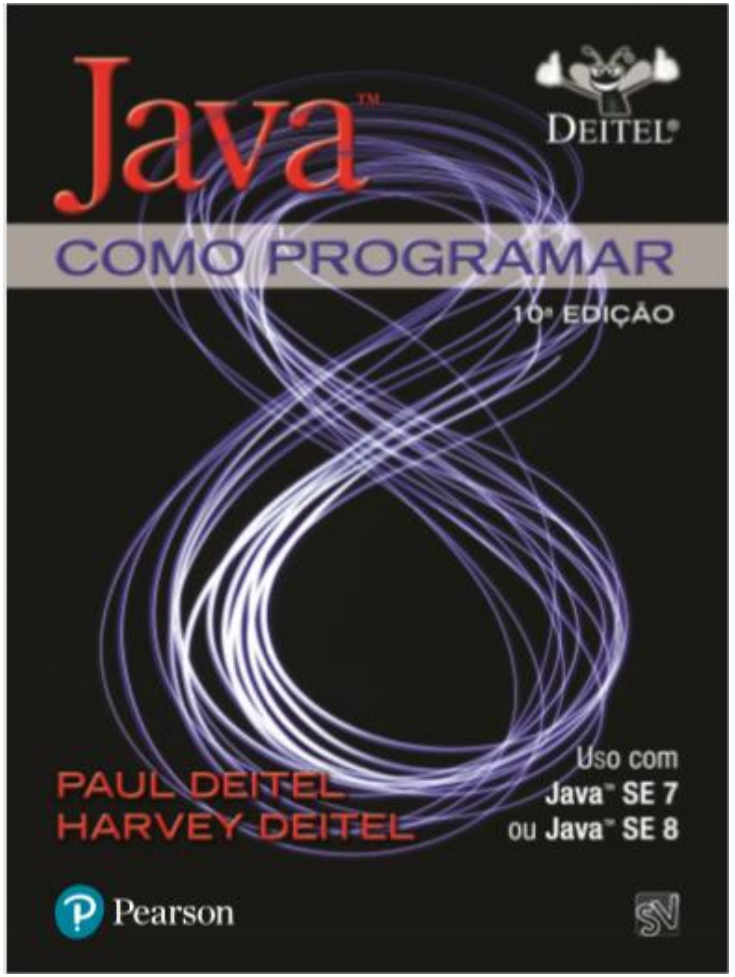
EAGS - CAP 2022

JAVA – EAGS - CAP 2022

Conteúdo do Edital: CAP

- Linguagem de programação JAVA;
- Bibliotecas de classe do Java;
- Classes e Objetos;
- Instruções de controle;
- Módulos de programa em Java;
- Arrays e Arraylists;
- Programação orientada a objetos;
- Tratamento de exceções;
- Componentes GUI;
- Strings, caracteres e expressões regulares;
- Recursão;
- Applets e Java Web Start;
- Multithreading; e
- Serviços Web.

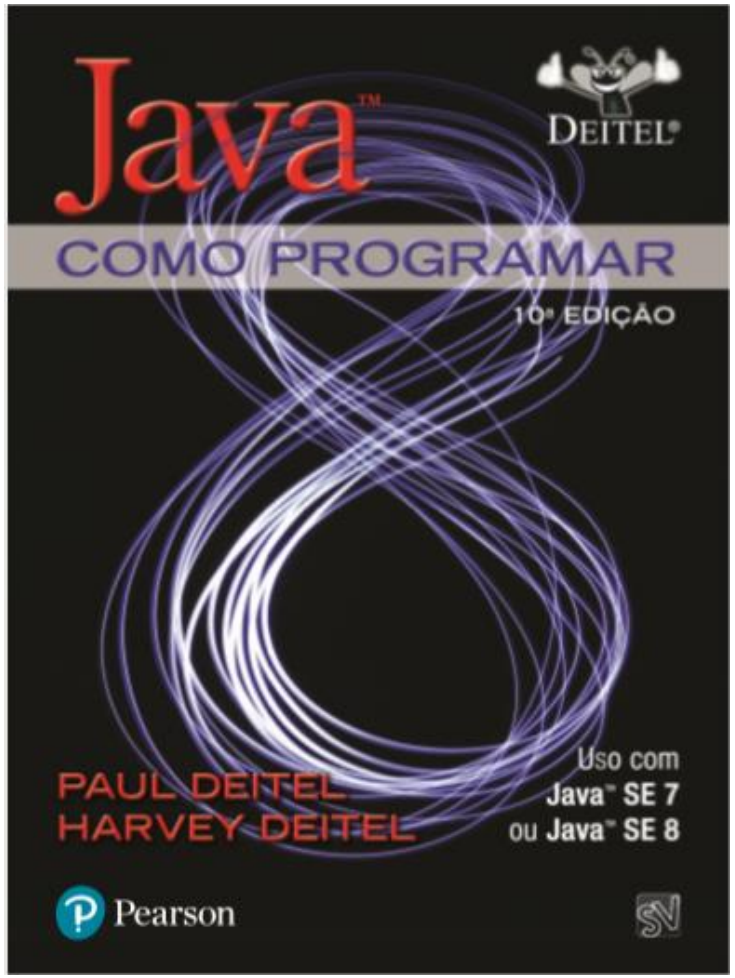


DEITEL, Paul; DEITEL, Harvey. JAVA como Programar. 10.ed. [S.l.]: Pearson Prentice Hall

JAVA – EAGS - CAP 2022

Conteúdo do Edital: EAGS

- Estrutura.
- Variáveis.
- Classes.
- Atributos.
- Métodos.
- Herança.
- Polimorfismo.
- Encapsulamento.
- Estrutura de Controle.
- Recursividade.;



JAVA – EAGS - CAP 2022

AULA 1 - JAVA

- Linguagem de programação JAVA;
- Bibliotecas de classe do Java;
- Desenvolvimento do primeiro programa em Java
- Como instalar o pacote Java e a IDE Netbeans
- Ciclo de desenvolvimento de um programa em java
- Formas de compilação e execução de um programa em Java
- JDK X JRE
- O que é a JVM
- Tipos de aplicações Java
- Módulos de programa em Java;
- Palavras reservadas da Linguagem Java
- Classes
- Objetos
- Tipos de Dados
- Variáveis, Literais e Pacotes
- Exemplos práticos no prompt de comandos
- Exemplos práticos no netbeans
- Operadores aritméticos e Lógicos
- Estruturas de Controle

Capítulo 1 – Introdução a Programação Java

1.9 História do Java

A contribuição mais importante da revolução do microprocessador até essa data é que ele tornou possível o desenvolvimento de computadores pessoais, que agora contam com mais de um bilhão em todo o mundo. Os computadores pessoais afetaram profundamente a vida das pessoas e a maneira que as organizações conduzem e gerenciam seus negócios.

Os microprocessadores têm um impacto profundo em dispositivos eletrônicos inteligentes de consumo popular. Reconhecendo isso, a Sun Microsystems, em 1991, financiou um projeto de pesquisa corporativa interna que resultou em uma linguagem baseada em C++ que seu criador, James Gosling, chamou de Oak em homenagem a uma árvore de carvalho vista por sua janela na Sun. Descobriu-se mais tarde que já havia uma linguagem de computador com esse nome. Quando uma equipe da Sun visitou uma cafeteria local, o nome Java (cidade de origem de um tipo de café importado) foi sugerido; e o nome pegou.

O projeto de pesquisa passou por algumas dificuldades. O mercado para dispositivos eletrônicos inteligentes destinados ao consumidor final não estava se desenvolvendo tão rapidamente como a Sun tinha previsto. Por uma feliz casualidade, 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. Isso deu nova vida ao projeto.

A Sun anunciou o Java formalmente em uma conferência do setor em maio de 1995. O Java chamou a atenção da comunidade de negócios por causa do enorme interesse na Web. O Java é agora utilizado para desenvolver aplicativos corporativos de grande porte, aprimorar a funcionalidade de servidores da Web (os computadores que fornecem o conteúdo que vemos em nossos navegadores da Web), fornecer aplicativos para dispositivos voltados para o consumo popular (como telefones celulares, pagers e PDAs) e para muitos outros propósitos.

Capítulo 1 – Introdução a Programação Java

1.10 Bibliotecas de classe do Java

Programas Java consistem em partes chamadas classes. As classes incluem partes chamadas **métodos** que realizam tarefas e retornam informações quando as tarefas são concluídas. Você pode criar cada parte necessária para formar seus programas Java. Entretanto, a maioria dos programadores Java tira proveito das ricas coleções de classes existentes nas **bibliotecas de classe Java**, que também são conhecidas como **Java APIs (Application Programming Interfaces)**. Portanto, na realidade há dois aspectos para aprender o "mundo" do Java. O primeiro aspecto é a própria linguagem Java, de modo que você possa programar suas próprias classes, o segundo são as classes nas extensas bibliotecas de classe Java. Por todo este livro discutimos muitas classes da Java API preexistentes. Para baixar a documentação da Java API, acesse `java.sun.com/javase/downloads/`, role para baixo até a Seção Additional Resources e clique no botão Download à direita de Java SE 6 Documentation.

Observação de engenharia de software 1.1

Utilize uma abordagem de bloco de construção para criar seus programas. Evite reinventar a roda — utilize partes existentes onde quer que possível. Essa reutilização de software é um benefício fundamental da programação orientada a objetos.

Incluímos muitas **Observações de engenharia de software** por todo o livro para explicar conceitos que afetam e aprimoram a arquitetura total e a qualidade de sistemas de software. Também destacamos outros tipos de dicas, incluindo:

- **Boas práticas de programação** (para ajudá-lo a escrever programas que são mais claros, mais compreensíveis, mais fáceis de manter e mais fáceis de testar e **depurar** — isto é, remover erros de programação).
- **Erros de programação comuns** (problemas a observar e evitar).
- **Dicas de desempenho** (técnicas para escrever programas que executam mais rapidamente e utilizam menos memória).
- **Dicas de portabilidade** (técnicas para ajudá-lo a escrever programas que podem executar, com pouca ou nenhuma modificação, em diferentes computadores — essas dicas também incluem observações gerais de como o Java alcança seu alto grau de portabilidade).
- **Dicas de prevenção de erros** (técnicas para remover bugs dos seus programas e, mais importante, antes de tudo, técnicas para escrever programas sem bugs).
- **Observações sobre a interface** (técnicas para ajudá-lo a projetar a "aparência" e o "comportamento" das interfaces com o usuário dos seus aplicativos em termos da aparência e facilidade de uso).

Muitas dessas técnicas são apenas diretrizes. Sem dúvida, você desenvolverá seu próprio estilo de programação preferido.

