FE1

Ferramenta de Desenvolvimento de Aplicativos 1

Introdução à Tecnologia Java - 01

Alencar de Melo Junior / Edgar Noda

Curso Técnico em Informática – IFSP Hortolândia 2011

Original de Helder da Rocha

Tecnologia Java

- Definições a respeito do nome “Java”:
  - É uma ***Linguagem de programação*** *orientada a objetos.*
  - *Uma coleção de* ***APIs*** *(application programming interface - classes, componentes, frameworks) para o desenvolvimento de aplicações multiplataforma.*
  - *Um* ***ambiente de execução*** *presente em browsers, mainframes, SOs, celulares, palmtops, cartões inteligentes, eletrodomésticos.*

2

Tecnologia Java

- Definições a respeito do nome “Java”:
  - Java foi lançada pela Sun Microsystems em 1995.
  - Java é um padrão controlado através da JCP Java Community Process (www.jcp.org).
  - De 2006 a 2007, a maior parte do Java passou para os termos da GNU General Public License (GPL).
  - Ambientes de execução e desenvolvimento são fornecidos por fabricantes de hardware e software (MacOS, Linux, etc.) Encontra-se na versão 7 (Java SE 7).

3

Linguagem Java

- *Principais características:*
  - *Orientação a objetos*
  - *Portabilidade - Independência de plataforma - "escreva uma vez, execute em qualquer lugar" ("write once, run anywhere");*
  - *Recursos de Rede - Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP;*
  - *Segurança - Pode executar programas via rede com restrições de execução (vários mecanismos para controlar segurança) ;*
  - *Familiar (sintaxe parecida com C).*

4

Linguagem Java

- *Principais características:*
  - *Simples e robusta (minimiza bugs, aumenta produtividade).*
  - *Suporte nativo a threads (+ simples, maior portabilidade), facilidades para criação de programas distribuídos e multitarefa.*
  - *Dinâmica - Programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.*
  - *Desalocação de memória automática por processo de coletor de lixo;*
  - *Facilidades de Internacionalização - suporta nativamente caracteres Unicode.*

5

Blibliotecas - APIs

- Programas Java consistem em partes chamadas classes.
  - As classes incluem os métodos que realizam as tarefas (os programadores implementaram).
  - Vantagem dessa abordagem: *Java possui uma coleção de APIs (Aplicattion programming interfaces -* bibliotecas*) padrão que podem ser usadas para construir aplicações .*
  - Organizadas em pacotes (java.*, javax.* e extensões).
  - Usadas pelos ambientes de execução (JRE) e de desenvolvimento (JDK).

6

Bliblioteca s - APIs

- *As principais APIs são distribuídas juntamente com os produtos para desenvolvimento de aplicações*
  - ***Java Standard Edition (Java SE)**: ferramentas e APIs essenciais para qualquer aplicação Java (inclusive GUI)*
  - ***Java Enterprise Edition (Java EE)**: ferramentas e APIs para o desenvolvimento de aplicações distribuídas*
  - ***Java Micro Edition (Java ME)**: ferramentas e APIs para o desenvolvimento de aplicações para aparelhos portáteis*

7

Ambiente de execução e desenvolvimento

- ***Java Development Kit** (JDK)*
  - *Coleção de ferramentas de linha de comando para, entre outras tarefas, compilar, executar e depurar aplicações Java*
  - *Para habilitar o ambiente via linha de comando é preciso colocar o caminho `$JAVA_HOME/bin` no PATH do sistema*
- ***Java Runtime Environment** (JRE)*
  - *Tudo o que é necessário para executar aplicações Java*
- *Variável JAVA_HOME (opcional: usada por vários frameworks)*
  - *Defina com o local de instalação do Java no seu sistema. Exemplos:*
  - *Windows:* `set JAVA_HOME=c:\jdk1.7.0`
  - *Linux:*

```
JAVA_HOME=/usr/java/jdk1.7.0
export JAVA_HOME
```

8

Compilação para bytecode

- ***Bytecode*** *é o código de máquina que roda em qualquer máquina através da* ***Máquina Virtual Java*** *(JVM)*
  - *Texto contendo código escrito em linguagem Java é traduzido em bytecode através do processo de* ***compilação*** *e armazenado em um arquivo* `*.class` *chamado de* ***Classe Java***

Código Java (texto)

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello, world!");
    }
}
```

HelloWorld.java

compilação (`javac`)

HelloWorld.class

