Introdução à tecnologia Java

Helder da Rocha
www.argonavis.com.br

Assuntos abordados neste módulo

- Conceitos
  - Tecnologia Java
  - Linguagem e API Java
  - Máquina virtual Java
  - Ambiente de execução (JRE) e desenvolvimento (SDK)
  - Carregador de classes (ClassLoader) e CLASSPATH
  - Verificador de bytecodes
  - Coletor de lixo (garbage collector)
- Introdução prática
  - Como escrever uma aplicação Java
  - Como compilar uma aplicação Java
  - Como executar uma aplicação Java
  - Como depurar erros de compilação e execução

Parte 1: Tecnologia Java

- O nome "Java" é usado para referir-se a
  - Uma **linguagem de programação** orientada a objetos
  - Uma coleção de **APIs** (classes, componentes, frameworks) para o desenvolvimento de aplicações multiplataforma
  - Um **ambiente de execução** presente em browsers, mainframes, SOs, celulares, palmtops, cartões inteligentes, eletrodomésticos
- Java foi lançada pela Sun em 1995. Três grandes revisões
  - Java Development Kit (JDK) 1.0/1.0.2
  - Java Development Kit (JDK) 1.1/1.1.8
  - Java 2 Platform (Java 2 SDK e JRE 1.2, 1.3, 1.4)
- A evolução da linguagem é controlada pelo **Java Community Process** (www.jcp.org) formado pela Sun e usuários Java
- Ambientes de execução e desenvolvimento são fornecidos por fabricantes de hardware e software (MacOS, Linux, etc.)

Linguagem Java

- Linguagem de programação orientada a objetos
  - Familiar (sintaxe parecida com C)
  - Simples e robusta (minimiza bugs, aumenta produtividade)
  - Suporte nativo a threads (+ simples, maior portabilidade)
  - Dinâmica (módulos, acoplamento em tempo de execução)
  - Com coleta de lixo (menos bugs, mais produtividade)
  - Independente de plataforma
  - Segura (vários mecanismos para controlar segurança)
  - Código intermediário de máquina virtual interpretado (compilação rápida - + produtividade no desenvolvimento)
  - Sintaxe uniforme, rigorosa quanto a tipos (código mais simples, menos diferenças em funcionalidades iguais)

Produtos e APIs

- *Java possui uma coleção de APIs (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 (**SDK**)*
- *As principais APIs são distribuídas juntamente com os produtos para desenvolvimento de aplicações*
  - ***Java 2 Standard Edition (J2SE)**: ferramentas e APIs essenciais para qualquer aplicação Java (inclusive GUI)*
  - ***Java 2 Enterprise Edition (J2EE)**: ferramentas e APIs para o desenvolvimento de aplicações distribuídas*
  - ***Java 2 Micro Edition (J2ME)**: ferramentas e APIs para o desenvolvimento de aplicações para aparelhos portáteis*

Ambiente de execução e desenvolvimento

- ***Java 2 System Development Kit*** *(J2SDK)*
  - *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*
  - *Parte do J2SDK e das principais distribuições Linux, MacOS X, AIX, Solaris, Windows 98/ME/2000 (exceto XP)*
- *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:\j2sdk1.4.0`
  - *Linux:*

```
JAVA_HOME=/usr/java/j2sdk1.4.0
export JAVA_HOME
```

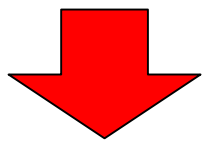
Compilação para bytecode

- **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

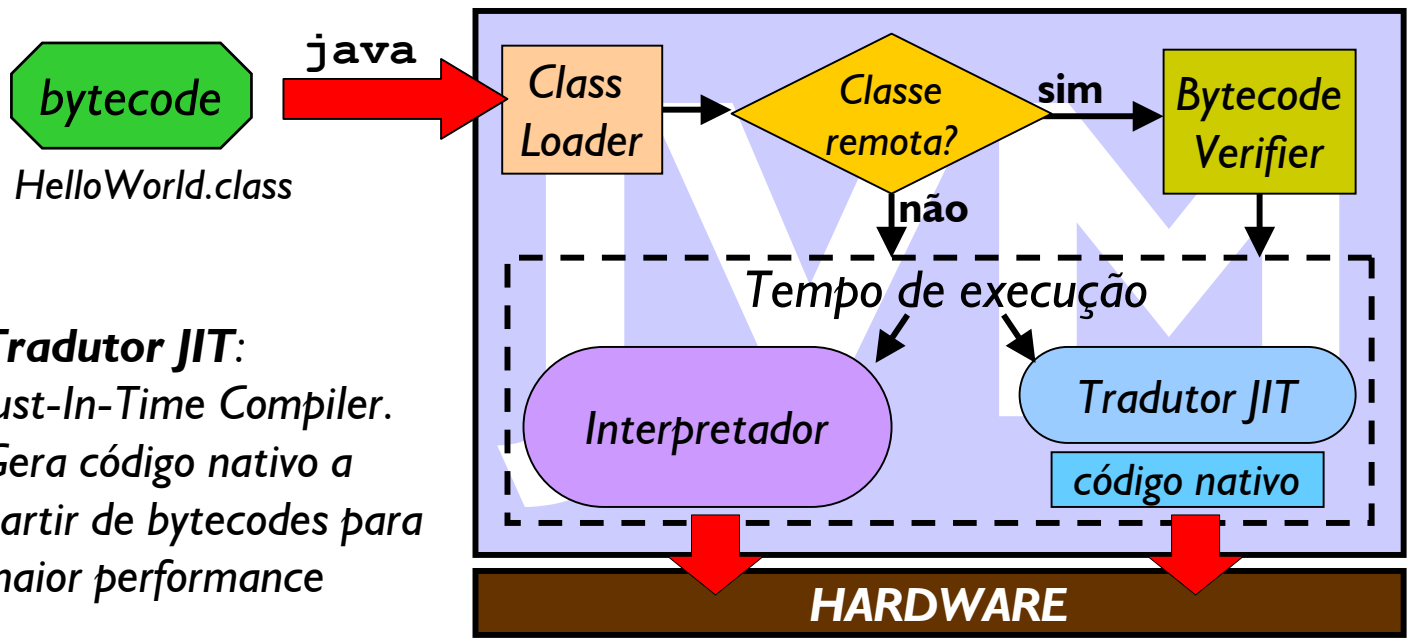
```
F4 D9 00 03 0A B2 FE FF FF 09 02 01 01 2E 2F 30 62 84 3D 29 3A C1
```

Bytecode Java (código de máquina virtual)