Capítulo 1 – Introdução a Programação Java

A Tecnologia JAVA Linguagem de programação

Java é uma linguagem de programação orientada a objetos, robusta, elegante e com todas as características das linguagens modernas, podendo ser utilizada nos mais diversos ambientes, desde aplicações simples no desktop até complexos aplicativos web, programação de robôs, redes de sensores, celulares e televisão digital interativa, além de muitos outros.

Capítulo 1 – Introdução a Programação Java

Arquitetura e Desenvolvimento

- 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)

Capítulo 1 – Introdução a Programação Java

Produtos e APIs

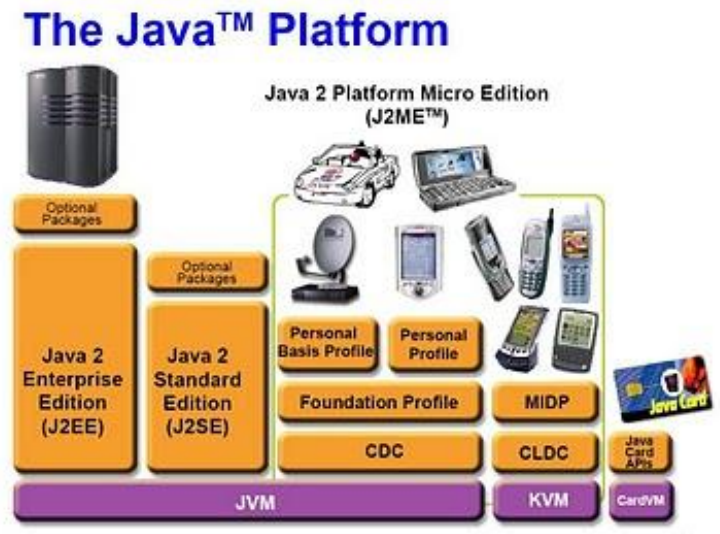
Java possui uma coleção de APIs (bibliotecas) padrão que podem ser usadas para construir aplicações
Organizadas em pacotes (java.*, javax.* e extensões)
Usadas pelos ambientes de execução (JRE) e de desenvolvimento (JDK)

Plataforma e Tipos de Aplicação

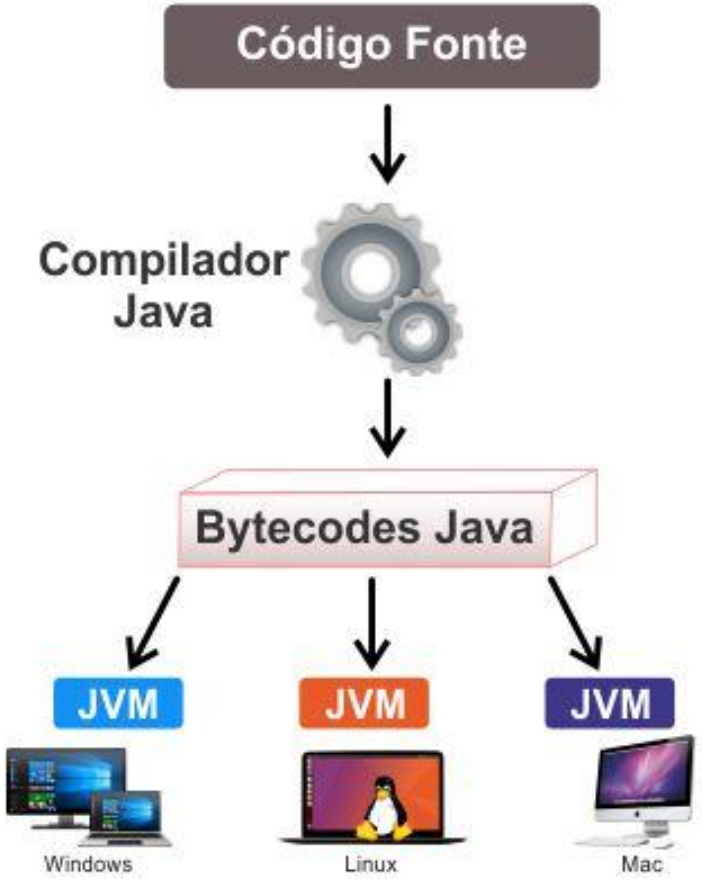
Plataformas: Java é portável, portanto pode ser executado em qualquer plataforma que tenha suporte a sua JVM.
Aplicativos: programas escritos em Java que não necessitam de um navegador para serem utilizados.
Applets: programas escritos em Java, normalmente carregados na WWW e executados por um navegador.
Servlets: programas escritos em Java, normalmente carregados na WWW e executados pelo Servidor WEB.

PLATAFORMAS

	Standard Edition	Enterprise Edition	Micro Edition
Versões em Março/2010	Java2SE	Java2EE	Java2ME
Caracteristicas	Console, desktop, applets, etc	Servlet, JSP, EJB, Componentes, Sistemas Distribuídos, etc	Dispositivos móveis, TV Digital, etc..



Ambiente de Desenvolvimento Java



Requisitos – Instalando e configurando o Java

Java Development Kit (JDK)

A JDK provê todos os componentes e recursos necessários para o desenvolvimento de aplicações Java.

Java Runtime Environment (JRE)

A JRE é a coleção de software que permite um sistema de computador executar uma aplicação Java. Esta coleção é composta pela Java Virtual Machines (JVMs) que interpreta o bytecode para o código de máquina, bibliotecas de classe padrão, ferramentas para interface com o usuário e uma variedade de utilitários.

Tudo o que é necessário para executar aplicações Java
Parte do JDK 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.X.0
Linux: JAVA_HOME=/usr/java/j2sdk1.X.0 export JAVA_HOME
Bibliotecas de classe do Java

Requisitos – Básico do Desenvolvimento

Java Virtual Machine (JVM)

Máquina virtual Java (em inglês: Java Virtual Machine, JVM) é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina. A JVM é responsável pelo gerenciamento dos aplicativos, à medida que são executados.

Application programming Interface (API)

O java tem uma ampla gama de recursos padrão: desde tocar um som até codificar uma mensagem de correio eletrônico. Esses recursos fazem parte da API (Application Programming Interface) padrão Java.

O API é um grupo de programas de suporte destinados a cumprir funções específicas que é dividido em diferentes partes funcionais chamadas pacotes (o que nós denominamos bibliotecas).

Pacotes mais usados da API Java:

Pacote API	Suporta
java.applet	**Recursos gerais dos applets**
java.awt	**Recursos gráficos**: botão, barra, caixas de texto, janela, etc. Em outras palavras: recursos GUI (Graphical User Interface).
java.io	Input e output de dados (entrada e saída)
java.lang	Recursos de linguagem
java.math	**Operações matemáticas**
java.net	Rede
java.security	Segurança
java.text	**Recursos de texto**
java.util	Miscelânea de recursos utilitários

Requisitos – Básico do Desenvolvimento

Integrated Development Environment ou Ambiente de Desenvolvimento Integrado (IDE)

Um ambiente de desenvolvimento integrado (IDE) é um software para criar aplicações que combinam ferramentas comuns de desenvolvimento em uma única interface gráfica do usuário (GUI).

Exemplos: NetBeans, Eclipse, BlueJ, IntelliJ IDEA, Jdeveloper …

Gerenciamento de memória e Coletor de Lixo

Todo objeto que você cria utiliza vários recursos do sistema, como a memória. Precisamos de uma maneira disciplinada de retornar os recursos para o sistema quando eles não são mais necessários, para evitar “vazamentos de recurso”, o Java Virtual Machine (JVM) realiza coleta de lixo automática para reivindicar a memória ocupada por objetos que não estão mais em uso.

Quando não houver mais referências a um objeto, o objeto é marcado para coleta de lixo pela JVM. A memória desse objeto pode ser liberada quando a JVM executa seu coletor de lixo, responsável por recuperar a memória dos objetos que não são mais utilizados de modo que a memória possa ser utilizada para outros objetos.

Portanto, vazamentos de memória que são comuns em outras linguagens como C e C++ (porque a memória não é automaticamente liberada nessas linguagens) são menos prováveis em Java (mas alguns ainda podem acontecer de maneiras sutis).

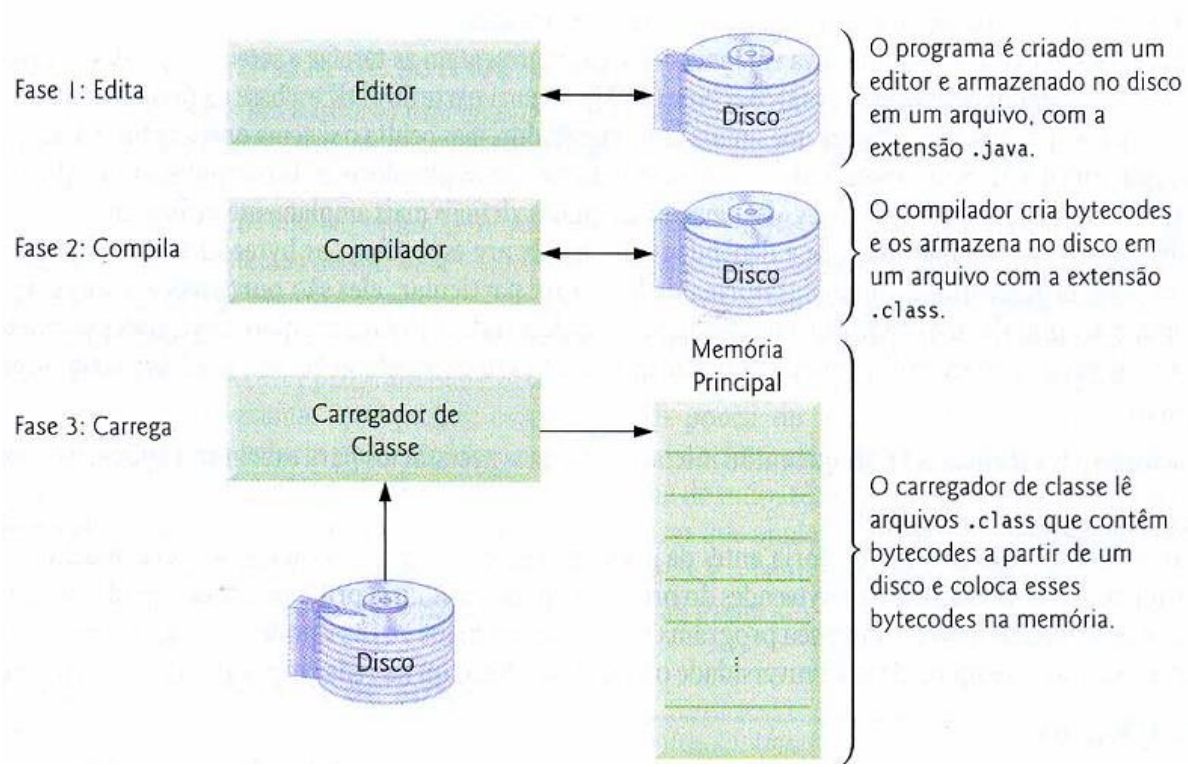
Outros tipos de vazamentos de recursos podem ocorrer. Por exemplo, um aplicativo poderia abrir um arquivo no disco para modificar o conteúdo do arquivo. Se o aplicativo não fechar o arquivo, nenhum outro aplicativo poderá utilizar o arquivo até que o aplicativo que abriu o arquivo conclua.

