Aula 03 – Introdução à linguagem Java

Baseado nos slides oficiais do livro Java – Como programar – Deitel e Deitel – 10ª edição

Java e suas versões

Introdução

- Java é uma das linguagens de programação mais usadas em computadores para realização de tarefas
- Criada na Sun Microsystems, empresa posteriormente adquirida pela Oracle que mantém a linguagem
- Pesquisa mostrou que 97% de desktops empresariais, 89% de desktops pessoais, 3 bilhões de dispositivos e 100% dos players Blu-ray rodam java, que conta com mais de 9 milhões de desenvolvedores. (http://www.oracle.com/technetwork/articles/java/javaone12review-1863742.html)

Introdução

Java Standard Edition

- Java SE provê o necessário para desenvolver aplicações desktop e servidor

- A Java SE dá suporte a quatro paradigmas de programação: estruturado, orientado a objetos, genérico (templates) e funcional,

Introdução

Java Enterprise Edition

- Java EE engloba a Java SE

- A Java EE é apropriada para aplicações de grande porte, distribuídas e web.

Introdução

Java Micro Edition

- Java ME contém um subconjunto do Java SE e um conjunto de pacotes e ferramentas específicas

- A Java ME é apropriada para aplicações para dispositivos com sistemas embarcados com restrições de recursos
  - Smartphones
  - Smartwatches
  - Set-top-boxes (receptores de TV digital e a cabo) e muitos outros

Introdução

Android

- Sistema baseado no kernel Linux e usa Java
- Código aberto e gratuito
- Desenvolvido pela Android, Inc., acquirida pela Google em 2005.
- Usa um processo de compilação / execução semelhante ao Java ME, mas não são a mesma coisa

Java e o processo de compilação

Linguagem de máquina e assembly

Linguagem de máquina

- Código binário que executam na arquitetura alvo
  - Dependente de máquina
  -

Linguagens Assembly and Assemblers

- Assembly: linguagem de baixo nível com instruções semelhantes a abreviações em inglês das operações elementares de linguagem de máquina
- Assembler ou tradutores: convertem linguagem assembly para programas em linguagem de máquina

Compiladores e interpretadores

Compiladores

- Transformam o código em linguagem de máquina
- Podem gerar códigos intermediários (assembly, por exemplo) para códigos objetos ou bibliotecas
- Tem maior desempenho, mas limita o executável à arquitetura

Interpretadores

- Interpreta instrução a instrução e executa quando correto
- Possibilita que o código seja independente de plataforma, desde que ela tenha o interpretador
- Evita a compilação, mas torna a execução mais lenta

Java

- Criada em 1991 pela equipe de James Gosling na Sun Microsystems baseada em C++
- O objetivo chave era escrever programas que executariam em uma grande variedade de sistemas
- “Write once, run anywhere.”
- Cresceu em popularidade a partir de 1993 com a popularização da Web e os Java Applets
  - Aplicações executadas dentro do navegador

Desenvolvimento de um programa Java

- Normalmente em 5 fases
  - Editar
  - Compilar
  - Carregar
  - Verificar
  - Executar

Edição de código

- Cada classe é salva (por padrão) em um arquivo .java
- Cada pacote de classes é mapeado em um diretório
- Arquivos Java devem ser criados em editores UTF-8
  - Strings, nomes de variáveis, métodos, etc. podem conter caracteres em alfabetos não ocidentais, com acentos, til, cedilha, etc.

Compilação

- Cada classe compilada gera um arquivo .class
- Ele não é um executável, mas um assembly chamado bytecode com instruções para a arquitetura da Máquina Virtual Java (Java Virtual Machine-JVM)
- A JVM faz parte do Java Runtime Environment (JRE) e é chamada com o comando “java classe”
- O bytecode torna-se portável para diferentes arquiteturas sem necessidade de recompilação
- O compilador e outras ferramentas fazem parte do Java Development Kit (JDK)

Carga em memória

- Uma ferramenta do JRE chamada Class Loader carrega todas as classe necessárias para a execução na memória
- Os arquivos `.class` podem ser carregados do disco da máquina ou através da rede