Uma "classe" Java

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

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`

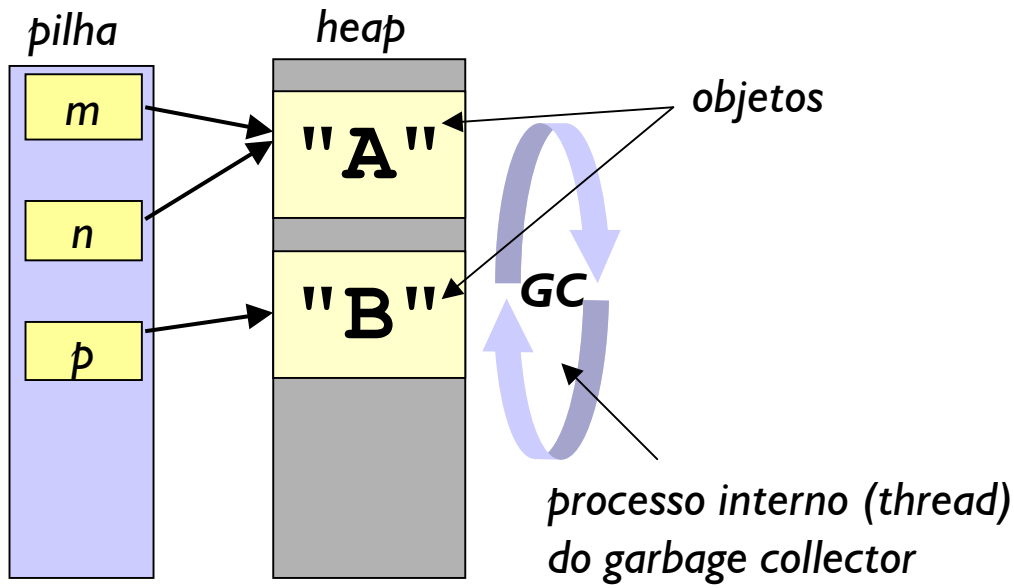
Bytecode Verifier

- *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*

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

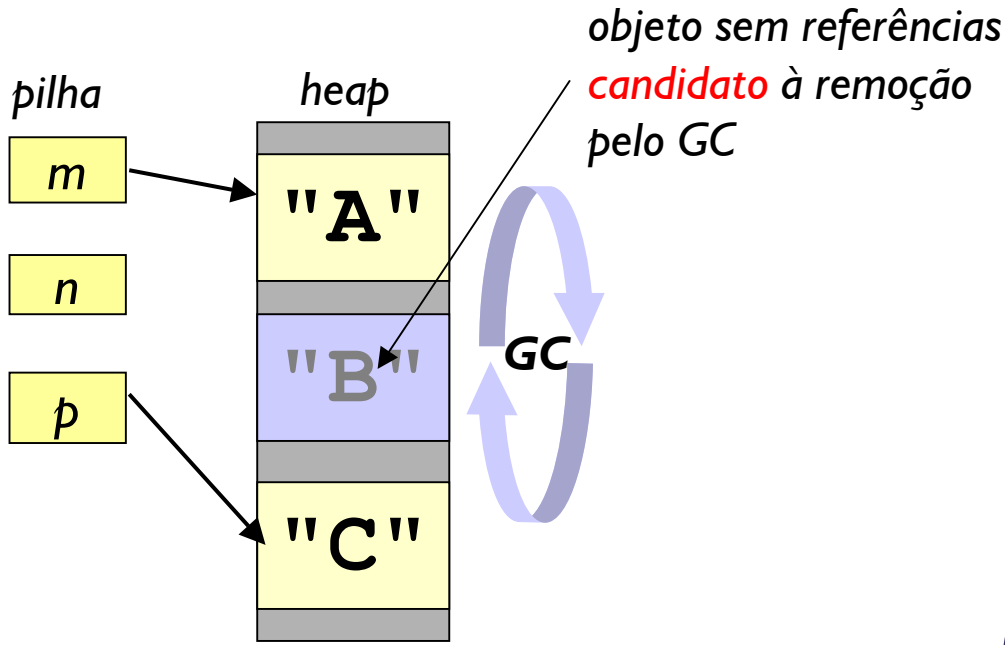
```
Mensagem m, n, p;

m = new Mensagem("A");

n = m;

p = new Mensagem("B");
```



Coleta de lixo (2)