O método finalize é chamado pelo coletor de lixo para realizar limpeza de terminação sobre um objeto um pouco antes de o coletor de lixo re ivindicar a memória do objeto.

O método finalize não aceita parâmetros e tem o tipo de retorno void. Um problema com relação ao método finalize é que não há garantias de o coletor de lixo executar em uma hora especificada. De fato, o coletor de lixo nunca pode executar antes de um programa terminar. Portanto, não fica claro se, ou quando, o método finalize será chamado.Por essa razão, a maioria dos programadores deve evitar o método finalize.

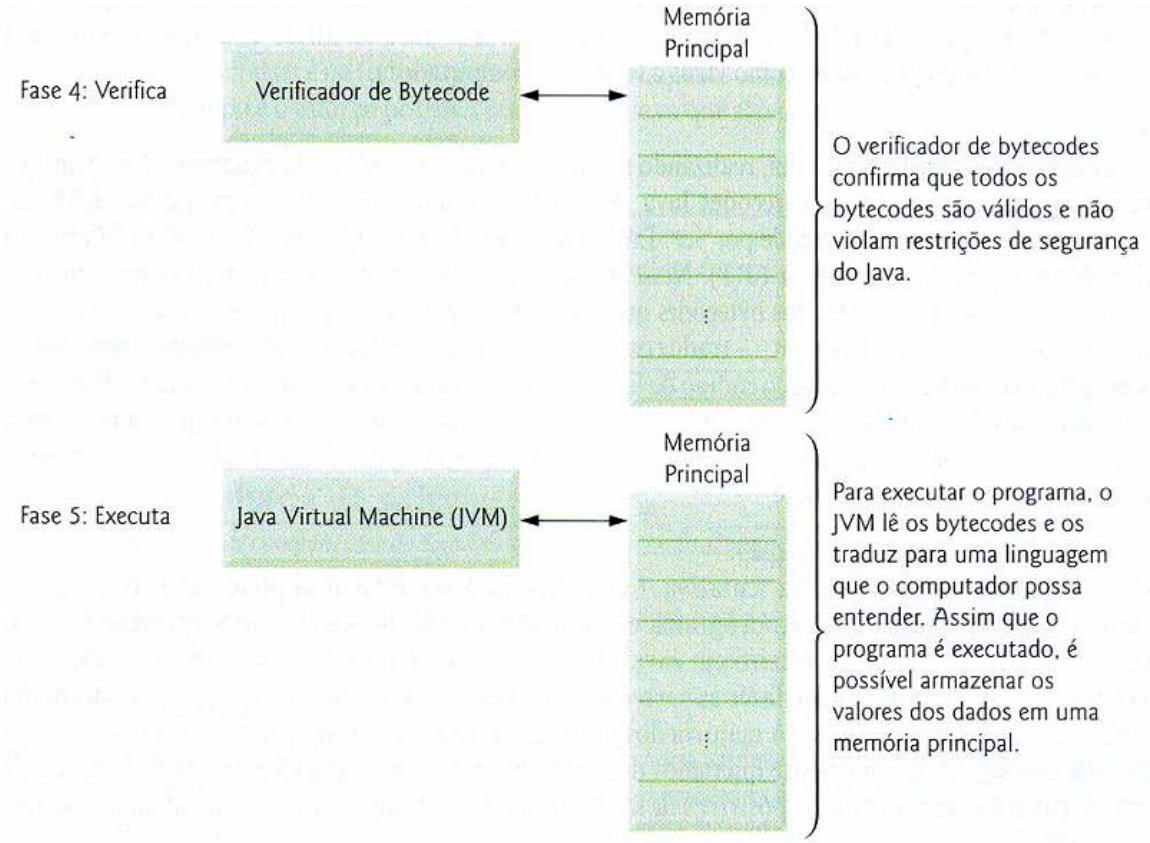
Requisitos – Básico do Desenvolvimento

Editar, Compilar, Carregar, Verificar e Executar.



Requisitos – Básico do Desenvolvimento

Editar, Compilar, Carregar, Verificar e Executar.



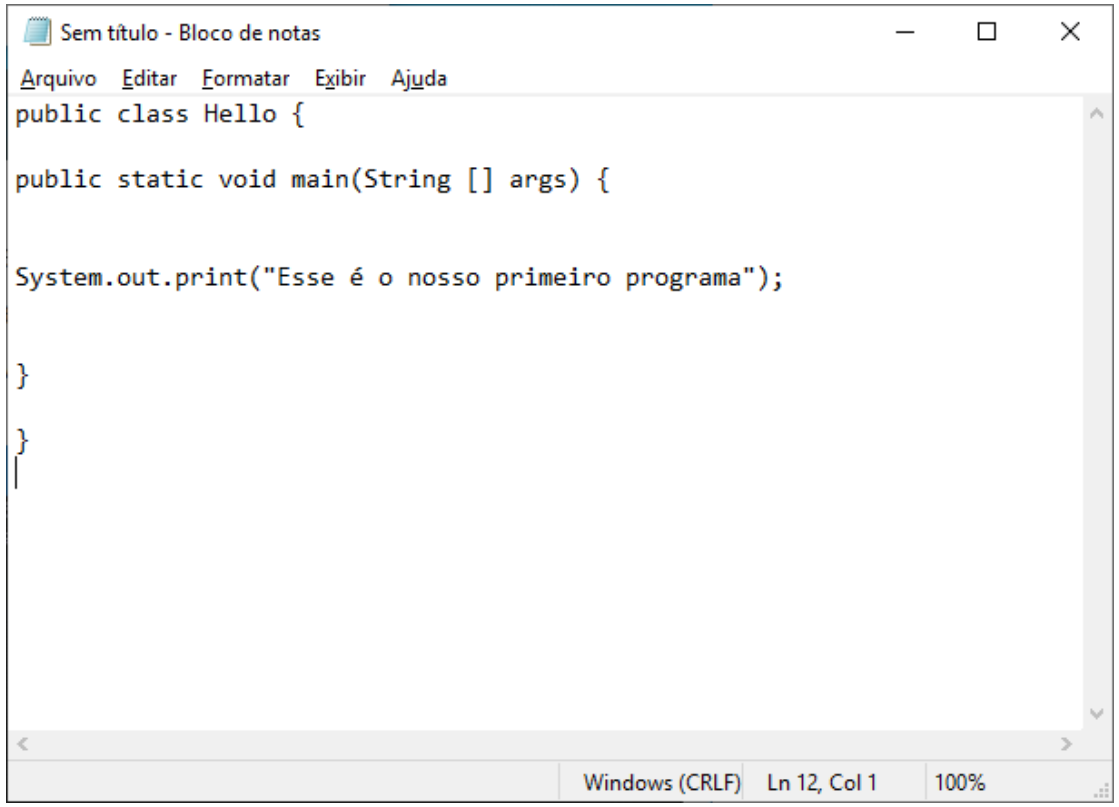
Requisitos – Primeiro Programa

Meu primeiro programa

```
1     public class Hello {
2     /**
3     * Nosso primeiro programa em Java
4     */
5
6      public static void main(String[ ] args) {
7      // A próxima linha imprime a frase   entre (“…”) no console da IDE
8      System.out.println("Esse é nosso primeiro programa");
9        }
10  }
```