Verificação de bytecode

- As classes carregadas em memória passam por um verificador de bytecode que examina sua validade e se não violam nenhuma restrição de segurança
- O código java executa sobre a JVM e não tem acesso direto a memória, arquivos ou dispositivos
  - Sem ponteiros
  - Menor desempenho

Execução

- A JVM executa os bytecodes interpretando instrução a instrução
  - Java é compilada e interpretada ao mesmo tempo
- A JVM possui um Just-In-Time (JIT) compiler que traduz em tempo de execução a instrução da JVM para a linguagem de máquina onde o JRE foi istalado

Revisando

- Diferença entre JavaSE, JavaME, JavaEE
- Código em UTF-8
- Compilação + interpretação
- Bytecode independente de plataforma
- Execução na JVM
- JIT compiler
- JRE e JDK

Sintaxe do Java

```
1  // Fig. 2.4: Welcome3.java
2  // Printing multiple lines of text with a single statement.
3
4  public class Welcome3
5  {
6     // main method begins execution of Java application
7     public static void main(String[] args)
8     {
9        System.out.println("Welcome\nto\nJava\nProgramming!");
10    } // end method main
11 } // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

Comentários

- Não influenciam em nada o código
- Três maneiras de comentar em java
- Comentário de linha

```
// Fig. 2.1: Welcome1.java
```

  - Ignora tudo até que o fim de linha é encontrado
- Comentário tradicional

```
/* This is a traditional comment. It
   can be split over multiple lines */
```

  - Começa com o /* e finaliza com o */ podendo ter múltiplas linhas

Comentários - documentação

- Delimitados por /** e */.
- Todo o texto é ignorado como um comentário tradicional, mas tem algumas tags específicas
- Usando o utilitário javadoc os comentários nesse estilo geram uma página html no estilo da API Java

Declarando uma classe

- Usa-se a palavra reservada class seguida do nome da classe

```
public class Welcome1
```

- Uma classe pública deve ser salva em um arquivo com o mesmo nome e gera um arquivo .class também com o mesmo nome
- Por convenção o nome de uma classe começa com letras maiúsculas assim como nomes intermediários

class SalaDeAula

  - Java é case sensitive

- O código da classe está delimitado em um bloco { }

Executando um código

Declaração do método principal

```
public static void main( String[] args )
```

- public: modificador de encapsulamento, a função pode ser acessada de fora da classe
- static: modificador que permite que a função seja usada sem a criação de um objeto
- void: o método não retorna resultado
- main: nome do método
- String args[ ]: argumentos passados pela linha de comando no mesmo estilo do argc e argv do C/C++

Entendendo o comando

- System.out
  - Objeto que representa a saída padrão (terminal)
- System.out.println
  - Mostra o conteúdo passado como argumento e pula uma linha
  - String entre aspas duplas
  - Caracteres especiais com uma contrabarra

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the *next* line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the *current* line—do *not* advance to the next line. Any characters output after the carriage return *overwrite* the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println("\"in quotes\"");` displays `"in quotes"`.

Fig. 2.5 | Some common escape sequences.

Exibindo com printf

- System.out.printf
  - `Saída formatada semelhante ao C`
- Primeiro parâmetro: uma string com texto e formatação indicada com o símbolo %
- Segue-se conjunto de parâmetros, sendo um para cada símbolo de formatação

```
1  // Fig. 2.6: Welcome4.java
2  // Displaying multiple lines with method System.out.printf.
3
4  public class Welcome4
5  {
6     // main method begins execution of Java application
7     public static void main(String[] args)
8     {
9        System.out.printf("%s%n%s%n",
10          "Welcome to", "Java Programming!");
11    } // end method main
12 } // end class Welcome4
```

```
Welcome to
Java Programming!
```

Fig. 2.6 | Displaying multiple lines with method `System.out.printf`.

```
1  // Fig. 2.7: Addition.java
2  // Addition program that inputs two numbers then displays their sum.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class Addition
6  {
7     // main method begins execution of Java application
8     public static void main(String[] args)
9     {
10       // create a Scanner to obtain input from the command window
11       Scanner input = new Scanner(System.in);
12
13       int number1; // first number to add
14       int number2; // second number to add
15       int sum; // sum of number1 and number2
16
17       System.out.print("Enter first integer: "); // prompt
18       number1 = input.nextInt(); // read first number from user
19
20       System.out.print("Enter second integer: "); // prompt
21       number2 = input.nextInt(); // read second number from user
22
```

Fig. 2.7 | Addition program that inputs two numbers then displays their sum. (Part 1 of 2.)