- *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");
```



O que Java não faz

- Java não suporta herança múltipla de implementação
  - Herança múltipla é característica comum a várias linguagens OO, e permite reuso de código de várias classes em outra classe
  - Tem vantagens porém aumenta a complexidade
  - Java oferece uma solução que preserva as principais vantagens da herança múltipla e evita os problemas
- Java não suporta aritmética de ponteiros
  - Ponteiros, ou referências, são usados em várias linguagens, inclusive Java, para manipular eficientemente grandes quantidades de informação na memória
  - Com ponteiros, em vez de copiar uma informação de um lugar para outro, copia-se apenas o seu endereço
  - Em linguagens como C, o programador pode manipular o endereço (que é dependente de plataforma) diretamente
  - Isto aumenta a complexidade e diminui a portabilidade

O J2SDK

- O J2SDK (**J**ava **2** **S**ystem **D**evelopment **K**it) é o ambiente padrão distribuído pela Sun para desenvolvimento de aplicações Java
- O J2SDK 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

Como compilar

- *Use o **java c**ompiler (linha de comando)*
  - `javac NomeDaClasse.java`
  - `javac -d ../destino Um.java Dois.java`
  - `javac -d ../destino *.java`
  - `javac -classpath c:\fontes -d ../destino *.java`
- *Algumas opções (opcionais)*
  - `-d` *diretório onde serão armazenadas as classes (arquivos* `.class`*) geradas*
  - `-classpath` *diretórios (separados por* `;` *ou* `:`*) onde estão as classes requeridas pela aplicação*
  - `-sourcepath` *diretórios onde estão as fontes*
- *Para conhecer outras opções do compilador, digite* `javac` *sem argumentos*
- *Compiladores de outros fabricantes (como o **Jikes**, da IBM) também podem ser usados no lugar do javac*

Como executar

- *Use o interpretador **java** (faz parte do JRE)**
  - `java NomeDaClasse`
  - `java pacote.subpacote.NomeDaClasse`
  - `java -classpath c:\classes;c:\bin;. pacote.Classe`
  - `java -cp c:\classes;c:\bin;. pacote.Classe`
  - `java -cp %CLASSPATH%;c:\mais pacote.Classe`
  - `java -cp biblioteca.jar pacote.Classe`
  - `java -jar executavel.jar`
- *Para rodar aplicações gráficas, use **javaw***
  - `javaw -jar executavel.jar`
  - `javaw -cp aplicacao.jar;gui.jar principal.Inicio`
- *Principais opções*
  - `-cp` *ou*
    `-classpath` *classpath novo (sobrepõe v. ambiente)*
  - `-jar` *executa aplicação executável guardada em JAR*
  - `-D`***`propriedade`***`=`***`valor`*** *define propriedade do sistema (JVM)*

* *sintaxe de PATH em Unix é diferente*

Algumas outras ferramentas do SDK

- Debugger: **jdb**
  - Depurador simples de linha de comando
- Profiler: **java -prof**
  - Opção do interpretador Java que gera estatísticas sobre uso de métodos em um arquivo de texto chamado java.prof
- Java Documentation Generator: **javadoc**
  - Gera documentação em HTML (default) a partir de código-fonte Java
- Java Archiver: **jar**
  - Extensão do formato ZIP; ferramenta comprime, lista e expande
- Applet Viewer: **appletviewer**
  - Permite a visualização de applets sem browser
- HTML Converter: **htmlconverter.jar**
  - Converte <applet> em <object> em páginas que usam applets
- Disassembler: **javap**
  - Permite ler a interface pública de classes

Parte 2: Introdução Prática

Parte 2: Introdução prática

- Nesta seção serão apresentados alguns exemplos de aplicações simples em Java
  1. Aplicação HelloWorld
  2. Aplicação HelloWorld modificada para promover reuso e design orientado a objetos (duas classes)
  3. Aplicação Gráfica Swing (três classes)
  4. Aplicação para cliente Web (applet)
- Compile código-fonte no CD
  - cap01/src/
- Todos os assuntos apresentados nesta seção serão explorados em detalhes em aulas posteriores
  - Conceitos como classe, objeto, pacote
  - Representação UML
  - Sintaxe, classes da API, etc.

1. Aplicação HelloWorld

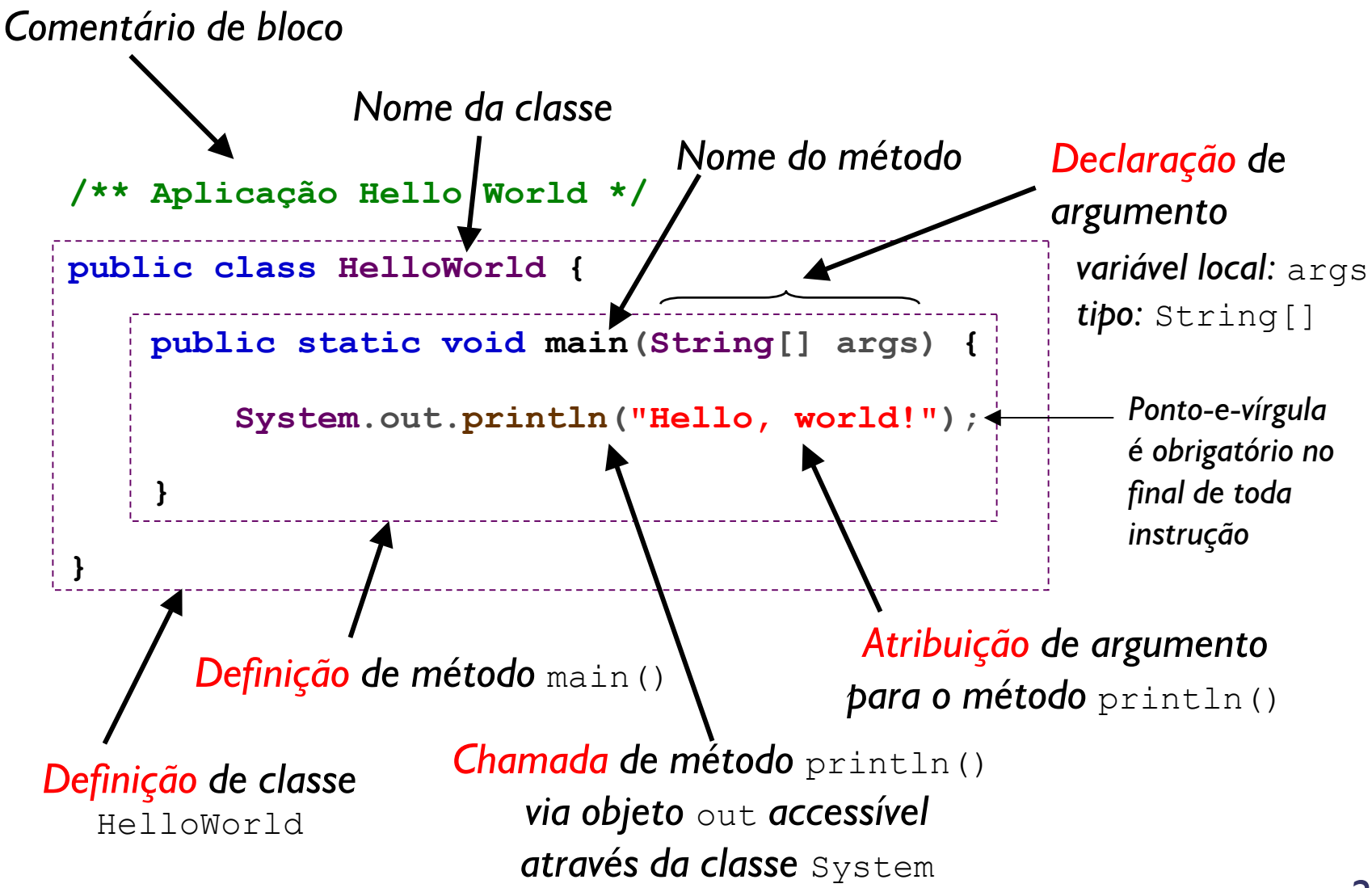
- *Esta mini-aplicação em Java imprime um texto na tela quando executada via linha de comando*

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

HelloWorld.java

- *Exercício: Use-a para testar seu ambiente e familiarizar-se com o desenvolvimento Java*
  - *Digite-a no seu editor de textos*
  - *Tente compilá-la*
  - *Corrija eventuais erros*
  - *Execute a aplicação*

Anatomia


Comentário de bloco
Nome da classe
Nome do método
Declaração de argumento
variável local: args
tipo: String[]
/** Aplicação Hello World */
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello, world!");
}
}
Ponto-e-vírgula é obrigatório no final de toda instrução
Definição de método main()
Atribuição de argumento para o método println()
Definição de classe HelloWorld
Chamada de método println() via objeto out accessível através da classe System

2. Uma classe 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

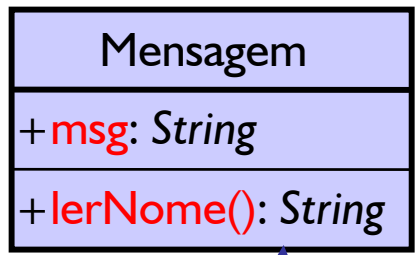
```
public class Mensagem {