JAVA – EAGS - CAP 2022

Requisitos – Primeiro Programa

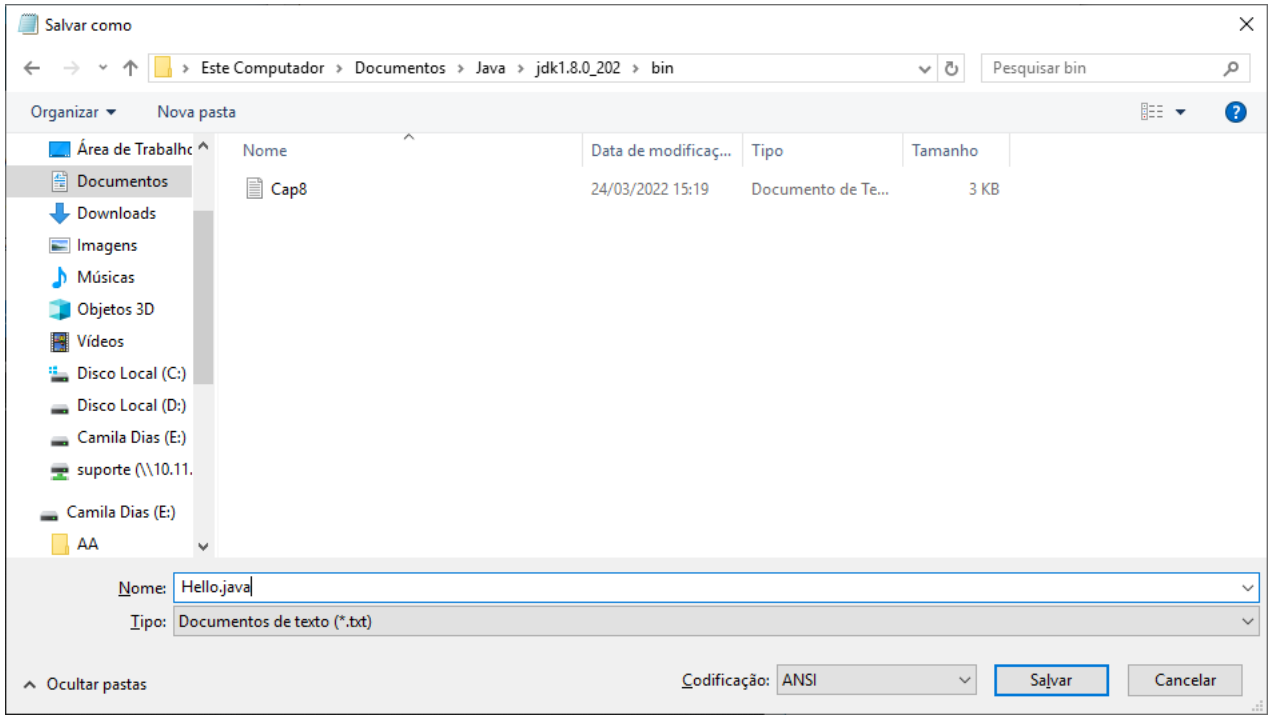


Salvar na variável de ambiente:
JAVA_HOME

JAVA – EAGS - CAP 2022

Requisitos – Primeiro Programa

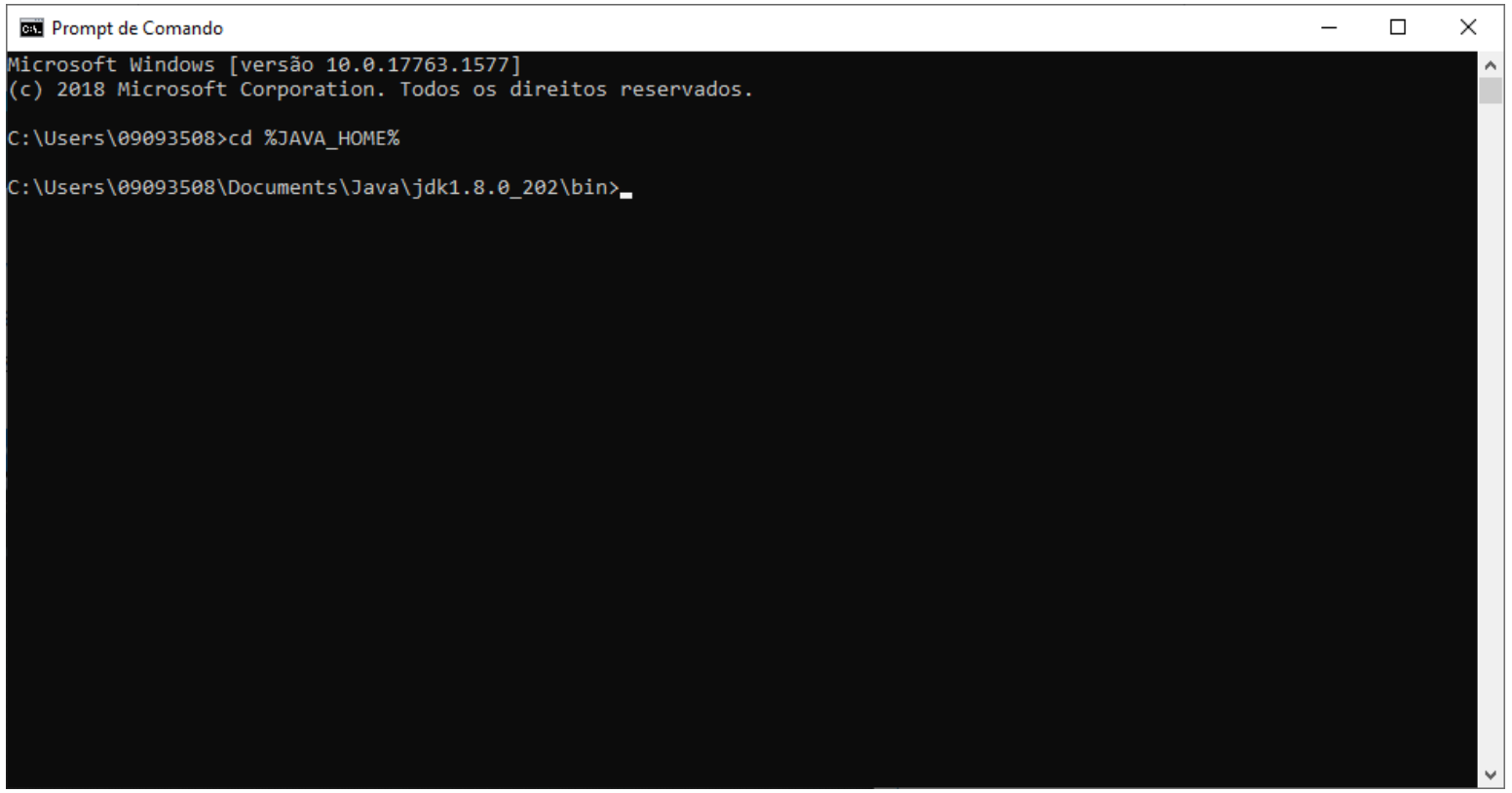
Salvar na variável de ambiente:
JAVA_HOME



JAVA – EAGS - CAP 2022

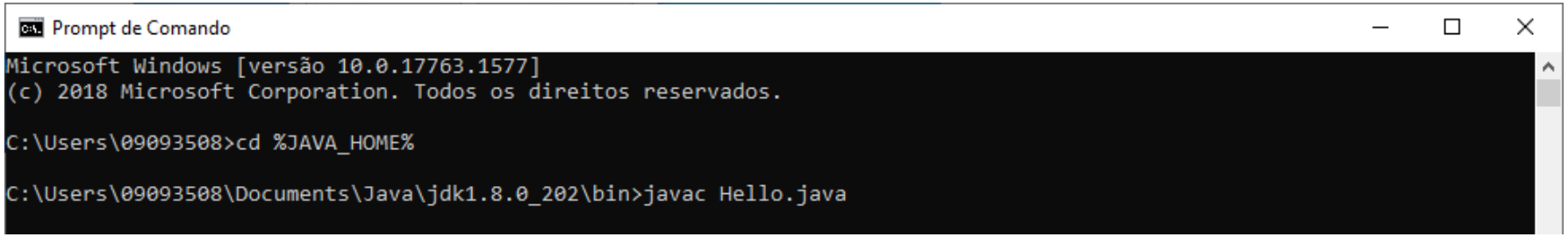
Requisitos – Primeiro Programa

1 – Abra o prompt de comandos e digite o comando para abrir o diretório por meio da variável de ambiente:

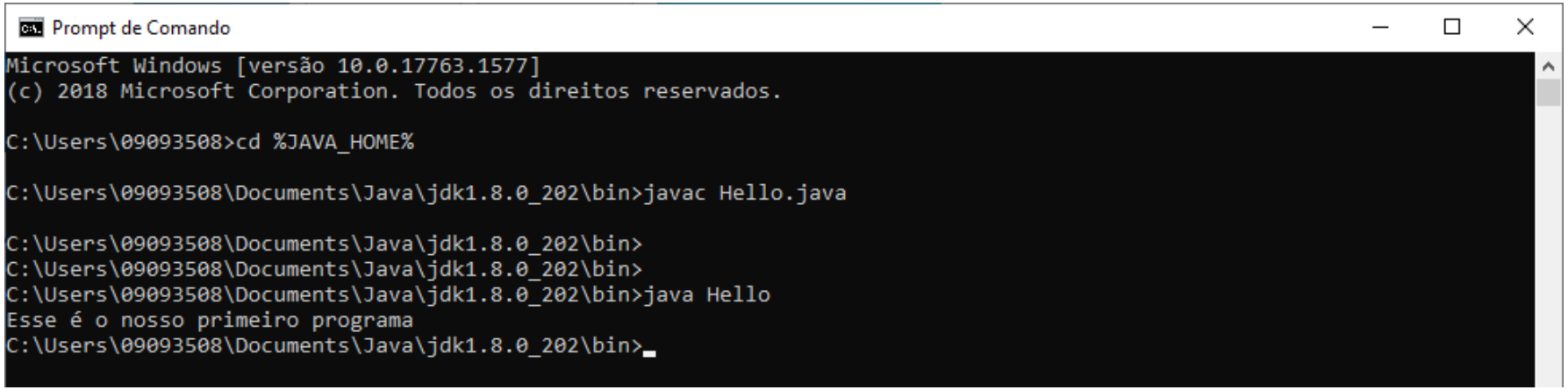


Requisitos – Primeiro Programa

2- Compile o programa:



3- Execute o programa:



Requisitos – Primeiro Programa

Um aplicativo Java é um programa de computador que é executado quando você utiliza o comando java para carregar a Java Virtual Machine (JVM).

```
1  // Figura 2.1: Welcome1.java
2  // Programa de impressão de texto.
3
4  public class Welcome1
5  {
6     // método main inicia a execução do aplicativo Java
7     public static void main(String[] args)
8     {
9        System.out.println("Welcome to Java Programming!");
10    } // fim do método main
11 } // fim da classe Welcome1
```

```
Welcome to Java Programming!
```

Figura 2.1 | Programa de impressão de texto.

Capítulo 2 – Livro de Java 10ª Edição

Comentando programas

Inserimos **comentários** para **documentar programas** e aprimorar sua legibilidade. O compilador Java *ignora* os comentários, portanto, eles *não* fazem o computador realizar qualquer ação quando o programa é executado.

Por convenção, iniciamos cada programa com um comentário indicando o número da figura e o nome do arquivo. O comentário na linha 1

```
// Figura 2.1: Welcome1.java
```

começa com `//`, indicando que é um **comentário de fim de linha**, e termina no fim da linha, onde os caracteres `//` aparecem. Um comentário de fim de linha não precisa começar uma linha; ele também pode começar no meio de uma linha e continuar até o final (como nas linhas 6, 10 e 11). A linha 2

```
// Programa de impressão de texto.
```

de acordo com nossa convenção, é um comentário que descreve o propósito do programa.

O Java também tem **comentários tradicionais**, que podem ser distribuídos ao longo de várias linhas, como em

```
/* Esse é um comentário tradicional. Ele
   pode ser dividido em várias linhas */
```

Eles começam e terminam com delimitadores, `/*` e `*/`. O compilador ignora todo o texto entre os delimitadores. O Java incorporou comentários tradicionais e comentários de fim de linha das linguagens de programação C e C++, respectivamente. Preferimos usar comentários `//`.

O Java fornece comentários de um terceiro tipo: **comentários Javadoc**. Esses são delimitados por `/**` e `*/`. O compilador ignora todo o texto entre os delimitadores. Os comentários no estilo Javadoc permitem-lhe incorporar a documentação do programa diretamente aos seus programas. Esses comentários são o formato de documentação Java preferido na indústria. O **programa utilitário** `javadoc` (parte do JDK) lê comentários Javadoc e os usa para preparar a documentação do programa no formato HTML. Demonstraremos comentários Javadoc e o utilitário `javadoc` no Apêndice G, na Sala Virtual (em inglês), "Criando documentação com `javadoc`".