```
23          sum = number1 + number2; // add numbers, then store total in sum
24
25          System.out.printf("Sum is %d%n", sum); // display sum
26       } // end method main
27    } // end class Addition
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Fig. 2.7 | Addition program that inputs two numbers then displays their sum. (Part 2 of 2.)

Importando classes

- Semelhante ao include do C++, quando se quer usar uma classe já pronta
- Deve-se especificar o pacote em que ela se encontra
  - import java.util.Scanner;
  - import java.util.*;

A classe Scanner

- Declaração do objeto usado para leitura

```
Scanner input = new Scanner( System.in );
```

- Scanner: nome da classe que realiza as leituras
- input: nome do ojbeto (variável) do tipo Scanner
- = operador de atribuição
- new Scanner: operador para criação do objeto
- System.in: objeto que indica a entrada padrão do terminal
- Resumo: o objeto input é criado como instância da classe Scanner, onde a leitura feita com input virá da entrada padrão

Lendo a entrada

- A classe Scanner possui um conjunto de métodos que fazem a leitura
- Cada método retorna um tipo lido diferente
  - next() - retorna uma String
  - nextInt( ) - retorna um inteiro
  - nextDouble ( ) - retorna um double
  - nextLine( ) - retorna a linha toda

```
1  // Fig. 3.12: Dialog1.java
2  // Using JOptionPane to display multiple lines in a dialog box.
3  import javax.swing.JOptionPane;
4
5  public class Dialog1
6  {
7     public static void main(String[] args)
8     {
9        // display a dialog with a message
10       JOptionPane.showMessageDialog(null, "Welcome to Java");
11    }
12 } // end class Dialog1
```

Fig. 3.12 | Using `JOptionPane` to display multiple lines in a dialog box.

```
1  // Fig. 3.13: NameDialog.java
2  // Obtaining user input from a dialog.
3  import javax.swing.JOptionPane;
4
5  public class NameDialog
6  {
7     public static void main(String[] args)
8     {
9        // prompt user to enter name
10       String name = JOptionPane.showInputDialog("What is your name?");
11
12       // create the message
13       String message =
14          String.format("Welcome, %s, to Java Programming!", name);
15
16       // display the message to welcome the user by name
17       JOptionPane.showMessageDialog(null, message);
18    } // end main
19 } // end class NameDialog
```

Fig. 3.13 | Obtaining user input from a dialog. (Part 1 of 2.)

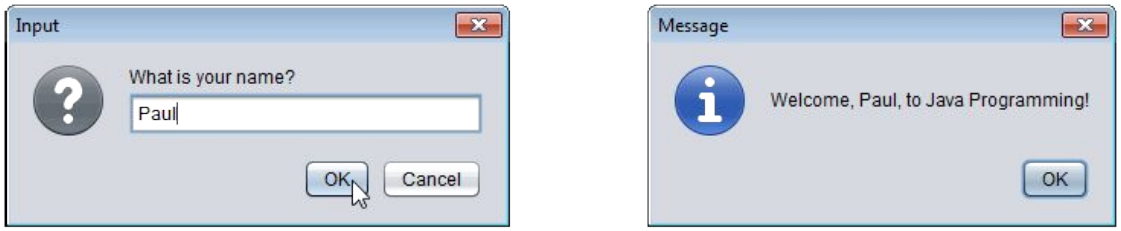

Fig. 3.13 | Obtaining user input from a dialog. (Part 2 of 2.)

Classe JOptionPane

- Caixa de diálogo gráfica
- Pertencente ao pacote javax.swing
  - import javax.swing.JOptionPane
- Métodos estáticos para leitura (showInputDialog) e exibição (showMessageDialog)
  - JOptionPane.showMessageDialog(null, “mensagem”);
  - String s = JOptionPane.showInputDialog(“digite algo”);