    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**.*

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*

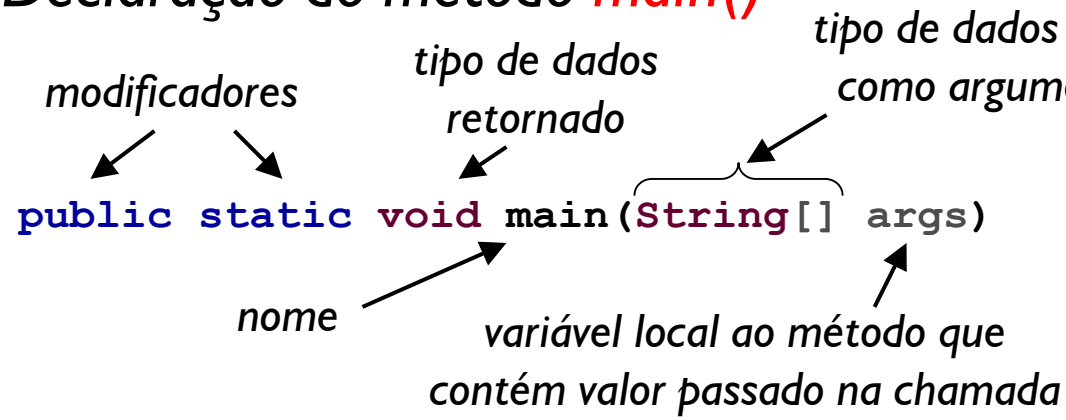
Este método é chamado pelo interpretador

Operador de concatenação

Veja demonstração

Detalhes

- *Declaração do método main()*

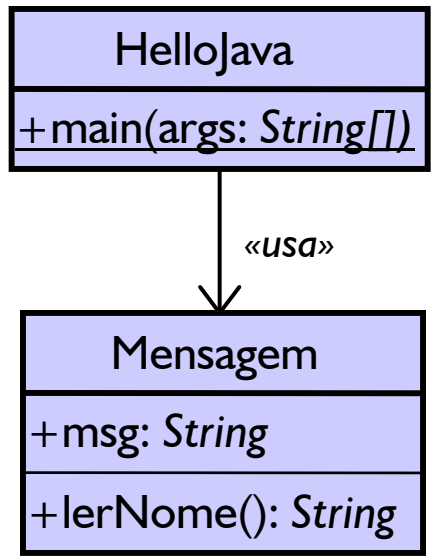


- *O método main() é chamado pelo interpretador Java, automaticamente*
  - *Deve ter sempre a assinatura acima*
- *O argumento é um vetor formado por textos passados na linha de comando:*

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

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

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



3. Primeira aplicação gráfica

- *A aplicação abaixo cria um objeto do tipo JFrame (da API Swing) e reutiliza a classe Mensagem*

```
import javax.swing.*; // importa JFrame e JLabel
import java.awt.Container;

public class MensagemGUI {
  public MensagemGUI(String texto) {
    JFrame janela = new JFrame("Janela");
    Container areaUtil = janela.getContentPane();
    areaUtil.add( new JLabel(texto) );
    janela.pack();
    janela.setVisible(true);
  }
}
```

Quando objeto é criado, construtor MensagemGUI é chamado.

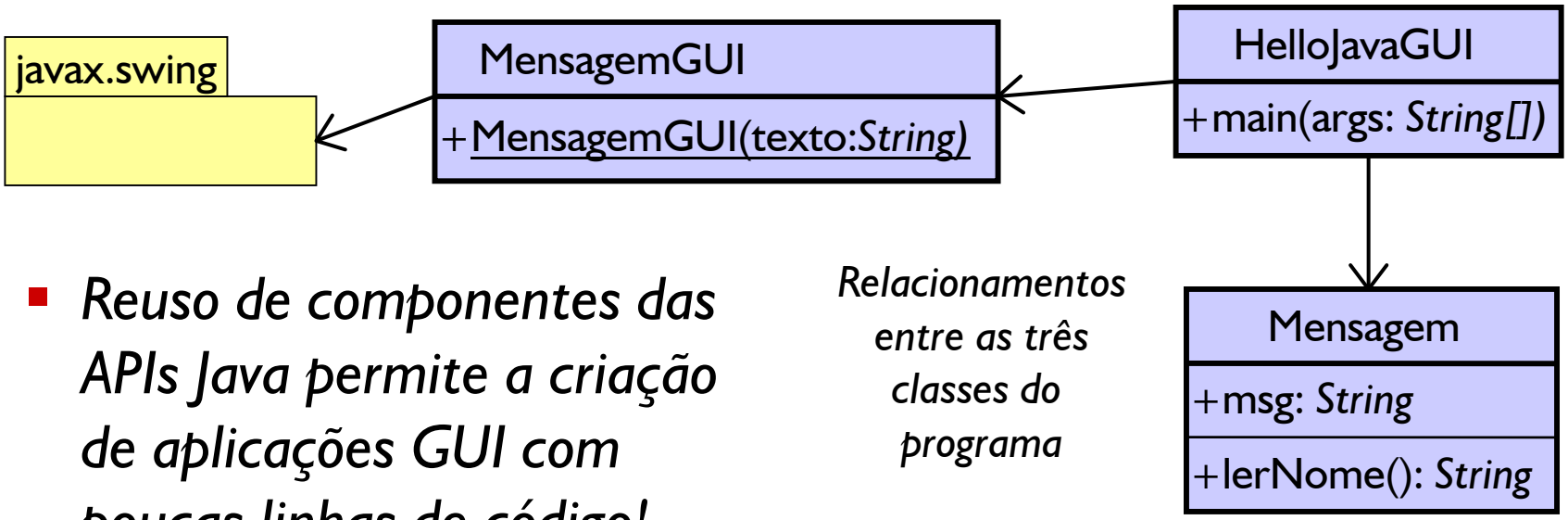
Construtor cria janela contendo texto recebido

```
public class HelloJavaGUI {
  private static Mensagem nome;
  public static void main(String[] args) {
    (... igual a HelloJava ...)
    String texto = nome.lerNome();
    new MensagemGUI
        ("Bem-vindo ao mundo Java, "+texto+"!");
  }
}
```

reuso!

