Introdução ao Java

Prof. Ítalo Assis

Ajude a melhorar este material =]

Encontrou um erro? Tem uma sugestão?

Envie e-mail para italo.assis@ufersa.edu.br

Agenda

- Um ambiente de desenvolvimento Java típico
- Conceitos iniciais de programação em Java
  - Comentários
  - Variáveis e constantes
  - Entrada e saída de dados
  - Expressões aritméticas

Java

- A **Sun Microsystems**, em 1991, financiou um **projeto de pesquisa** corporativa que resultou em uma linguagem de programação orientada a objetos chamada **Java**.
- Um objetivo-chave do Java é ser capaz de escrever programas a serem **executados em uma grande variedade de sistemas computacionais** e dispositivos controlados por computador.
- A web explodiu em popularidade em 1993 e a Sun viu o potencial de utilizar o Java para adicionar **conteúdo dinâmico**, como interatividade e animações, **às páginas da web**.
- Ele é agora utilizado para desenvolver **aplicativos corporativos** de grande porte, aprimorar a funcionalidade de **servidores da web**, e desenvolver de **aplicativos Android**
- A Sun Microsystems foi adquirida pela **Oracle** em 2010.
- Muitos programadores Java tiram proveito das ricas coleções de classes existentes e métodos nas bibliotecas de classe Java, também conhecidas como **Java APIs**

Um ambiente de desenvolvimento Java típico

- Como instalar o Java?
  - Java Downloads | Oracle
  - How to Install Java on Ubuntu 18.04 | Linuxize
- O Java é distribuído em 3 diferentes edições: **Standard Edition (SE)**, Enterprise Edition (EE) e Micro Edition (ME).
- **OpenJDK** e **Oracle Java** são as duas principais implementações de Java, com quase nenhuma diferença entre elas, exceto que Oracle Java tem alguns recursos comerciais adicionais.
- Existem dois pacotes Java diferentes, o Java Runtime Environment (JRE) e o **Java Development Kit (JDK)**.

Exemplo

1. Instalar o OpenJDK no Ubuntu
    a. Enquanto instala, seguir para os próximos slides
2. Compilar e executar um programa em Java

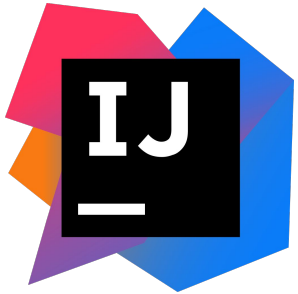
Um ambiente de desenvolvimento Java típico

- Normalmente, existem cinco passos para criar e executar um aplicativo Java: editar, compilar, carregar, verificar e executar.
- Fase 1: criando um programa
  - Você digita um programa Java (código-fonte) utilizando o editor e salva o programa
  - Arquivos de código-fonte Java recebem um nome que termina com a **extensão *.java***

Visual Studio

Netbeans

Eclipse



IntelliJ

Um ambiente de desenvolvimento Java típico

- Fase 2: compilando um programa Java em bytecodes
  - Utilize o compilador Java para compilar o programa. Por exemplo, para compilar um programa chamado *Programa.java*, você digitaria:
    - ***javac Programa.java***
  - Se o programa compilar, o compilador **produz um arquivo *.class*** (*Programa.class*)
    - IDEs tipicamente fornecem um item de menu, como *Build* ou *Make*, que chama o comando *javac* para você.
    - Se o compilador detectar erros, você precisa voltar para a Fase 1 e corrigí-los
    - **O compilador Java converte o código-fonte Java em bytecodes** que representam as tarefas a serem executadas

Um ambiente de desenvolvimento Java típico

- **A Java Virtual Machine (JVM)** - uma parte do JDK e a base da plataforma Java - **executa bytecodes**
- A máquina virtual (virtual machine - VM) é um aplicativo de software que simula um computador, mas oculta o sistema operacional e o hardware subjacentes dos programas que interagem com ela.
- Se a mesma máquina virtual é implementada em muitas plataformas de computador, os aplicativos escritos para ela podem ser utilizados em todas essas plataformas.
- A JVM é uma das máquinas virtuais mais utilizadas.

