Linguagem de Programação II

Introdução à Linguagem de Programação Java

Universidade do Estado do Rio de Janeiro-UERJ
Instituto de Matemática e Estatística-IME
Ciência da Computação
Professor: Alexandre Sztajnberg

Java

- Linguagem de programação orientada a objetos
- Desenvolvida pela *Sun Microsystems*, no *Green Project* iniciado em 1991
  - Patrick Naughton, Mike Sheridan, e James Gosling
  - Primeira versão estável lançada em 1996
- Compra da *Sun Microsystems* pela *Oracle* iniciada em 2009
- Em 2010, *Java* passou a ser propriedade da *Oracle*

Onde podemos encontrar?

- 97% dos Desktops Corporativos executam o Java
- 3 Bilhões de Telefones Celulares Executam o Java
  - Antes do Android, havia um ambiente Java bem próximo ao JDK
    - J2ME (https://www.java.com/pt-BR/download/help/whatis_j2me_pt-br.html)
  - Procure pelo J9 da IBM também
- 100% dos reprodutores de discos Blu-Ray vêm equipados com o Java
- 5 bilhões de placas Java em uso
- 125 milhões de aparelhos de TV executam o Java
  - Ginga J
  - TQTVD
- Consoles de games
- Supercomputadores científicos
- Aplicativos (IOS, Android, Desktop)

Características

- "Write once, run anywhere"
  - Portabilidade: ByteCode é padronizado
  - O programa fonte é compilado para ByteCode, em qualquer plataforma
  - Os arquivos gerados são interpretados por uma JVM (Java Virtual Machine), em qualquer plataforma
  - A conversão é feita para operações da plataforma onde a JVM está instalada.
- Sintaxe semelhante a C/C++.
- *Case sensitive*
- Sem uso de ponteiros *Java Virtual Machine*
  - **Mas temos referências o tempo todo** ...

ByteCode

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0123456789ABCDEF
000000  CA FE BA BE 00 00 00 32 00 2C 07 00 02 01 00 09   .......2.,......
000010  54 65 73 74 41 72 72 61 79 07 00 04 01 00 10 6A   TestArray......j
000020  61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 01   ava/lang/Object.
000030  00 06 3C 69 6E 69 74 3E 01 00 03 28 29 56 01 00   ..<init>...()V..
000040  04 43 6F 64 65 0A 00 03 00 09 0C 00 05 00 06 01   .Code...........
000050  00 0F 4C 69 6E 65 4E 75 6D 62 65 72 54 61 62 6C   ..LineNumberTabl
000060  65 01 00 12 4C 6F 63 61 6C 56 61 72 69 61 62 6C   e...LocalVariabl
000070  65 54 61 62 6C 65 01 00 04 74 68 69 73 01 00 0B   eTable...this...
000080  4C 54 65 73 74 41 72 72 61 79 3B 01 00 04 6D 61   LTestArray;...ma
000090  69 6E 01 00 16 28 5B 4C 6A 61 76 61 2F 6C 61 6E   in...([Ljava/lan
0000A0  67 2F 53 74 72 69 6E 67 3B 29 56 07 00 11 01 00   g/String;)V.....
0000B0  02 5B 49 09 00 13 00 15 07 00 14 01 00 10 6A 61   .[I...........ja
0000C0  76 61 2F 6C 61 6E 67 2F 53 79 73 74 65 6D 0C 00   va/lang/System..
0000D0  16 00 17 01 00 03 6F 75 74 01 00 15 4C 6A 61 76   ......out...Ljav
0000E0  61 2F 69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D   a/io/PrintStream
0000F0  3B 08 00 19 01 00 07 65 6C 65 6D 65 6E 74 0A 00   ;......element..
000100  1B 00 1D 07 00 1C 01 00 13 6A 61 76 61 2F 69 6F   .........java/io
000110  2F 50 72 69 6E 74 53 74 72 65 61 6D 0C 00 1E 00   /PrintStream....
000120  1F 01 00 07 70 72 69 6E 74 6C 6E 01 00 15 28 4C   ....println...(L
000130  6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67   java/lang/String
```