No lugar de imprimir o texto, passa-o como parâmetro na criação de MensagemGUI

Componentes da aplicação gráfica



- *Reuso de componentes das APIs Java permite a criação de aplicações GUI com poucas linhas de código!*

Relacionamentos entre as três classes do programa

Execução da apliação passando parâmetro via linha de comando

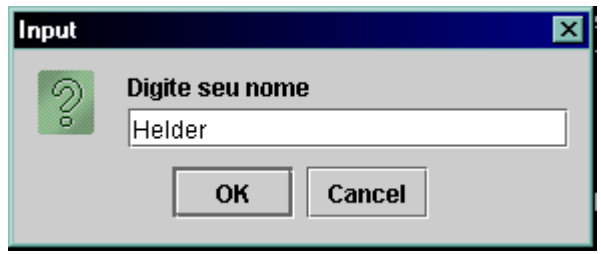
Entrada de dados via GUI

- *javax.swing.JOptionPane oferece uma interface gráfica para entrada de dados e exibição de informações*
- *Exemplo de exibição de caixa de diálogo*
  - `JOptionPane.showMessageDialog(null, "Hello, World!");`

É preciso importar javax.swing. ou javax.swing.JOptionPane*

- *Exemplo de diálogo para entrada de dados*

```
String nome =
    JOptionPane.showInputDialog("Digite seu nome");
if (nome != null) {
    JOptionPane.showMessageDialog(null, nome);
} else {
    System.exit(0);
}
```



4. Primeiro applet

- *Componentes gráficos que podem ser executados no browser*
- *Para criar e usar um applet é preciso*
  - *criar uma classe que herde da classe Applet ou JApplet (API Java)*
  - *criar uma página HTML que carregue o applet*
- *A classe abaixo implementa um JApplet*

```
import javax.swing.*; // importa JFrame e JLabel
import java.awt.Container;

public class HelloJavaApplet extends JApplet {
  private Mensagem nome;

  public void init() {
    nome = new Mensagem();
    nome.msg = this.getParameter("texto"); // parâmetro HTML
    String texto = nome.lerNome();
    Container areaUtil = this.getContentPane();
    JLabel label =
       new JLabel("Bem-vindo ao mundo Java, " +texto+ "!");
    areaUtil.add(label);
  }
}
```

Herança!

Chamado automaticamente pelo browser

Página HTML

- O elemento <applet> é usado para incluir applets antigos (Java 1.0 e 1.1) em páginas HTML ou servir de template para a geração de código HTML 4.0
- A seguinte página carrega o applet da página anterior

```
<html>
 <head>
  <title>Sem Título</title>
 </head>
 <body>
  <h1>Um Applet</h1>
  <applet code="HelloJavaApplet.class" height="50" width="400">
    <param name="texto" value="Helder">
  </applet>
 </body>
</html>
```

- Converta o código para HTML 4.0: ferramenta htmlconverter
  - Guarde uma cópia do original, e rode (use htmlconverter.bat)

```
> htmlconverter pagina.html
```

Detalhes

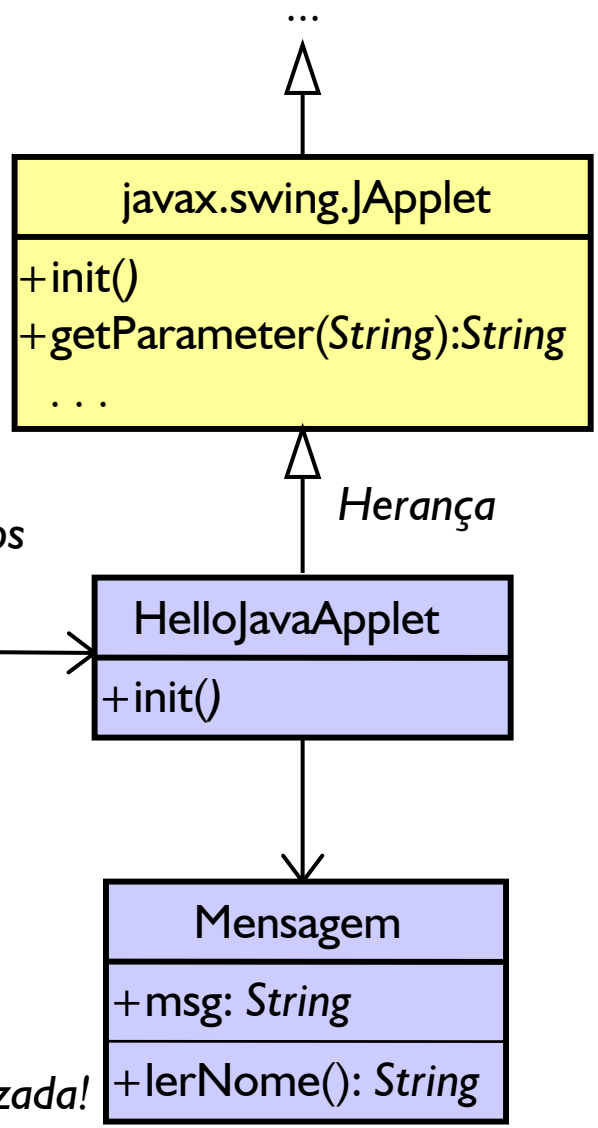
- *Para rodar o applet abra a página com seu browser ou use o appletviewer*

```
> appletviewer pagina.html
```

- *Mude o valor dos parâmetros do HTML e veja os resultados*

Browser oferece container para rodar o Applet

Relacionamentos



Resumo

- *Foram apresentados quatro exemplos de pequenas aplicações Java, demonstrando*
  - *Sintaxe elementar, compilação e execução*
  - *Classe como unidade de código para execução*
  - *Classe como definição de tipo de dados*
  - *Reuso de objetos através de associação (uso da classe Mensagem em três aplicações diferentes) e herança (infraestrutura de Applets reaproveitada)*
  - *Aplicações gráficas*
  - *Componentes de um framework (Applets) que executam em um container padrão (dentro do browser)*

Apêndice: como lidar com erros

Erros (I)

- Durante o desenvolvimento, erros podem ocorrer em dois domínios: tempo de compilação e tempo de execução
- Erros ocorridos durante a fase de compilação ocorrem quando se executa o **javac**, e são fáceis de corrigir. Há dois tipos:
  - Erros de processamento do arquivo (parsing): ponto-e-vírgula faltando, parênteses, aspas, chaves ou colchetes descasados. Identifica apenas o arquivo e a linha onde o erro pode ter iniciado. Um erro causa vários outros e nem sempre a mensagem é precisa.
  - Erros de compilação do código, realizada depois do parsing: além da linha e do arquivo, identificam a classe e método. Geralmente as mensagens são bastante elucidativas.
- É essencial aprender a identificar a causa da mensagem de erro
  - **LEIA** a mensagem e localize a linha onde o erro foi detectado
  - Corrija os erros na ordem em que eles aparecerem
  - Sempre recompile depois de corrigir cada erro de parsing (ponto-e vírgula, etc.) já que eles causam mensagens de erro falsas.