Um ambiente de desenvolvimento Java típico

- Diferentemente das instruções em linguagem de máquina, que são dependentes de plataforma, instruções bytecode são independentes de plataforma.
- Portanto, **os bytecodes do Java são portáveis** - sem recompilar o código-fonte, as mesmas instruções em bytecodes podem ser executadas em qualquer plataforma contendo uma JVM que entende a versão do Java na qual os bytecodes foram compilados.
- A JVM é invocada pelo comando *java*. Por exemplo, para executar um aplicativo Java chamado *HelloWorld*, você digitaria:
    - ***java HelloWorld***

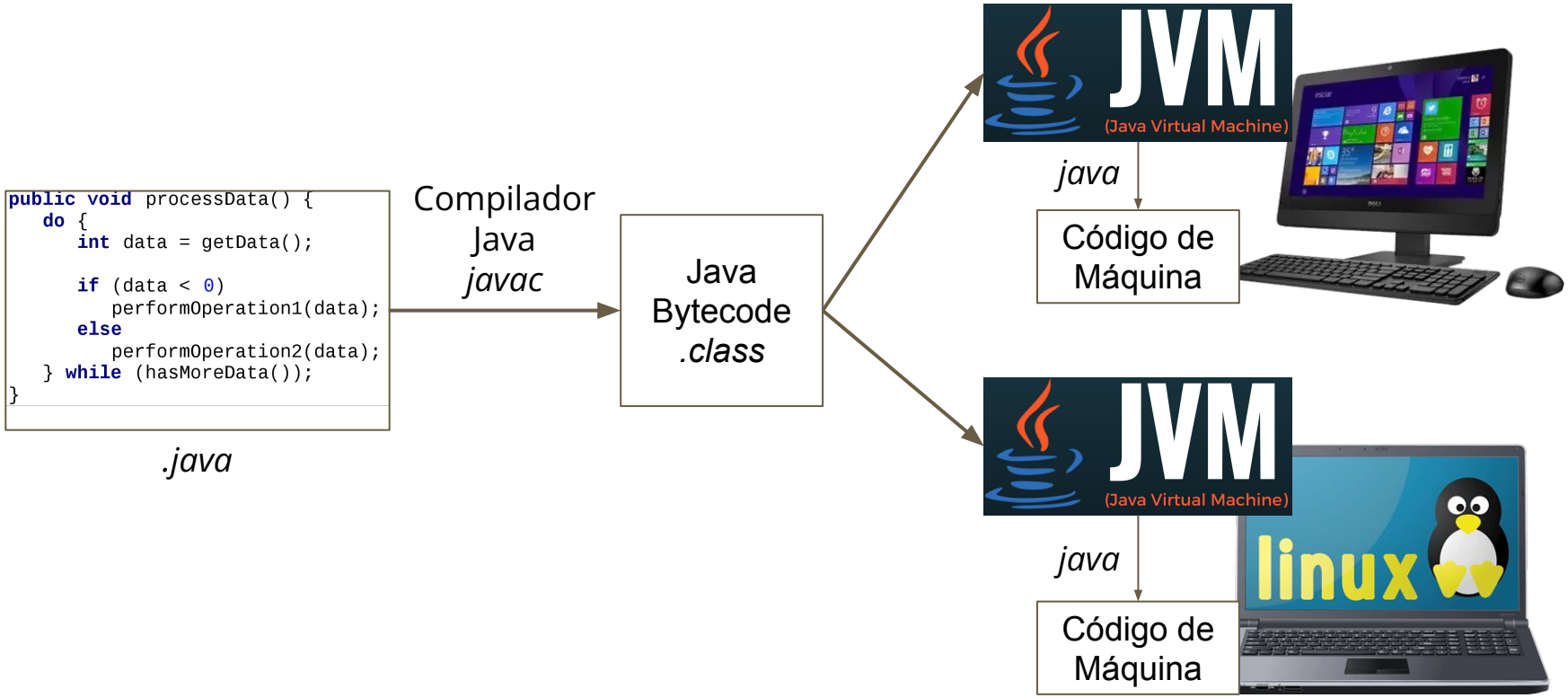
Um ambiente de desenvolvimento Java típico