```
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object
Constant pool:
const #1 = Method #6.#15; // java/lang/Object."<init>":()V
const #2 = Field #16.#17; //
java/lang/System.out:Ljava/io/PrintStream;
const #3 = String #18; // Hello World
const #5 = class #21; // HelloWorld
const #7 = Asciz <init>;
const #8 = Asciz ()V;
const #9 = Asciz Code;
const #10 = Asciz LineNumberTable;
const #11 = Asciz main;
const #12 = Asciz ([Ljava/lang/String;)V;
const #13 = Asciz SourceFile;
const #14 = Asciz HelloWorld.java;
const #15 = NameAndType #7:#8;// "<init>":()V
const #16 = class #23; // java/lang/System
const #17 = NameAndType #24:#25;//
out:Ljava/io/PrintStream;
const #18 = Asciz Hello World;
const #19 = class #26; // java/io/PrintStream
```

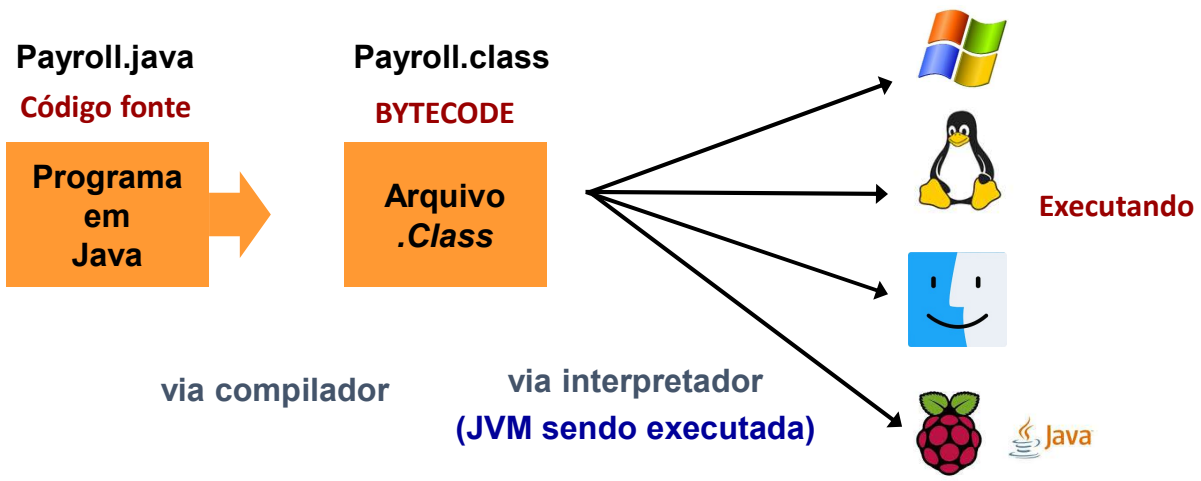
ByteCode

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0123456789ABCDEF
000000  CA FE BA BE 00 00 00 32 00 2C 07 00 02 01 00 09   .......2.,......
000010  54 65 73 74 41 72 72 61 79 07 00 04 01 00 10 6A   TestArray......j
000020  61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 01   ava/lang/Object.
000030  00 06 3C 69 6E 69 74 3E 01 00 03 28 29 56 01 00   ..<init>...()V..
000040  04 43 6F 64 65 0A 00 03 00 09 0C 00 05 00 06 01   .Code...........
000050  00 0F 4C 69 6E 65 4E 75 6D 62 65 72 54 61 62 6C   ..LineNumberTabl
000060  65 01 00 12 4C 6F 63 61 6C 56 61 72 69 61 62 6C   e...LocalVariabl
000070  65 54 61 62 6C 65 01 00 04 74 68 69 73 01 00 0B   eTable...this...
000080  4C 54 65 73 74 41 72 72 61 79 3B 01 00 04 6D 61   LTestArray;...ma
000090  69 6E 01 00 16 28 5B 4C 6A 61 76 61 2F 6C 61 6E   in...([Ljava/lan
0000A0  67 2F 53 74 72 69 6E 67 3B 29 56 07 00 11 01 00   g/String;)V.....
0000B0  02 5B 49 09 00 13 00 15 07 00 14 01 00 10 6A 61   .[I...........ja
0000C0  76 61 2F 6C 61 6E 67 2F 53 79 73 74 65 6D 0C 00   va/lang/System..
0000D0  16 00 17 01 00 03 6F 75 74 01 00 15 4C 6A 61 76   ......out...Ljav
0000E0  61 2F 69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D   a/io/PrintStream
0000F0  3B 08 00 19 01 00 07 65 6C 65 6D 65 6E 74 0A 00   ;......element..
000100  1B 00 1D 07 00 1C 01 00 13 6A 61 76 61 2F 69 6F   .........java/io
000110  2F 50 72 69 6E 74 53 74 72 65 61 6D 0C 00 1E 00   /PrintStream....
000120  1F 01 00 07 70 72 69 6E 74 6C 6E 01 00 15 28 4C   ....println...(L
000130  6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67   java/lang/String
```

```
{
public HelloWorld();
 Code: Stack=1, Locals=1, Args_size=1
  0: aload_0
  1: invokespecial #1; //Method java/lang/Object."<init>":()V
  4: return
 LineNumberTable:
  line 2: 0

public static void main(java.lang.String[]);
 Code: Stack=2, Locals=1, Args_size=1
  0: getstatic #2; //Field
java/lang/System.out:Ljava/io/PrintStream;
  3: ldc #3; //String Hello World
  5: invokevirtual #4; //Method
java/io/PrintStream.println:(Ljava/lang/String;)V
  8: return
 LineNumberTable:
  line 9: 0
  line 10: 8
}
```