Alguns erros de compilação comuns

- *Cannot resolve symbol*: compilador é incapaz de localizar uma definição do símbolo encontrado. Causas comuns:
  - *Erro de sintaxe no nome de variável ou método*
  - *Variável/método não declarado*
  - *Classe usada não possui variável, método ou construtor*
  - *Número ou tipo de argumentos do método ou construtor incorretos*
  - *Definição de classe não encontrada no CLASSPATH*
- *Class Hello is public, should be declared in a file named Hello.java*: nome do arquivo tem que ser igual ao nome da classe pública*:
  - *Nome tem que ser Hello.java, literalmente. O nome hello.java causa este erro porque o "h" está minúsculo.*
  - *Para consertar altere o nome da classe no código ou no nome do arquivo para que sejam iguais.*

* Se classe não for pública, essa restrição não vale

Exemplos de erros de compilação

- *Erro de parsing*
  - *Na verdade, só há um erro no código, apesar do compilador acusar três*

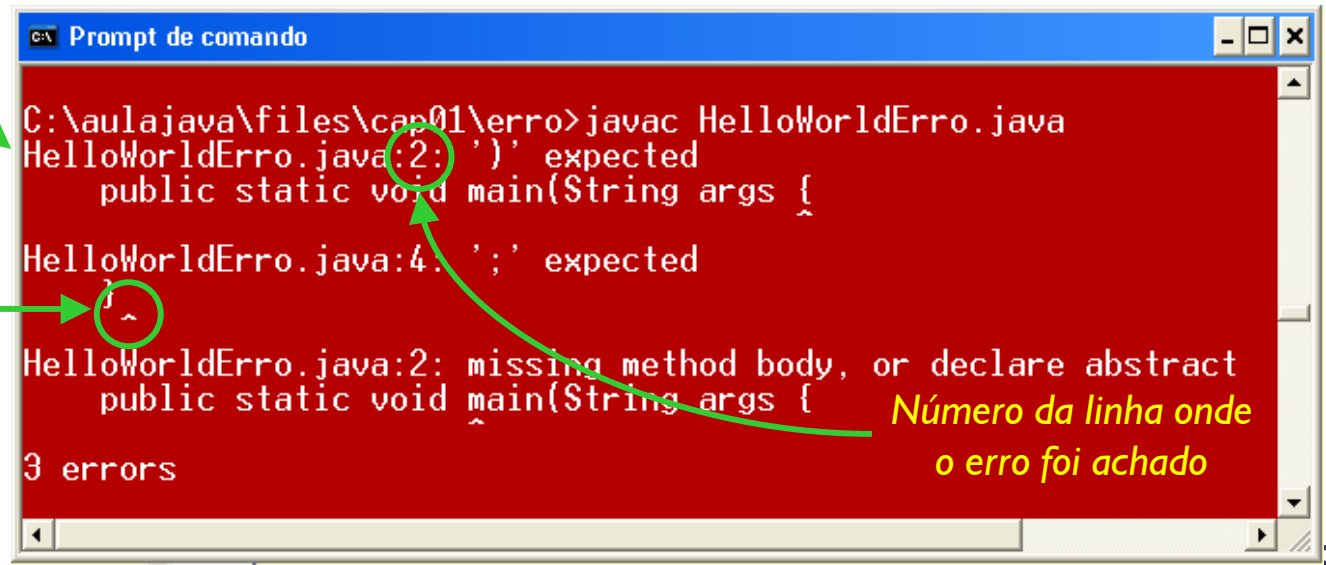
Apenas o primeiro erro é verdadeiro.

Ignore os outros. Eles foram causados pelo primeiro.

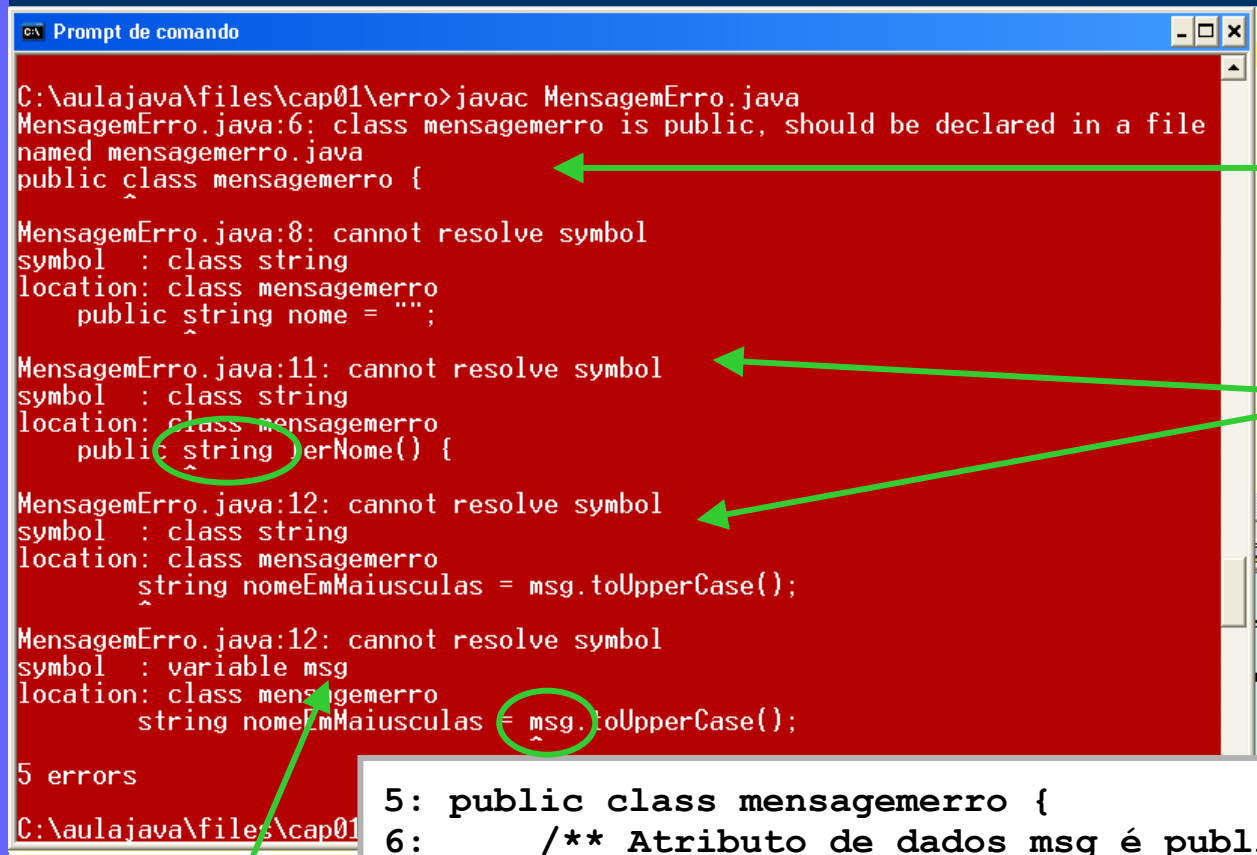
```
1: public class HelloWorldErro {
2:     public static void main(String args {
3:         System.out.println("Hello, 4: world!");
5:     }
6: }
```