- Fase 3: carregando um programa na memória
  - O carregador de classe da JVM pega os arquivos *.class* que contêm os bytecodes do programa e os transfere para a memória primária
  - Ele também carrega qualquer um dos arquivos *.class* fornecidos pelo Java que seu programa usa
- Fase 4: verificação de bytecode
  - O verificador de bytecode examina seus bytecodes para assegurar que eles são válidos e não violam restrições de segurança do Java

Um ambiente de desenvolvimento Java típico

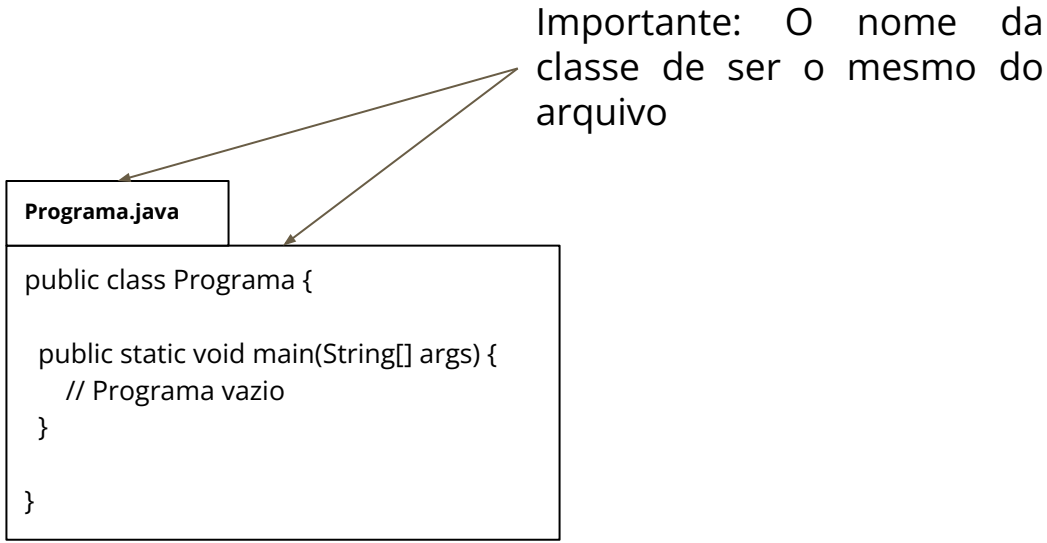
- Fase 5: execução
  - A JVM executa os bytecodes do programa, realizando, assim, as ações especificadas por ele
  - A JVM traduz os bytecodes para a linguagem de máquina do computador

Um ambiente de desenvolvimento Java típico



Um ambiente de desenvolvimento Java típico

- Programa mínimo:



Exemplo

1. Instalar o OpenJDK no Ubuntu
2. Compilar e executar um programa em Java

Comentários

- Inserimos comentários para documentar programas e aprimorar sua legibilidade
- O compilador Java ignora os comentários
- Comentários de uma linha:

```
// Este eh um comentario de uma linha
```

- Comentários de várias linhas:

```
/*
Este eh um comentario
De varias linhas
*/
```

- Comentários Javadoc
  - São delimitados por //** e */
  - Permitem incorporar a documentação diretamente aos seus programas
  - São o formato de documentação Java preferido na indústria
  - O *javadoc* lê comentários Javadoc e os usa para preparar a documentação do programa no formato HTML

Variáveis e constantes

- Variável: espaço na memória designado para o armazenamento de um determinado valor
- Constantes: Espaço na memória para o armazenamento de um valor que não pode ser alterado durante o desenvolvimento do código
- Regras para nomes de variáveis e constantes:
  - Não pode iniciar com números
  - Não pode possuir caracteres especiais
  - Não pode possuir espaços em branco
- O padrão camelCase é uma convenção utilizada para variáveis e constantes em Java
- Nomes corretos: *distanciaPercorrida*, *notaFinal*, *situacaoCadastral*....
- Nomes incorretos: *1aresposta*, *soluç@o*, *x 1*...