```
F4 D9 00 03 0A B2 FE FF FF 09 02 01 01 2E 2F 30 62 84 3D 29 3A C1
```

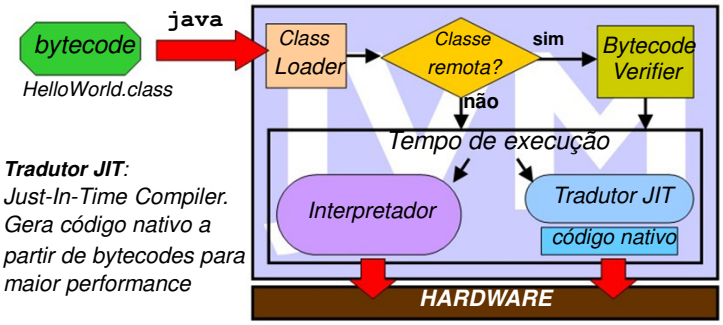
Bytecode Java (código de máquina virtual)

Uma "classe" Java

9

Máquina Virtual Java (JVM)

- *"Máquina imaginária implementada como uma aplicação de software em uma máquina real"* [JVMS]
- *A forma de execução de uma aplicação depende ...*
  - *... da origem do código a ser executado (remoto ou local)*
  - *... da forma como foi implementada a JVM pelo fabricante (usando tecnologia JIT, HotSpot, etc.)*



Tradutor JIT*: Just-In-Time Compiler. Gera código nativo a partir de bytecodes para maior performance*

10

Class Loader e CLASSPATH

- Primeira tarefa executada pela JVM: carregamento das classes necessárias para rodar a aplicação. O **Class Loader**
  1. Carrega primeiro as classes nativas do **JRE** (APIs)
  2. Depois carrega extensões do **JRE**: JARs em `$JAVA_HOME/jre/lib/ext` e classes em `$JAVA_HOME/jre/lib/classes`
  3. Carrega classes do sistema local (a ordem dos caminhos no **CLASSPATH** define a precedência)
  4. Por último, carrega possíveis classes remotas
- **CLASSPATH**: variável de ambiente local que contém todos os caminhos locais onde o Class Loader pode localizar classes
  - A CLASSPATH é lida depois, logo, suas classes nunca substituem as classes do JRE (não é possível tirar classes JRE do CLASSPATH)
  - Classes remotas são mantidas em área sujeita à verificação
  - CLASSPATH pode ser redefinida através de parâmetros durante a execução do comando `java`

11

Bytecode Verifier

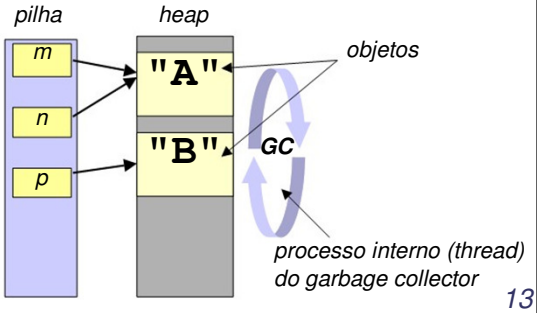
- Etapa que antecede a execução do código em classes carregadas através da rede
  - Class Loader distingue classes locais (seguras) de classes remotas (potencialmente inseguras)
- Verificação garante
  - Aderência ao formato de arquivo especificado [JVMS]
  - Não-violação de políticas de acesso estabelecidas pela aplicação
  - Não-violação da integridade do sistema
  - Ausência de estouros de pilha
  - Tipos de parâmetros corretamente especificados e ausência de conversões ilegais de tipos

12

Coleta de lixo

- Memória alocada em Java não é liberada pelo programador
  - Ou seja, objetos criados não são destruídos pelo programador
- A criação de objetos em Java consiste de
  1. Alocar memória no heap para armazenar os dados do objeto
  2. Inicializar o objeto (via construtor)
  3. Atribuir endereço de memória a uma variável (referência)
- Mais de uma referência pode apontar para o mesmo objeto

```
Mensagem m, n, p;
m = new Mensagem("A");
n = m;
p = new Mensagem("B");
```

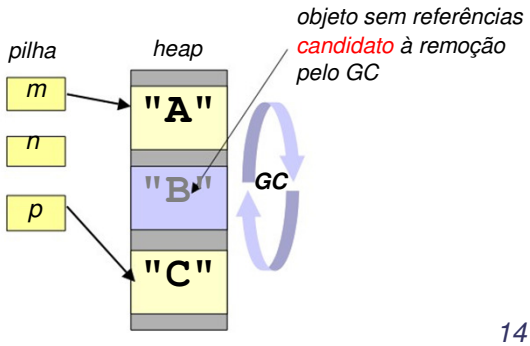


13

Coleta de lixo

- Quando um objeto não tem mais referências apontando para ele, seus dados não mais podem ser usados, e a memória deve ser liberada.
- O coletor de lixo irá liberar a memória na primeira oportunidade