Arquivo onde foi detectado o erro

Trecho do código e indicação da provável localização da causa do erro



Exemplos de erros de compilação (2)



Nome do arquivo é MensagemErro.java mas classe foi criada com nome mensagemerro.java

Compilador não sabe quem é string: O tipo String sempre tem um S maiúsculo (como todas as classes da API)

Compilador não sabe quem é msg: não foi declarada nenhuma variável com esse nome.

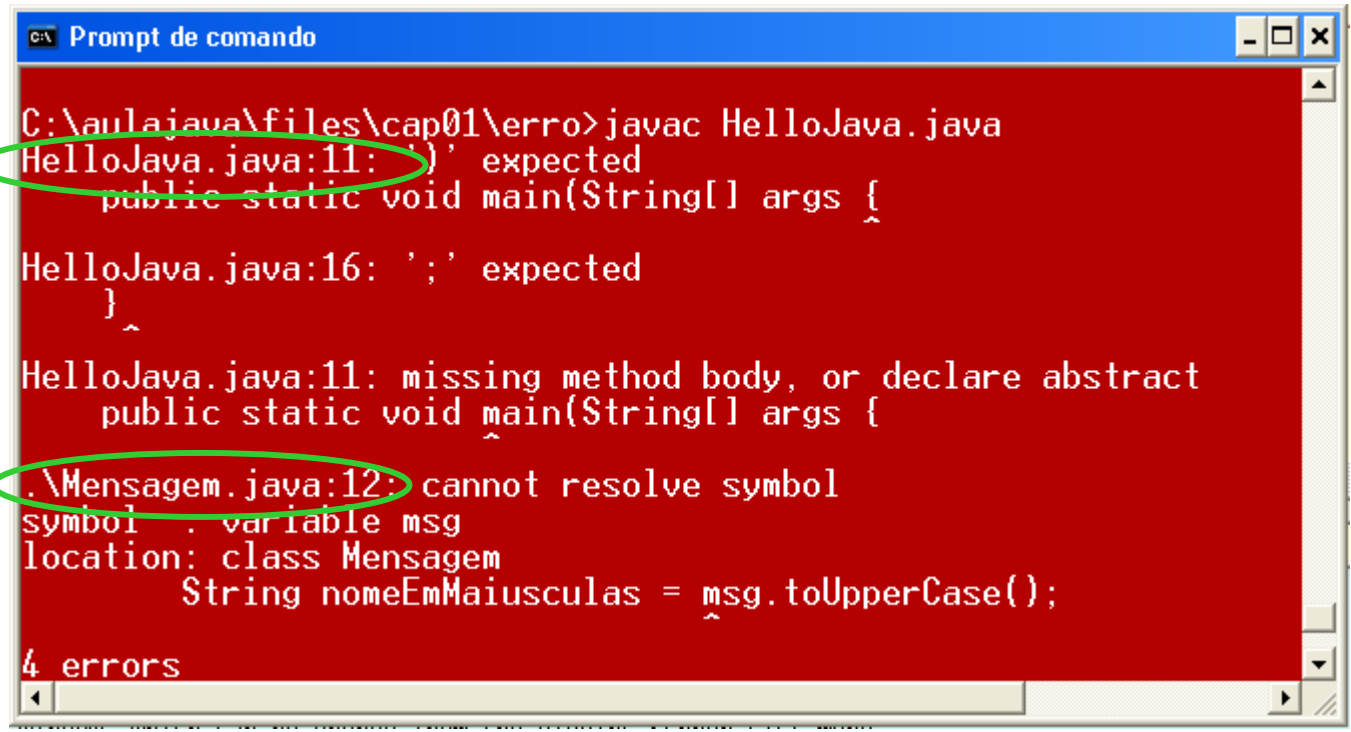
```
5: public class mensagemerro {
6:     /** Atributo de dados msg é publicamente visível */
7:     public string nome = "";
8:
9:     /** Método lerNome() devolve objeto do tipo String */
10:    public string lerNome() {
11:        string nomeEmMaiusculas = msg.toUpperCase();
12:        return nomeEmMaiusculas;
13:    }
14:}
```

Exemplos de erros de compilação (3)

- *Erros em múltiplas classes*
  - *Quando uma classe que possui dependências é compilada, suas dependências são compiladas primeiro e o compilador mostra mensagens de erros referentes a todas as classes envolvidas*
  - *Identifique sempre o arquivo e o número da linha*
  - *Compile as dependências primeiro*

Erro na linha 11 do arquivo HelloJava.java

Erro na linha 12 do arquivo Mensagem.java localizado no mesmo diretório que HelloJava.java



Erros (II)

