser passados para a aplicação a partir da linha de comando. Essas características determinam a assinatura do método.

- Assinatura do método main é:
 - `public static void main(String args [])` ou
 - `static public void main(String args [])`
 - O nome do parâmetro (args) poderia ser diferente, mas os demais termos da assinatura devem obedecer ao formato especificado.
 - Se a máquina virtual Java do interpretador não encontrar um método com essa assinatura para a classe especificada, uma exceção será gerada em tempo de execução: `Exception in thread "main"`
`java.lang.NoSuchMethodError: main`

- **Argumento do main:**
 - O método *main* recebe como argumento um parâmetro do tipo arranjo de objetos String.
 - Cada elemento desse arranjo corresponde a um argumento passado para o interpretador Java na linha de comando que o invocou.
 - Por exemplo, se a linha de comando é
 - `java Xyz abc 123 def`
 - o método `main (String args [])` da classe `Xyz` vai receber, nessa execução, um arranjo de três elementos na variável `args` com os seguintes conteúdos:
 - em `args[0]`, o objeto String com conteúdo "abc";
 - em `args[1]`, o objeto String com conteúdo "123";
 - em `args[2]`, o objeto String com conteúdo "def".

- Usando o SDK:
 - Para **compilar**:
 - Abrir uma janela de comando
 - Mudar para o diretório onde o **código fonte** está armazenado
 - Digitar:

```
javac NomeDoPrograma.java
```

 - **Resultado**: se o programa não contiver **erros de sintaxe**, o comando precedente criará um novo arquivo - `NomeDoPrograma.class` contendo os *bytecodes* de Java que representam nosso aplicativo
 - Para **executar**:
 - Digitar no diretório onde se encontram os arquivos `.java` e `.class`:

```
java NomeDoPrograma
```

 - Este comando dispara a execução do interpretador Java e indica que ele deve carregar o arquivo `.class`

```
// Principal.java
// Mostra o uso de argumentos
public class Principal{
    public static void main(String args[]){
        int numargs = args.length;
        String arg1 = args[0];
        String arg2 = args[1];
    }
}
```

args.length

args[0] args[1] args[2] ...

C:\>java Principal nome endereco ...

System.out (objeto de saída padrão)

- `System.out.println("Welcome to Java Programming!");`
 - Instrui o computador a realizar uma ação: imprimir a *string* de caracteres contida entre aspas duplas.
 - `System.out` é conhecido como objeto de saída padrão
 - Permite exibir *strings* e outros tipos de informações na janela de comando a partir da qual o programa Java é executado.
 - O método `System.out.println` **exibe** (ou imprime) uma linha de texto na janela de comando.
 - A linha inteira `System.out.println` , seu *argumento* entre parênteses e `;` , é uma instrução.
 - Cada instrução deve terminar com `;`

Estrutura do Programa