JAVA – EAGS - CAP 2022

Capítulo 2 – Livro de Java 10ª Edição

Erro comum de programação 2.1

Esquecer um dos delimitadores de um comentário tradicional ou Javadoc causa um erro de sintaxe. Um erro de sintaxe ocorre quando o compilador encontra o código que viola as regras da linguagem do Java (isto é, sua sintaxe). Essas regras são semelhantes às da gramática de um idioma natural que especifica a estrutura da sentença. Erros de sintaxe também são chamados erros de compilador, erros em tempo de compilação ou erros de compilação, porque o compilador os detecta ao compilar o programa. Quando um erro de sintaxe é encontrado, o compilador emite uma mensagem de erro. Você deve eliminar todos os erros de compilação antes que o programa seja compilado corretamente.

JAVA – EAGS - CAP 2022

Capítulo 2 – Livro de Java 10ª Edição

Declarando uma classe

A linha 4

```
public class Welcome1
```

começa uma **declaração de `class`** para a classe `Welcome1`. Todo programa Java consiste em pelo menos uma classe que você (o programador) define. A **palavra-chave `class`** introduz uma declaração de classe e é imediatamente seguida pelo **nome da classe** (`Welcome1`). **Palavras-chave** (às vezes chamadas de **palavras reservadas**) são reservadas para uso pelo Java e sempre escritas com todas as letras minúsculas. A lista completa de palavras-chave é mostrada no Apêndice C.

Nos capítulos 2 a 7, cada classe que definimos inicia com a palavra-chave `public`. Por enquanto, simplesmente exigimos `public`. Você aprenderá mais sobre as classes `public` e não `public` no Capítulo 8.

Nome de arquivo para uma classe `public`

Uma classe `public` *deve* ser inserida em um arquivo com um nome na forma *NomeDaClasse*`.java`, assim a classe `Welcome1` é armazenada no arquivo `Welcome1.java`.

Erro comum de programação 2.2

Um erro de compilação ocorre se um nome de arquivo da classe `public` não for exatamente igual ao nome dessa classe (tanto em termos de ortografia como capitalização), seguido pela extensão `.java`.

Capítulo 2 – Livro de Java 10ª Edição

Nomes e identificadores de classe

Por convenção, os nomes de classes iniciam com uma letra maiúscula e apresentam a letra inicial de cada palavra que eles incluem em maiúscula (por exemplo, `SampleClassName`). O nome de uma classe é um **identificador** — uma série de caracteres que consiste em letras, dígitos, sublinhados (`_`) e sinais de cifrão (`$`) que *não* inicie com um dígito e *não* contenha espaços. Alguns identificadores válidos são `Welcome1`, `$valor`, `_valor`, `m_campoDeEntrada1` e `botao7`. O nome `7botao` *não* é um identificador válido porque inicia com um dígito, e o nome `campo de entrada` não é um identificador válido porque contém espaços. Normalmente, um identificador que não inicia com uma letra maiúscula não é um nome de classe. O Java faz **distinção entre maiúsculas e minúsculas** — letras maiúsculas e letras minúsculas são diferentes — assim, `value` e `Value` são identificadores diferentes (mas ambos válidos).

Corpo de classe

A **chave esquerda** (como na linha 5), `{`, inicia o **corpo** de cada declaração de classe. Uma **chave direita** correspondente (na linha 11), `}`, deve terminar cada declaração de classe. As linhas 6 a 10 são recuadas.

Capítulo 2 – Livro de Java 10ª Edição

Declarando um método

A linha 6

```
// método main inicia a execução do aplicativo Java
```

é um comentário de fim de linha que indica o propósito das linhas 7 a 10 do programa. A linha 7

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

é o ponto de partida de cada aplicativo Java. Os **parênteses** depois do identificador `main` indicam que ele é um bloco de construção do programa chamado **método**. Declarações de classe Java normalmente contêm um ou mais métodos. Para um aplicativo Java, um dos métodos *deve* ser chamado `main` e ser definido como mostrado na linha 7; caso contrário, a JVM não executará o aplicativo. Os métodos realizam tarefas e podem retornar informações quando as completam. Explicaremos o propósito da palavra-chave `static` na Seção 3.2.5. A palavra-chave `void` indica que esse método *não* retornará nenhuma informação. Mais tarde, veremos como um método pode fazer isso. Por enquanto, basta simular a primeira linha de `main` nos aplicativos Java. Na linha 7, a `String[] args` entre parênteses é uma parte necessária da declaração `main` do método, que discutiremos no Capítulo 7.

A chave esquerda na linha 8 inicia o **corpo da declaração do método**. Uma chave direita correspondente deve terminá-la (linha 10). A linha 9 no corpo do método é recuada entre as chaves.

Capítulo 2 – Livro de Java 10ª Edição

Gerando saída com `System.out.println`

A linha 9

```
System.out.println("Welcome to Java Programming!");
```

instrui o computador a executar uma ação, ou seja, exibir os caracteres contidos entre as aspas duplas (as próprias aspas *não* são exibidas). Juntos, as aspas e os caracteres entre elas são uma **string** — também conhecida como **string de caracteres** ou **string literal**. Os caracteres de espaço em branco em strings *não* são ignorados pelo compilador. As strings *não podem* distribuir várias linhas de código.

O objeto `System.out` — que é predefinido para você — é conhecido como **objeto de saída padrão**. Ele permite que um aplicativo Java exiba informações na **janela de comando** a partir da qual ele é executado. Em versões recentes do Microsoft Windows, a janela de comando chama-se Prompt de Comando. No Unix/Linux/Mac OS X, a janela de comando é chamada **janela terminal** ou **shell**. Muitos programadores simplesmente a chamam **linha de comando**.

O método `System.out.println` exibe (ou imprime) uma linha de texto na janela de comando. A string entre parênteses na linha 9 é o **argumento** para o método. Quando `System.out.println` completa sua tarefa, ele posiciona o cursor de saída (o local em que o próximo caractere será exibido) no começo da linha seguinte na janela de comando. Isso é semelhante àquilo que acontece ao pressionar a tecla *Enter* depois de digitar em um editor de texto — o cursor aparece no início da próxima linha no documento.

A linha 9 inteira, incluindo `System.out.println`, o argumento `"Welcome to Java Programming!"` entre parênteses e o **ponto e vírgula** (`;`), é uma **instrução**. Um método normalmente contém uma ou mais instruções que executam a tarefa. A maioria das instruções acaba com um ponto e vírgula. Quando a instrução na linha 9 executa, ela exibe `Welcome to Java Programming!` na janela de comando.

Ao aprender a programar, às vezes é útil "quebrar" um programa funcional para você poder familiarizar-se com as mensagens de erro de sintaxe do compilador. Essas mensagens nem sempre declaram o problema exato no código. Ao encontrar um erro, ele lhe dará uma ideia do que o causou. [Tente remover um ponto e vírgula ou uma chave do programa da Figura 2.1 e, então, recompile-o para ver as mensagens de erro geradas pela omissão.]

Capítulo 2 – Livro de Java 10ª Edição

2.3 Modificando nosso primeiro programa Java

Nesta seção, modificaremos o exemplo na Figura 2.1 para imprimir texto em uma linha utilizando várias instruções e imprimir texto em várias linhas utilizando uma única instrução.

Exibindo uma única linha de texto com múltiplas instruções

`Welcome to Java Programming!` pode ser exibido de várias maneiras. A classe `Welcome2`, mostrada na Figura 2.3, usa duas declarações (linhas 9 e 10) para produzir a saída mostrada na Figura 2.1. [*Observação*: deste ponto em diante, adotaremos um fundo amarelo para destacar os recursos novos e principais em cada listagem de código, como fizemos para as linhas 9 e 10.]

O programa é similar à Figura 2.1, portanto, discutiremos aqui somente as alterações.

```
// Imprimindo uma linha de texto com múltiplas instruções.
```

```
1  // Figura 2.3: Welcome2.java
2  // Imprimindo uma linha de texto com múltiplas instruções.
3
4  public class Welcome2
5  {
6     // método main inicia a execução do aplicativo Java
7     public static void main(String[] args)
8     {
9        System.out.print("Welcome to ");
10       System.out.println("Java Programming!");
11    } // fim do método main
12 } // fim da classe Welcome2
```

JAVA – EAGS - CAP 2022

Capítulo 2 – Livro de Java 10ª Edição

Exibindo múltiplas linhas de texto com uma única instrução

Uma única instrução pode exibir múltiplas linhas utilizando **caracteres de nova linha**, os quais indicam aos métodos `print` e `println` de `System.out` quando posicionar o cursor de saída no começo da próxima linha na janela de comando. Como ocorre com linhas em branco, caracteres de espaço em branco e caracteres de tabulação, os caracteres de nova linha são caracteres de espaço em branco. O programa na Figura 2.4 exibe quatro linhas de texto utilizando caracteres de nova linha para determinar quando iniciar cada nova linha. A maior parte do programa é idêntica àquelas nas figuras 2.1 e 2.3.

A linha 9

```
System.out.println("Welcome\nto\nJava\nProgramming!");
```

exibe quatro linhas de texto na janela de comando. Em geral, os caracteres em uma string são exibidos *exatamente* como aparecem entre as aspas duplas. Mas os caracteres emparelhados \ e n (repetidos três vezes na instrução) *não* aparecem na tela. A **barra invertida** (\) é um **caractere de escape**, que tem um significado especial para os métodos `print` e `println` de `System.out`. Quando aparece uma barra invertida em uma string, o Java a combina com o próximo caractere para formar uma **sequência de escape** — \n representa o caractere de nova linha. Quando um caractere de nova linha surge em uma string sendo enviada para saída com `System.out`, esse caractere de nova linha faz com que o cursor de saída na tela se mova para o começo da próxima linha na janela de comando.