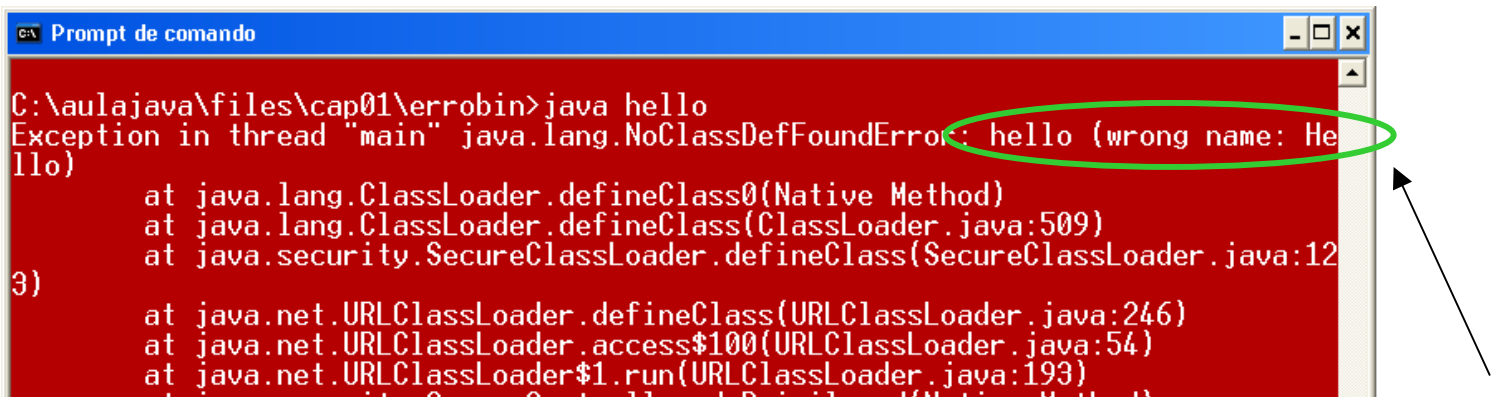
- Depois que o código compila com sucesso, os bytecodes (arquivos .class) são gerados e podem ser usados em um processo de execução
- Erros ocorridos durante a fase de execução (runtime) ocorrem quando se executa o interpretador ***java***, e são muito mais difíceis de localizar e consertar.
- A mensagem impressa geralmente é um "stack trace" e mostra todo o "caminho" percorrido pelo erro
  - Relaciona métodos e classes da sua aplicação e classes da API Java que sua aplicação usa (direta ou indiretamente)
  - Nem sempre mostra a linha de código onde o erro começou
  - O início do trace geralmente contém informações mais úteis
- Erros de runtime nem sempre indicam falhas no software
  - Freqüêntemente se devem a causas externas: não existência de arquivos externos, falta de memória, falha em comunicação de rede

Erros de execução comuns e possíveis causas

- *Exception in thread "main": NoClassDefFoundError: Classe*: *a classe "Classe" não foi encontrada no CLASSPATH.*
  - *O CLASSPATH não inclui todos os diretórios requeridos*
  - *O nome da classe foi digitado incorretamente ou requer pacote*
- *Exception in thread "main": NoSuchMethodError: main*: *o sistema tentou chamar main() mas não o encontrou.*
  - *A classe não tem método main() (talvez não seja executável)*
  - *Confira assinatura do main:* `public static void main(String[])`
- *ArrayIndexOutOfBoundsException*: *programa tentou acessar vetor além dos limites definidos.*
  - *Erro de lógica com vetores*
  - *Aplicação pode requerer argumentos de linha de comando*
- *NullPointerException*: *referência para objeto é nula*
  - *Variável de tipo objeto foi declarada mas não inicializada*
  - *Vetor foi declarado mas não inicializado*

Exemplos de erros de tempo de execução

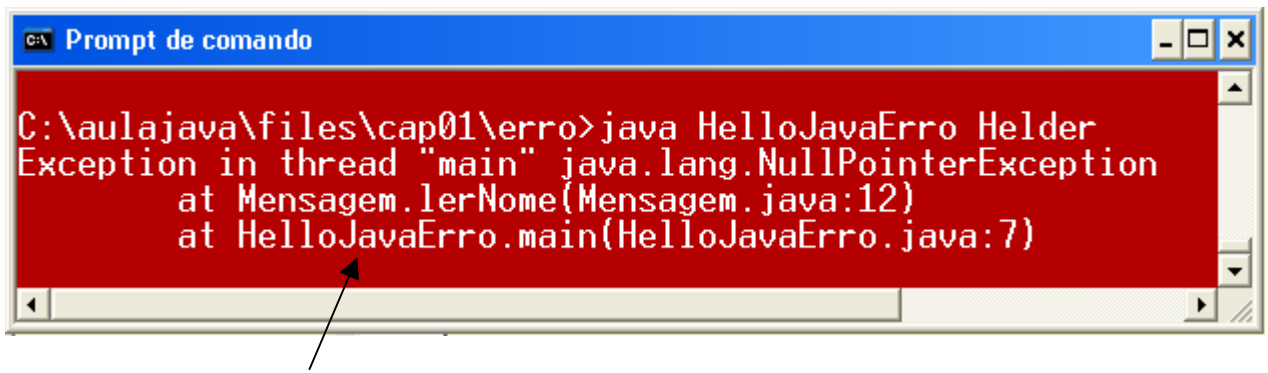
1. Na linha 13, está havendo uma referência a um índice de um vetor. Esse índice tenta acessar uma posição inexistente no vetor. Neste caso específico, o erro pode ser evitado passando-se um parâmetro após o nome da classe na linha de comando, porém a aplicação é pouco robusta pois não prevê que o erro possa acontecer.



2. A classe não foi encontrada. Pode ser que ela não esteja no CLASSPATH. Neste caso específico o interpretador nos sugere que o nome pode está errado. Para consertar, basta chamar a classe pelo nome correto: Hello. O Stack Trace mostra que esse erro teve origem em outras classes, mas foi nossa classe que, na verdade, o provocou.

Exemplos de erros de tempo de execução (2)

- Erros de tempo de execução freqüentemente ocorrem em dependências
  - Causa pode estar na dependência
  - Causa pode ter tido origem na dependência mas ter sido iniciada por erro na classe principal
- Stack Trace pode ajudar a localizar a origem do erro
  - As informações também podem desviar a atenção



Erro começou na linha 7 (método main) de HelloJavaErro mas teve origem na linha 12 do método lerNome de Mensagem

Como achar erros de tempo de execução

- Há dois tipos de erros de tempo de execução
  - Causados por situações externas, que fogem do controle do programador (ex: rede fora do ar)
  - Causados por erros de lógica de programação
- Devemos criar aplicações robustas que prevejam a possibilidade de erros de tempo de execução devido a fatores externos e ajam da melhor forma possível
- Devemos achar erros de lógica e evitar que sobrevivam além da fase de desenvolvimento. Para evitá-los:
  - Escreva código claro, fácil de entender, organizado, pequeno
  - Use endentação, siga convenções, nomes significativos, documente
  - Escreva testes para todo código e rode-os com freqüência
- Para achar os erros difíceis
  - Rode código de testes se os tiver; Ative nível de mensagens de log
  - Aprenda a usar um depurador para navegar no fluxo de execução

Exercícios

- *1. Compile e execute os exemplos localizados no subdiretório cap01/*
- *2. Há vários arquivos no diretório cap01/exercicios/erro. Todos apresentam erros de compilação. Corrija os erros.*
- *3. Execute os arquivos executáveis do diretório errobin (quais são?). Alguns irão provocar erros de tempo de execução. Corrija-os ou descubra como executar a aplicação sem que eles ocorram.*

Curso J100: Java 2 Standard Edition

Revisão 17.0

(helder@acm.org)

argonavis.com.br