Variáveis e constantes

- Variáveis e constantes precisam ser declaradas
- Elas devem possuir um tipo associado
  - Os tipos do Java são divididos em primitivos e por referência
  - Entenderemos melhor os tipos por referência mais a frente no curso
  - Tipos primitivos: *boolean*, *byte*, *char*, *short*, *int*, *long*, *float* e *double*
  - As variáveis locais não são inicializadas por padrão

Variáveis e constantes

VariaveisConstantes.java

```
public class VariaveisConstantes{
   public static void main(String[] args) {
      // variaveis
      int numero = 2, n2;
      float valor1, v1 = 3.68F;
      double valor2, v2 = 3.68;
      String palavra = "Orientacao a Objetos", palavra2;
      char letra = 'w', outraLetra;
      boolean resposta1 = true, resposta2 = false, resposta3;
      // constantes
      final double ACELERACAO_GRAVIDADE = 9.78;
      final String msg = "Bem vindo(a)!\n";
      // O algoritmo continua aqui
   }
}
```

Entrada e saída de dados

- Comandos para exibir texto:
  - *System.out.print(...);*
  - *System.out.println(...);*
    - Salta uma linha no final
  - *System.out.printf(...);*
    - Exibe os dados formatados (similar ao comando em C)
- Exemplos:
  - *String nome = “UFERSA”; System.out.println(nome);*
  - *int espera = 15; System.out.printf("O tempo de espera é de %d minutos%n", espera);*

Entrada e saída de dados

- Um **Scanner** permite a um programa ler os dados
- No exemplo, o **import** permite o uso dos métodos da API Java utilizados no programa
- O objeto de entrada padrão, **System.in**, permite que aplicativos leiam bytes de informações digitadas pelo usuário
- O Scanner traduz esses bytes em tipos (como ints) que podem ser utilizados em um programa

```
import java.util.Scanner;

public class NomeDaClasse {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      String str = input.next();
      String str2 = input.nextLine();
      double numd = input.nextDouble();
      int numi = input.nextInt();
      float numf = input.nextFloat();
      System.out.printf("%s %g %d %f %n", str, numd, numi, numf);
      input.close();
   }
}
```

Exemplo

- Escreva um programa que recebe o nome de uma pessoa e deseja a ela as boas vindas mencionando seu nome

Expressões aritméticas

Operação(ões)	Operador(es)	Expressão algébrica	Expressão Java
Multiplicação	*	*bm*	b * m
Divisão	/	*x/y ou x÷y*	x / y
Resto	%	*r mod s*	r % s
Adição	+	*f* + 7	f + 7
Subtração	-	*p* – *c*	p - c

- A divisão de inteiros produz um quociente inteiro. Por exemplo, a expressão 7 / 4 é avaliada como 1.

Expressões aritméticas

Operador(es)	Operação(ões)	Precedência
*	Multiplicação	Avaliado primeiro. Se houver vários operadores desse tipo, eles são avaliados da esquerda para a direita.
/	Divisão	
%	Resto	
+	Adição	Avaliado em seguida. Se houver vários operadores desse tipo, eles são avaliados da esquerda para a direita.
-	Subtração	
=	Atribuição	Avaliado por último

Expressões aritméticas

Álgebra: $z = pr\,\%q + w/x - y$

Java:

```
z = p * r % q + w / x - y;
    6   1   2   4   3   5
```

Expressões aritméticas

- Os parênteses são utilizados para agrupar termos em expressões Java da mesma maneira como em expressões algébricas
- Por exemplo, para multiplicar *a* por *b* + *c* escrevemos *a* * *(b + c)*
- Se uma expressão contiver parênteses aninhados, como *((a + b) * c)*, a expressão no conjunto mais interno dentro dos parênteses é avaliada primeiro
- Parênteses tem precedência com relação as operações aritméticas

Exemplo

Escreva um algoritmo que, tendo como dados de entrada dois pontos quaisquer no plano, *(x1,y1)* e *(x2,y2)*, escreva a distância entre eles.

Os códigos relacionados a esta aula estão disponíveis em

https://github.com/italoaug/Programacao-Orientada-a-Objetos/tree/main/codigos/intro-java

Referências

DEITEL, Paul J. **Java: como programar**. 8.ed. São Paulo: Pearson Prentice Hall, 2010. 1144 p. ISBN: 9788576055631.