```
n = null;
p = new Mensagem("C");
```



14

O JDK

- O JDK (**J**ava **D**evelopment **K**it) é o ambiente padrão distribuído pela Sun para desenvolvimento de aplicações Java.
- O JDK consiste de
  - JRE (Java Runtime Environment) - também distribuído separadamente: ambiente para execução de aplicações
  - Ferramentas para desenvolvimento: compilador, debugger, gerador de documentação, empacotador JAR, etc.
  - Código-fonte das classes da API
  - Demonstrações de uso das APIs, principalmente Applets, interface gráfica com Swing e recursos de multimídia
- A documentação é distribuída separadamente

15

Aplicação HelloWorld

- Este código em Java imprime um texto na tela quando executada via linha de comando

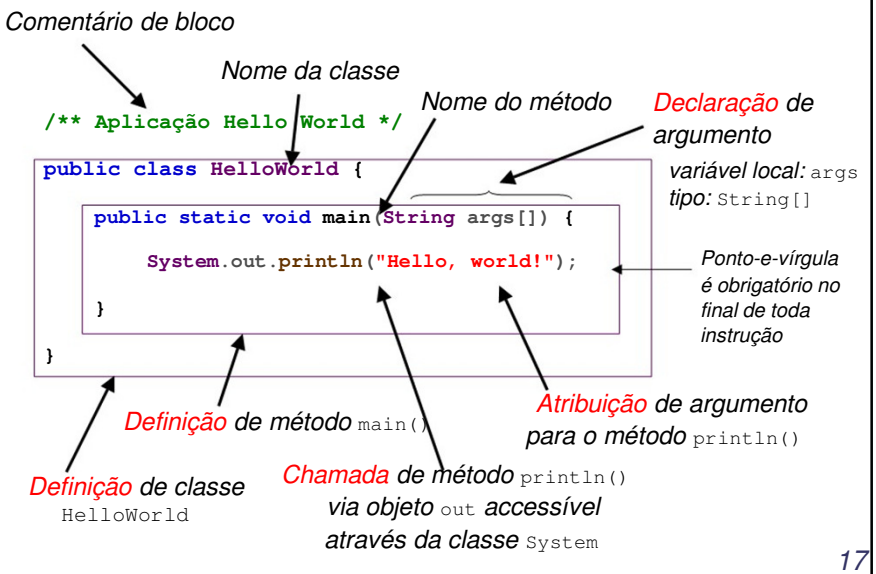
```
/** Aplicação Hello World */
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello, world!");
        }
}
```

HelloWorld.java

- No diretório onde o arquivo estiver gravado:
  - javac HelloWorld.java (compilando)
  - java Helloworld (para executar).

16

Anatomia



17

Uma classe que define um tipo de dados

- *Esta classe representa objetos que guardam um texto (tipo String) em um atributo (**msg**) publicamente acessível.*
- *Além de guardar um String, retorna o texto em caixa-alta através do método **lerNome()**.*

Definição da classe (tipo) Mensagem em Java

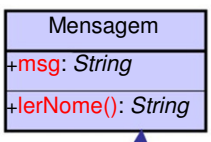
```
public class Mensagem {
    public String msg = "";
    public String lerNome() {
        String nomeEmMaiusculas =
                    msg.toUpperCase();
        return nomeEmMaiusculas;
    }
}
```

atributo

método

Membros da classe. Outras classes podem acessá-los, se declarados como "public", usando o operador ponto "."

Representação em UML



*Esta é a **interface pública** da classe. É só isto que interessa a quem vai usá-la. Os detalhes (código) estão **encapsulados**.*

18

Classe executável que usa um tipo

- *Esta outra classe **usa** a classe anterior para criar um objeto e acessar seus membros visíveis por sua interface pública*
  - *Pode alterar ou ler o valor do atributo de dados* `msg`
  - *Pode chamar o método* `lerNome()` *e usar o valor retornado*

```
public class HelloJava {
  private static Mensagem nome;

  public static void main(String args[]) {
    nome = new Mensagem();    // cria objeto

    if (args.length > 0) {    // há args de linha de comando?
       nome.msg = args[0];    // se houver, copie para msg
    } else {
       nome.msg = "Usuario"; // copie palavra "Usuario"
    }

    String texto = nome.lerNome(); // chama lerNome()
    System.out.println("Bem-vindo ao mundo Java, "+texto+"!");
  }
}
```

atributo `nome` *é do tipo Mensagem*

Este método é chamado pelo interpretador

Operador de concatenação

19

Detalhes

- *Declaração do método main()*

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

modificadores

tipo de dados retornado

tipo de dados aceito como argumento

nome

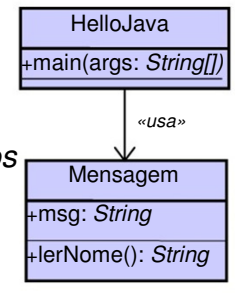
variável local ao método que contém valor passado na chamada

- *O método main() é chamado pelo interpretador Java, automaticamente*
  - *Deve ter sempre a assinatura acima*
- *O argumento é um vetor formado por textos passados na linha de comando:*

```
> java NomeDaClasse Um "Dois Tres" Quatro
```

args[0] *args[1]* *args[2]*

Dependência entre as duas classes (HelloJava tem referência para Mensagem)



20

Referências Bibliográficas

Rocha, H. da. Curso J100: Java 2 Standart Edition. Revisão 17.0. www.argonavis.com.br, 2003.

DEITEL, H. M.; DEITEL, P. J. **Java: Como Programar**. 6ª edição, Pearson Prentice Hall,2005. São Paulo.

21