Portabilidade de Java



JDK - *Java Development Kit*

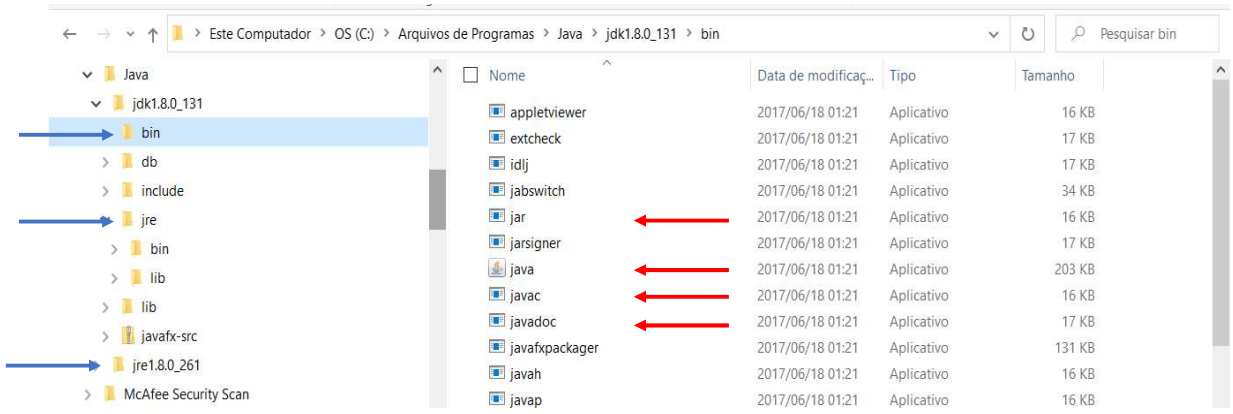
- O JDK (*Java Development Kit*) é um kit de ferramentas que permite desenvolver aplicações usando a linguagem *Java*
  - JVM, `java`
  - compilador para *Java*, `javac`
  - Gerador de documentação, `javadoc`
- Download (JDK 8)
- Documentação geral
- Documentação de classes/pacotes
- Tutorial

JRE - *Java Runtime Environment*

- O JRE (*Java Runtime Environment*) é o ambiente de execução Java
  - Se você só vai usar o Java, não desenvolver, pode ficar com o JRE
  - O JRE está incluindo no JDK
    - Como desenvolvedor, prefiro usar as ferramentas do JDK
    - Mas, na instalação, do Windows por exemplo, você instala o JDK e ele acaba ajustando o PATH para o JRE

- Configurando a variável de ambiente *PATH*, após seguir o tutorial para chegar na variável *PATH*, adicione o seguinte caminho: *C:\Program Files\Java\jdk1.8.0_231\bin* (Pode variar de máquina para máquina)

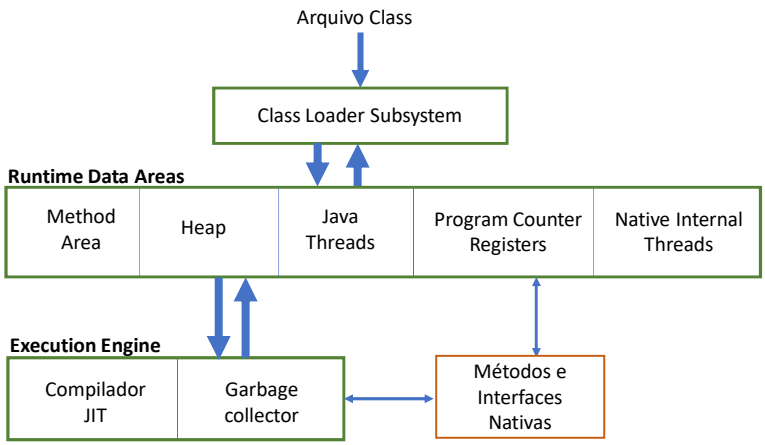
JDK "versus" JRE



JVM - *Java Virtual Machine*

- A JVM (*Java Virtual Machine)* é um programa utilitário
  - executar programas desenvolvidos em *Java*
  - interpretando e executando instruções contidas no *ByteCode*.