A Figura 2.5 listas várias sequências de escape comuns e descreve como elas afetam a exibição de caracteres na janela de comando. Para obter a lista completa de sequências de escape, visite

```
http://docs.oracle.com/javase/specs/jls/se7/html/jls-3.html#jls-3.10.6
```

```
1  // Figura 2.4: Welcome3.java
2  // Imprimindo múltiplas linhas de texto com uma única instrução.
3
4  public class Welcome3
5  {
6     // método main inicia a execução do aplicativo Java
7     public static void main(String[] args)
8     {
9        System.out.println("Welcome\nto\nJava\nProgramming!");
10    } // fim do método main
11 } // fim da classe Welcome3
```

```
Welcome
to
Java
Programming!
```

JAVA – EAGS - CAP 2022

O uso da Classe Scanner:

```
1  // Figura 2.7: Addition.java
2  // Programa de adição que insere dois números, então exibe a soma deles.
3  import java.util.Scanner; // programa utiliza a classe Scanner
4
5  public class Addition
6  {
7     // método main inicia a execução do aplicativo Java
8     public static void main(String[] args)
9     {
10       // cria um Scanner para obter entrada a partir da janela de comando
11       Scanner input = new Scanner(System.in);
12
13       int number1; // primeiro número a somar
14       int number2; // segundo número a somar
15       int sum; // soma de number1 e number2
16
17       System.out.print("Enter first integer: "); // prompt
18       number1 = input.nextInt(); // lê primeiro o número fornecido pelo usuário
19
20       System.out.print("Enter second integer: "); // prompt
21       number2 = input.nextInt(); // lê o segundo número fornecido pelo usuário
22
23       sum = number1 + number2; // soma os números, depois armazena o total em sum
24
25       System.out.printf("Sum is %d%n", sum); // exibe a soma
26    } // fim do método main
27 } // fim da classe Addition
```

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

JAVA – EAGS - CAP 2022

Capítulo 2 – Livro de Java 10ª Edição

2.4 Exibindo texto com `printf`

O método `System.out.printf` (`f` significa "formato") exibe os dados *formatados*. A Figura 2.6 utiliza esse método para gerar a saída em duas linhas das strings "`Welcome to`" e "`Java Programming!`".

As linhas 9 e 10

```
System.out.printf("%s%n%s%n",
   "Welcome to", "Java Programming!");
```

chamam o método `System.out.printf` para exibir a saída do programa. A chamada de método especifica três argumentos. Quando um método exige múltiplos argumentos, estes são colocados em uma **lista separada por vírgulas**. Chamar um método também é referido como **invocar** um método.

```
1  // Figura 2.6: Welcome4.java
2  // Exibindo múltiplas linhas com o método System.out.printf.
3
4  public class Welcome4
5  {
6     // método main inicia a execução do aplicativo Java
7     public static void main(String[] args)
8     {
9        System.out.printf("%s%n%s%n",
10          "Welcome to", "Java Programming!");
11    } // fim do método main
12 } // fim da classe Welcome4
```

```
Welcome to
Java Programming!
```

Requisitos – Primeiro Programa

Algumas sequências de escape comuns:

Sequência de escape	Descrição
\n	Nova linha. Posiciona o cursor de tela no início da *próxima* linha.
\t	Tabulação horizontal. Move o cursor de tela para a próxima parada de tabulação.
\r	Retorno de carro. Posiciona o cursor da tela no início da linha *atual* — *não* avança para a próxima linha. Qualquer saída de caracteres depois do retorno de carro *sobrescreve* a saída de caracteres anteriormente gerada na linha atual.
\\	Barras invertidas. Utilizadas para imprimir um caractere de barra invertida.
\"	Aspas duplas. Utilizadas para imprimir um caractere de aspas duplas. Por exemplo, `System.out.println("\"entre aspas\"");` exibe `"entre aspas"`.

Capítulo 2 – Livro de Java 10ª Edição

O primeiro argumento do método `printf` é uma **string de formato** que pode consistir em **texto fixo** e **especificadores de formato**. A saída do texto fixo é gerada por `printf` exatamente como seria gerada por `print` ou `println`. Cada especificador de formato é um *marcador de lugar* para um valor e especifica o *tipo da saída de dados*. Especificadores de formato também podem incluir informações opcionais de formatação.

Especificadores de formato iniciam com um sinal de porcentagem (%) seguido por um caractere que representa o *tipo de dados*. Por exemplo, o especificador de formato `%s` é um marcador de lugar para uma string. A string de formato na linha 9 especifica que `printf` deve gerar a saída de duas strings, cada uma seguida por um caractere de nova linha. Na primeira posição do especificador de formato, `printf` substitui o valor do primeiro argumento depois da string de formato. Na posição do especificador de cada formato subsequente, `printf` substitui o valor do próximo argumento. Portanto, esse exemplo substitui "`Welcome to`" pelo primeiro `%s` e "`Java Programming!`" pelo segundo `%s`. O resultado mostra que duas linhas de texto são exibidas em duas linhas.

Observe que, em vez de usar a sequência de escape `\n`, utilizamos o especificador de formato `%n`, que é uma linha separadora *portável* entre diferentes sistemas operacionais. Você não pode usar `%n` no argumento para `System.out.print` ou `System.out.println`; mas o separador de linha gerado por `System.out.println` *depois* que ele exibe seu argumento *é* portável em diferentes sistemas operacionais. O Apêndice I (na Sala Virtual, em inglês) apresenta mais detalhes sobre a formatação de saída com `printf`.

Requisitos – Primeiro Programa

type	*code*	*typical literal*	*sample format strings*	*converted string values for output*
int	d	512	"%14d" "%-14d"	" 512" "512 "
double	f e	1595.1680010754388	"%14.2f" "%.7f" "%14.4e"	" 1595.17" "1595.1680011" " 1.5952e+03"
String	s	"Hello, World"	"%14s" "%-14s" "%-14.5s"	" Hello, World" "Hello, World " "Hello "
boolean	b	true	"%b"	"true"

Requisitos – Identificadores

Cada linguagem de programação tem seu próprio conjunto de regras e convenções para os tipos de nomes que você está autorizado a utilizar, e a linguagem de programação Java não é diferente.
As regras e convenções para nomear suas variáveis podem ser assim resumidas:

- Os nomes das variáveis distinguem letras maiúsculas e minúsculas.
- Um nome de variável pode ser qualquer identificador legal - uma seqüência de comprimento ilimitado-Unicode de letras e dígitos, começando com uma letra, o cifrão $ ou o caractere sublinhado _.
- A convenção, porém, é sempre começam seus nomes de variável com uma letra, e não $ ou _. Além disso, o caractere cifrão, por convenção, nunca é utilizado.
- Uma convenção semelhante existe para o carácter de sublinhado, é tecnicamente legal utilizá-lo para começar o nome da sua variável, contudo essa prática é desencorajada.
- Espaços em branco não são permitidos.
- Os caracteres subseqüentes podem ser letras, algarismos, sinais de dólar, ou caracteres de sublinhado. Tenha em mente que o o nome escolhido **não deve ser uma palavra reservada.**

Requisitos – Identificadores

Identificadores são os nomes que são associados às estruturas de um programa em Java, seja o nome de uma classe, variável ou método.
Vale lembrar que o Java é case-sensitive, ou seja, diferencia caracteres maiúsculos de minúsculos. Assim, um recurso do código cujo identificador é Teste é diferente de outro recurso cujo identificador é teste.

Existem inclusive regras de boas práticas a serem seguidas nas definições de letras maiúsculas e minúsculas dos identificadores de acordo com o Code Conventions da Sun (http://java.sun.com/docs/codeconv).

Basicamente, classes se iniciam com letras maiúsculas e o restante das letras minúsculas; se o identificador de classe for composto de mais de uma palavra, elas virão aglutinadas, cada uma delas também se inicia com maiúsculas. Como em: ExemploDeNomeDeClasse. Para método e variável (ou atributo) vale a mesma regra, sendo que a primeira palavra se inicia com letra minúscula. Exemplo:

exemploDeNomeDeMetodo( )
ExemploDeNomeDeVariavel

Existem algumas outras regras, por isso recomendamos a leitura do Code Conventions. Quanto ao universo que pode ser utilizado, os identificadores podem começar com qualquer letra, underscore (_) ou cifrão ($). Os caracteres subsequentes podem utilizar também números de 0 a 9.

Requisitos – Palavras Reservadas

Abstract	double	int	super
assert	else	interface	switch
boolean	enum	long	synchronized
break	extends	native	this
byte	final	new	throw
case	finally	package	throws
catch	float	private	transient
char	for	protected	try
class	if	public	void
const	goto	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp	

O Java também contém as palavras reservadas **true e false**, que são literais boolean e o literal **null** que representa **"referência nula".** Como as palavras-chave, essas palavras reservadas não podem ser usadas como identificadores.

Requisitos – Variáveis e suas classificações

Definição

Variáveis são espaços de memória utilizados para guardar dados de um determinado tipo.

Nomenclatura

O nome de uma variável pode ser qualquer identificador válido. Identificadores válidos podem conter somente:

- Letras (‘a’ a ‘z’ e ‘A’ a ‘Z’)
- Dígitos (0 a 9)
- Sublinhados (_)
- Sinais de Cifrão ($)

Cuidado!!!!
INICIAR um identificador com dígito(s) causa erro de compilação!

Requisitos – Variáveis e suas classificações

Classificação das variáveis

Variáveis podem receber até 4 classificações (ao mesmo tempo) de acordo com a sua visibilidade, escopo,constância e tipo de dado. Objetivamente, visibilidade está relacionado ao uso de modificadores de acesso; escopo ao contexto em que a variável foi definida; constância ao fato de uma variável ser ou não uma constante; e tipo de dado ao tipo de informação que a variável é capaz de armazenar.

Requisitos – Variáveis e suas classificações

Classificação das variáveis

Visibilidade (04)

- public
- private
- protected
- padrão ou default

MODIFICADOR	CLASSE	MESMO PACOTE	PACOTE DIFERENTE (SUBCLASSE)	PACOTE DIFERENTE(GLOBAL)
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

A visibilidade public
- Quem tem acesso à classe tem acesso também a qualquer membro com visibilidade public
- O alvo aqui é o programador cliente que usa suas classes
- É raro ter atributos públicos mas é comum ter métodos públicos

A visibilidade private
- O membro private não é acessível fora da classe
- A intenção aqui é permitir que apenas você que escreve a classe possa usar esse membro

A visibilidade protected
- O membro protected é acessível à classe e a suas subclasses
- A intenção é dar acesso ao programadores que estenderão sua classe

Requisitos – Variáveis e suas classificações

Classificação das variáveis

Escopo (03)

classe (static)
Atributos declarados com o modificador static. São acessíveis diretamente pela classe, sem a necessidade de instanciar um objeto. Espécie de variável “global” compartilhada entre os objetos da classe.

instância
Atributos declarados sem o modificador static. Cada instância de uma classe possui o seu próprio conjunto de variáveis de instância.

local
Variáveis declaradas dentro de um método ou bloco de código. São variáveis temporárias, válidas apenas dentro do método ou bloco onde foi declarada.

Requisitos – Variáveis e suas classificações

Classificação das variáveis

Escopo (03)

A linguagem de programação Java define os seguintes tipos de variáveis com relação ao escopo:

Variáveis de instância (Non-Static Fields) Tecnicamente falando, os objetos armazenam seus estados individuais em "campos não-estáticos", isto é, campos declarados sem a palavra-chave static Non-static fields também são conhecidos como *variáveis de instância,* porque seus valores são únicos para cada *instância* de uma classe.

Variáveis de classe (Static Fields) Uma variável de classe é qualquer campo declarado com o modificador static o que informa o compilador que existe apenas uma cópia dessa variável na existência, independentemente de quantas vezes a classe foi instanciada.

Variáveis locais Similar a um objeto que armazena seu estado em campos, um método, muitas vezes, armazena seu estado temporário em *variáveis locais.* A sintaxe para declarar uma variável local é semelhante à declaração de um campo Não há nenhuma palavra-chave especial que designa uma variável como local; essa determinação vem inteiramente a partir do local em que a variável é declarada - que está entre a abertura e fechamento de chaves de um método. Como tal, as variáveis locais são visíveis apenas para os métodos em que são declarados, não são acessíveis a partir do resto da classe.

Parâmetros A assinatura para o método main é public static void main(String[] args). Aqui, a variável args é o parâmetro para esse método. O importante a lembrar é que os parâmetros são sempre classificados como "variáveis" não "campos". Variáveis “parâmetro” aplicam-se a outras construções, tais como aos construtores e aos manipuladores de exceção.

Requisitos – Variáveis e suas classificações

Classificação das variáveis

Mutabilidade (02)

- **Constante (final)**

São variáveis não modificáveis após a sua inicialização, isto é, são constantes. Utiliza-se a palavra-chave final antes do tipo de dado da variável para especificar que ela é uma constante. Constantes podem ser inicializadas diretamente no momento da declaração ou por meio de construtores. Quando é inicializada diretamente na declaração, significa que o valor da constante será o mesmo para todos os objetos da classe.

- **Não constante**

O valor da variável pode ser modificado a qualquer momento.

Cuidado!!!!
Não inicializar uma constante OU tentar modificar o valor de uma constante já inicializada causa erro de compilação!

Requisitos – Variáveis e suas classificações

Classificação das variáveis

A linguagem de programação Java é **estaticamente tipada**, o que significa que todas as variáveis devem primeiramente ser declaradas, indicando seu tipo, antes de serem utilizadas.

Exemplo (Estaticamente tipada): int contador = 1;

int é o tipo inteiro da linguagem Java, o tipo deve ser especificado e a variável não poderá receber valores diferentes tipo do inteiro.

Exemplo (Dinamicamente tipada – na linguagem PHP)

$contador = 1; O tipo da variável não é especificado.
$contador = “Alo Mundo” (Conteúdo da variável pode ser modificado, durante o código)

Requisitos – Variáveis e suas classificações

Classificação das variáveis

Tipo de dado (02)

- **primitiva**

Quando o tipo de dados da variável é um tipo primitivo. Guarda um valor compatível com o tipo primitivo declarado.

- **de referência**

Variável de referência ou de instância: quando o tipo de dados da variável é uma classe. Guarda uma referência para um objeto da classe declarada.

Uma variável contém um identificador e um tipo, os dois de acordo com as definições que você já estudou. Isso é o que chamamos de declaração de uma variável.

Além da declaração, uma variável também pode ser inicializada por meio do operador de atribuição “=”.

As declarações de variáveis do mesmo tipo podem ser feitas todas na mesma linha, mas essa é considerada uma má prática, pois deixa o código ilegível, confuso e desorganizado.

Requisitos – Variáveis e suas classificações

Classificação das variáveis

Tipo de dado (02)

- **primitiva**

Quando o tipo de dados da variável é um tipo primitivo. Guarda um valor compatível com o tipo primitivo declarado.

- **de referência**

Variável de referência ou de instância: quando o tipo de dados da variável é uma classe. Guarda uma referência para um objeto da classe declarada.

Uma variável contém um identificador e um tipo, os dois de acordo com as definições que você já estudou. Isso é o que chamamos de declaração de uma variável.

Além da declaração, uma variável também pode ser inicializada por meio do operador de atribuição “=”.

As declarações de variáveis do mesmo tipo podem ser feitas todas na mesma linha, mas essa é considerada uma má prática, pois deixa o código ilegível, confuso e desorganizado.

Acesso a variável da Classe

```
1     public class Teste1 {
2
3     int x=0;
4
5
6      public static void main(String[ ] args) {
7
8        System.out.println(x);
9          }
10  }
```

Acesso a variável da Classe – usando STATIC

```
1     public class Teste1 {
2
3     static int x=0;
4
5
6      public static void main(String[ ] args) {
7
8      System.out.println(x);
9        }
10  }
```

Acesso a variável da Classe – usando INSTÂNCIA

```
1     public class Teste1 {
2
3     int x=0;
4
5
6      public static void main(String[ ] args) {
7      Teste1 teste1=new Teste1();
8      System.out.println(teste1.x);
9        }
10  }
```

Inicialização de variáveis

```
1     public class Teste1 {
2
3     static int x;
4
6      public static void main(String[ ] args) {
7       int y;
8    System.out.println(y);
9    System.out.println(x);
10         }
11  }
```

Requisitos – Tipos Primitivos

Tipos Primitivos

Existem oito tipos primitivos de dados suportados pela linguagem de programação Java:

Tipo	Valores	Tamanho em bits	Padrão
boolean	true ou false		
char	‘\u0000’ a ‘\uFFFF’ (0 a 65535)	16	(conjunto de caracteres unicode)
byte	–128 a +127	8	
short	–32768 a +32767	16	
Int	–2147483648 a +2147483647	32	
long	-2^{63} a $2^{63}-1$	64	
float	$1.4e^{-45}$ to $3.4e^{+38}$	32	(ponto flutuante IEEE 754)
double	$5e^{-324}$ to $1.8e^{+308}$	64	(ponto flutuante IEEE 754)

Requisitos – Tipos Referência

Tipos Referência

Os tipos referência guardam o endereço para os objetos e funcionam como um mecanismo de acesso aqueles objetos armazenados na memória. A alocação de memória é irrelevante aos programadores Java devido ao total gerenciamento executado pela JVM. Todos os tipos referência são subclasses do tipo java.lang.Object.

Tipos Referência	Tipos primitivos
Número **ilimitados** de tipos, pois podem ser definidos pelo programador.	Compostos pelos booleanos e os numéricos: **char, byte, short, int, long, float, and double.**
Armazena a **endereço de memória** para o dado.	Local na memória armazena o **valor real** armazenado pelo tipo primitivo.
Quando um tipo referência é atribuído por outro tipo referência, **ambos irão apontar** para o mesmo objeto.	Quando um o valor de tipo primitivo é atribuído para outra variável de mesmo tipo, uma **cópia é realizada**.
Quando um método recebe um objeto por parâmetro, este método pode **alterar o conteúdo do objeto** recebido, **mas não o endereço** do objeto.	Quando um tipo primitivo é recebido pelo parâmetro de um método, somente uma **cópia do primitivo é passada**. Os métodos chamados não acessam o valor primitivo original e consequentemente não podem alterá-lo. O método chamado pode **alterar apenas o valor copiado**.

Requisitos – Classes Empacotadoras

Classes Empacotadoras

Cada tipo primitivo possui uma classe / tipo referência wrapper correspondente, a qual é localizada no pacote java.lang. Cada classe empacotadora possui uma variedade de métodos, incluindo um para retornar o tipo valor (primitivo) correspondente.

Tipo primitivo	Tipo referência	Métodos para retornar valores primitivos
boolean	Boolean	booleanValue( )
char	Character	charValue( )
byte	Byte	byteValue( )
short	Short	shortValue( )
int	Integer	intValue( )
long	Long	longValue( )
float	Float	floatValue( )
double	Double	doubleValue( )

Requisitos – Valores Padrão

Valores padrão

Não é obrigatório atribuir um valor quando um campo é declarado. Os campos que são declarados, mas não inicializados serão definidos com seu respectivo valor padrão pelo compilador. De um modo geral, este padrão será zero ou null dependendo do tipo de dado.

O quadro abaixo resume os valores padrão para os tipos de dados acima.

Tipo de dados	Valor padrão (para os campos)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\ u0000'
String (ou qualquer objeto)	null
boolean	false

O comportamento para as **variáveis locais é diferente**, **nunca o compilador atribui um valor padrão para uma variável local não inicializada**. Se você não puder inicializar a sua variável local onde ela é declarada, certifique-se de atribuir um valor a ela antes de tentar usá-lo. **Acessar a uma variável local não inicializada irá resultar em um erro tempo de compilação.**

Requisitos – Literais

Literais

Você deve ter notado que a palavra-chave new não é utilizada quando inicializamos uma variável de um tipo primitivo. Tipos primitivos são tipos especiais de dados incorporadas a linguagem, não são objetos criados a partir de uma classe. Um *literal* é a representação de código-fonte de um valor fixo; literais são representados diretamente em seu código sem a necessidade de computação. Como mostrado abaixo, é possível atribuir um literal para uma variável de um tipo primitivo:

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

Os tipos inteiros byte short int e long podem ser expressos usando os sistemas de numeração: decimal, octal ou hexadecimal.

```
int decVal = 26;  // O número 26, em decimal
int octVal = 032;  // O número 26, em octal
int hexVal = 0x1a; // O número 26, em hexadecimal
```

Os tipos de ponto flutuante float e double também podem ser expressos usando E ou e (para notação científica), F ou f (32-bit float literal) e D ou d (64-bit duplo literal, este é o padrão e por convenção é omitido).

```
double d1 = 123.4;
double d2 = 1.234e2; // Valor que d1, mas em notação científica
float f1  = 123.4f;
```

Requisitos – Literais

Literais

A linguagem de programação Java também suporta algumas sequências de escape especiais para char e literais String \b (backspace), \t (tab), \n (alimentação de linha), \f (alimentação do formulário), \r (retorno de carro), \" (aspas duplas), e \\ (barra invertida).
Há também um literal especial null que pode ser usado como um valor para qualquer tipo de referência. null pode ser atribuído a qualquer variável, exceto as variáveis de tipos primitivos. Pouco você pode fazer com um valor null, além de testar a sua presença. Contudo, null é freqüentemente usada em programas como um marcador para indicar que algum objeto não está disponível.
Por fim, há também um tipo especial chamado literal class, sendo formado por um nome do tipo e acrescentando-se (.class). Por exemplo: String.class, refere-se ao objeto (do tipo class) que representa o próprio tipo.

Requisitos – Operadores

Operadores

Os operadores desempenham **operações com um, dois, ou mais operandos** e retornam um resultado. Os tipos de operadores em Java incluem os de **atribuição, aritméticos, comparação, orientados a bit (bitwise), incremento / decremento e classes/objetos**.