Mas, por que *Máquina Virtual*?



JVM - *Java Virtual Machine*

A JVM tem características e módulos de um sistema operacional e de máquina virtual

- Gerencia memória, controlando a ocupação de classes e objetos
- Controla listas de classes, referências
- Memória *heap* e memória de pilha (*stack)*
- *Garbage Collection*: periodicamente procura partes não utilizadas do programa e as desaloca. Como uma função *free()* de C, mas não pode ser acionada manualmente (na verdade, pode, mas nunca vi funcionar direito)
- Carrega classes do disco, acionando o *Classloader*
- *Realiza ligações dinamicamente*

JVM - *Java Virtual Machine*

- Após ocorrer o carregamento das classes e pacotes (bibliotecas) de apoio necessárias
  - ... A classe "principal" do programa, com entry-point (`main`) é finalmente carregada ...
- Então, um mecanismo de interpretação e execução de ByteCode é acionado.
  - ... Começando com o método `main` da classe "principal", outras classes podem ser dinamicamente carregadas, se referenciadas e se já não estiverem carregadas
- A JVM atua como intermediária entre o programa em execução, que demanda recursos como: arquivos, conexão à internet, memória, entre outros, e o sistema operacional, que os fornece.
- A JVM implementa um escalonador de threads
  - Escalonamento com prioridades fixas
  - Threads são executadas dependendo de sua prioridade em relação a outras threads

JVM - *Java Virtual Machine*

- Late *binding* (associação/ligação tardia)
  - Em linguagens como C e C++, a compilação do código fonte cria um código objeto com uma "tabela" de referências não resolvidas
  - Para se tornar um executável, o link editor resolve essas referências, substituindo-as por endereços de memória no segmento de código
  - Em Java, isso não acontece!
    - o código em ByteCode mantém todas as referências com seus próprios nomes (de variável, classe, objeto)
    - JVM somente as resolve em tempo de execução
- Carregamento dinâmico de classes
  - Não é necessário saber, em tempo de compilação, todas as classes que farão parte da execução do programa em Java.
  - Enquanto o programa é executado, a JVM verifica referências à classes não carregadas e executa a carga para a memória.
    - Problema: e se não encontar?

JVM - *Java Virtual Machine*

- Otimização adaptativa
  - Hot Spot?
  - O interpretador monitora a atividade do programa, compilando a parte do código mais usada do programa de máquina
    - Neste momento observamos um pequeno atraso nas respostas, mas depois o desempenho melhora bem
  - É muito mais rápido do que uma simples interpretação mas requer um pouco mais de memória
  - A exigência da memória é somente ligeiramente maior devido à regra de 20%/80% da execução de programa (no geral, 20% do código é responsável por 80% da execução)

Compilação Java

- A tarefa do compilador Java é traduzir o código fonte escrito em Java, salvo em um arquivo .java, para código binário intermediário, ByteCode, salvo em um arquivo .class.
- Bytecode: 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.
- O processo consiste em chamar o compilador javac no prompt de comando seguido do nome do arquivo .java
- Exemplo:
  - `javac OlaMundo.java`
- Após fazer o uso do javac, um arquivo .class com o mesmo nome do arquivo usado na compilação será criado.

Compilação Java

- Dentro do arquivo .class temos:
  - ByteCode para dados e métodos;
  - Referências simbólicas de um arquivo .class para outro:
    - Os nomes das classes em ficam em strings
    - Descompilação/Engenharia reversa mais simples
  - Nomes de métodos e suas descrições (tipos e números de argumentos);
  - Nomes de atributos e suas descrições (tipo do campo);
  - Referências simbólicas para métodos e atributos de outras classes / métodos e atributos próprios

Compilação Java

- Compilador *Just-in-time*
  - Métodos em *bytecode* são compilados em código de máquina na primeira vez que são invocados.
  - O código de máquina é guardado para uma invocação futura.
  - Esse processo requer mais memória.

Exercícios

- Baixe e instale o JDK no seu computador
- Ajuste a variável PATH
- Examine a documentação dos utilitários, e a documentação das classes
  - Veja o conteúdo, leve dúvidas para a aula