Operador	Descrição	Associatividade
++	pós-incremento unário	da direita para a esquerda
--	pós-decremento unário	
++	pré-incremento unário	da direita para a esquerda
--	pré-decremento unário	
+	mais unário	
-	menos unário	
!	negação lógica unária	
~	complemento unário sobre bits	
(*tipo*)	coerção unária	
*	multiplicação	da esquerda para a direita
/	divisão	
%	módulo	
+	adição ou concatenação de string	da esquerda para a direita
-	subtração	
<<	deslocamento de bits para a esquerda	da esquerda para a direita
>>	deslocamento de bits para a direita com sinal	
>>>	deslocamento de bits para a direita sem sinal	
<	relacional menor que	da esquerda para a direita
<=	relacional menor que ou igual a	
>	relacional maior que	
>=	relacional maior que ou igual a	
instanceof	comparação de tipo	

Requisitos – Operadores

Operadores

== !=	relacional é igual a relacional não é igual a	da esquerda para a direita
&	E sobre bits E lógico booleano	da esquerda para a direita
^	OU exclusivo sobre bits OU lógico booleano exclusivo	da esquerda para a direita
|	OU inclusivo sobre bits OU inclusivo lógico booleano	da esquerda para a direita
&&	E condicional	da esquerda para a direita
||	OU condicional	da esquerda para a direita
?:	condicional	da direita para a esquerda

Requisitos – Operadores

Operadores

=	atribuição	da direita para a esquerda
+=	atribuição de adição	
-=	atribuição de subtração	
*=	atribuição de multiplicação	
/=	atribuição de divisão	
%=	atribuição de resto	
&=	atribuição E sobre bits	
^=	atribuição OU exclusiva sobre bits	
|=	atribuição OU inclusiva sobre bits	
<<=	atribuição de deslocamento para a esquerda de bits	
>>=	atribuição de bits com deslocamento para a direita com sinal	
>>>=	atribuição de bits com deslocamento para a direita com sinal	

Requisitos – Comparando tipos referência

Usando os operadores de iguadade (== e !=)

São usados para comparar a localização de dois objetos na memória. Se os endereços de memória dos objetos são os mesmos, eles são considerados iguais.

Não são utilizados para comparar o conteúdo dos objetos.
No exemplo seguinte, frase1 e frase2 tem o mesmo endereço de memória, então “Eles são iguais” será impresso na saída.

```
String frase1 = new String("ADONAI");
        String frase2 = frase1;
        if (frase1 == frase2)
                System.out.println("Eles são iguais");
```

No exemplo seguinte, os endereços de memória não são iguais, então “Eles não são iguais” será impresso na saída.

```
String frase3 = new String("ADONAI");
        String frase4 = new String("ADONAI");
        if (frase3 == frase4)
          System.out.println("Eles são iguais.");
        else
          System.out.println("Eles não são iguais");
```

Requisitos – Comparando tipos referência

Usando o método equals()

Para comparar o conteúdo de duas objetos de classe, o método equals() oriundo da classe Object pode ser utilizado ou sobrescrito.

Por padrão, o método equals() original da classe Object, usa somemte o operador de comparação ==.

Este método deve ser sobrescrito para ser realmente útil.

Por exemplo, se você quiser comparar valores contidos em duas instâncias de uma mesma classe, você deveria usar o método equals() definido pelo programador.

Requisitos – Comparando Strings

Existem dois modos de verificar se Strings são iguais em Java, mas a definição de “igual” para cada uma deles é diferente. Tipicamente, se o desejo é comparar a sequência de caracteres contidos em duas Strings o método **equals()** deve ser utilizado.

Exemplo:

```
class Comparacoes {

 // Adicionando a string ao pool
 String primeira = "abcde";

 // Usando a string do pool
 String segunda = "abcde";

 // Criando uma nova string
 String terceira = new String ("abcde");

 void metodo( ) {

 // Avaliada como true
 if (primeira == segunda) {
  System.out.println("primeira == segunda");
 }

 // Avaliada como true
 if (primeira.equals(segunda)) {
  System.out.println("primeira equals segunda");
 }
 // Avaliada como false
 if (primeira == terceira) {
  System.out.println("primeira == terceira");
 }
 // Avaliada como true
 if (primeira.equals(terceira)) {
  System.out.println("primeira equals terceira");
 }
}      }
```

Requisitos – Comparando Strings

Existem dois modos de verificar se Strings são iguais em Java, mas a definição de “igual” para cada uma deles é diferente. Tipicamente, se o desejo é comparar a sequência de caracteres contidos em duas Strings o método **equals()** deve ser utilizado.

Exemplo:

```
class Comparacoes {

 // Adicionando a string ao pool
 String primeira = "abcde";

 // Usando a string do pool
 String segunda = "abcde";

 // Criando uma nova string
 String terceira = new String ("abcde");

 void metodo( ) {

 // Avaliada como true
 if (primeira == segunda) {
  System.out.println("primeira == segunda");
 }

 // Avaliada como true
 if (primeira.equals(segunda)) {
  System.out.println("primeira equals segunda");
 }
 // Avaliada como false
 if (primeira == terceira) {
  System.out.println("primeira == terceira");
 }
 // Avaliada como true
 if (primeira.equals(terceira)) {
  System.out.println("primeira equals terceira");
 }
 }      }
```

Operadores do Java

Operador de igualdade ou relacional algébrico padrão	Operador de igualdade ou relacional Java	Exemplo de condição em Java	Significado da condição em Java
Operadores de igualdade			
=	==	x == y	x é igual a y
≠	!=	x != y	x é diferente de y
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior que ou igual a y
≤	<=	x <= y	x é menor que ou igual a y

Figura 2.14 | Operadores de igualdade e operadores relacionais.

Operadores	Associatividade	Tipo
* / %	da esquerda para a direita	multiplicativo
+ -	da esquerda para a direita	aditivo
=	da esquerda para a direita	relacional
== !=	da esquerda para a direita	igualdade
=	da direita para a esquerda	atribuição

Figura 2.16 | Operadores de precedência e de associatividade discutidos.

JAVA – EAGS - CAP 2022

Lista de operadores do Java

OPERADOR	FUNÇÃO	OPERADOR	FUNÇÃO
+	Adição	~	Complemento
-	Subtração	<<	Deslocamento à esquerda
*	Multiplicação	>>	Deslocamento à direita
/	Divisão	>>>	Desloc. a direita com zeros
%	Resto	=	Atribuição
++	Incremento	+=	Atribuição com adição
--	Decremento	-=	Atribuição com subtração
>	Maior que	*=	Atribuição com multiplicação
>=	Maior ou igual	/=	Atribuição com divisão
<	Menor que	%=	Atribuição com resto
<=	Menor ou igual	&=	Atribuição com AND
==	Igual	|=	Atribuição com OR
!=	Não igual	^=	Atribuição com XOR
!	NÃO lógico	<<=	Atribuição com desl. esquerdo
&&	E lógico	>>=	Atribuição com desloc. direito
||	OU lógico	>>>=	Atrib. C/ desloc. a dir. c/ zeros
&	AND	? :	Operador ternário
^	XOR	*(tipo)*	Conversão de tipos (cast)
|	OR	instanceof	Comparação de tipos

Exercícios

1.2 Preencha as lacunas em cada uma das seguintes frases sobre o ambiente Java:

a) O comando _____ do JDK executa um aplicativo Java.

b) O comando _____ do JDK compila um programa Java.

c) Um arquivo de programa Java deve terminar com a extensão de arquivo _____.

d) Quando um programa Java é compilado, o arquivo produzido pelo compilador termina com a extensão de arquivo _____.

e) O arquivo produzido pelo compilador Java contém _____ que são executados pela Java Virtual Machine.

JAVA – EAGS - CAP 2022

Questão 39 – CAP 2009 – Prova Amarela

39) Em relação à linguagem de programação Java, complete corretamente as lacunas das sentenças abaixo, e assinale a opção corrtea.

I - O comando _____ do J2SE Development Kit executa um aplicativo Java.
II - O comando _____ do J2SE Development Kit compila um programa Java.
III- Um arquivo de programa Java deve terminar com a extensão de arquivo _____.
IV - Quando um programa Java é compilado, o arquivo produzido pelo compilador termina com a extensão de arquivo _____.
V - O arquivo produzido pelo compilador Java contém _____ que são executados pela Java Virtual Machine.

(A) java / javac / java / class / bytecodes
(B) javac / java / class / java / bytecodes
(C) main / javac / java / jar / class
(D) java / class / rar / jar / javac
(E) rar / jar / java / javac / class

- Operadores aritméticos e Lógicos

Operação Java	Operador	Expressão algébrica	Expressão Java
Adição	+	$f + 7$	`f + 7`
Subtração	-	$p - c$	`p - c`
Multiplicação	*	bm	`b * m`
Divisão	/	x/y ou $\frac{x}{y}$ ou $x \div y$	`x / y`
Resto	%	$r \bmod s$	`r % s`

Figura 2.11 | Operadores aritméticos.

Regras de precedência de operador

O Java aplica os operadores em expressões aritméticas em uma sequência precisa determinada pelas seguintes **regras de precedência de operador**, que são geralmente as mesmas que aquelas seguidas em álgebra:

1. As operações de multiplicação, divisão e de resto são aplicadas primeiro. Se uma expressão contiver várias dessas operações, elas serão aplicadas da esquerda para a direita. Os operadores de multiplicação, divisão e resto têm o mesmo nível de precedência.
2. As operações de adição e subtração são aplicadas em seguida. Se uma expressão contiver várias dessas operações, os operadores serão aplicados da esquerda para a direita. Os operadores de adição e subtração têm o mesmo nível de precedência.

Essas regras permitem que o Java aplique os operadores na *ordem* correta.[1] Quando dizemos que operadores são aplicados da esquerda para a direita, estamos nos referindo à sua **associatividade**. Alguns operadores associam da direita para a esquerda. A Figura 2.12 resume essas regras de precedência de operador. Um gráfico completo de precedência está incluído no Apêndice A.

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

Figura 2.12 | Precedência de operadores aritméticos.

Operador algébrico	Operador de igualdade ou relacional Java	Exemplo de condição em Java	Significado da condição em Java
Operadores de igualdade			
=	==	x == y	x é igual a y
≠	!=	x != y	x é não igual a y
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior que ou igual a y
≤	<=	x <= y	x é menor que ou igual a y

Figura 2.14 | Operadores de igualdade e operadores relacionais.

Operadores	Associatividade	Tipo
* / %	da esquerda para a direita	multiplicativo
+ -	da esquerda para a direita	aditivo
< <= > >=	da esquerda para a direita	relacional
== !=	da esquerda para a direita	igualdade
=	da direita para a esquerda	atribuição

Figura 2.16 | Operadores de precedência e de associatividade discutidos.