Fundamentos da Programação Orientada a Objetos

UNIDADE 01

Dados e Variáveis

Conhecendo o Java

Olá, bem-vindo à **programação orientada a objetos** com **Java**!

Você sabe o que é o Java, afinal? Ele é uma poderosa linguagem de programação, lançada pela Sun Microsystems, em 1995. Desde então, se mantém como uma das principais e mais populares linguagens de desenvolvimento (TIOBE, 2020), muito utilizada em servidores de internet e celulares Android, pois está evoluindo e se redefinindo constantemente (GODOY, 2019).

Aprender a **orientação a objetos** utilizando o Java é uma ótima forma de conhecer todas as vantagens e recursos que esse tipo de programação oferece. Com esse objetivo, vamos nos concentrar em entender como utilizar a linguagem Java, nesta **Unidade 1**, para então explorarmos os conceitos da orientação a objetos nas próximas Unidades.

Iniciamos com a apresentação dos conceitos básicos de preparação à programação, mostrando como eles são usados com a plataforma Java. Com isso, estaremos prontos para instalar um ambiente de desenvolvimento apropriado ao Java, que permitirá criar nossos primeiros programas. Em seguida, vamos praticar programação com Java, explorando sua sintaxe e conceitos. Assim, estaremos prontos para realizar operações de leitura e escrita de dados, utilizar comandos de desvio e repetições e trabalhar com estrutura de dados como vetores e matrizes. Pronto para começar? Vamos lá!

Conceitos básicos de preparação à programação

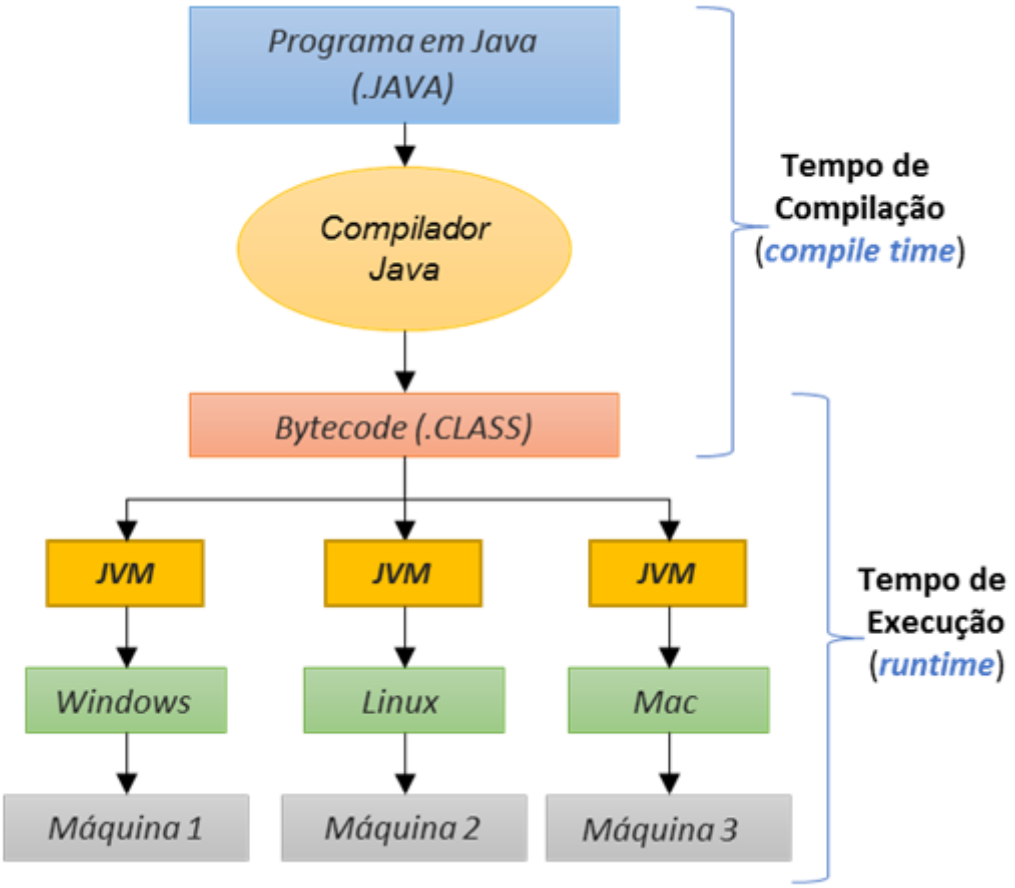
Compilador

É um programa que transforma um código escrito em uma **linguagem de programaçã**o, que é o código-fonte (***source code***), em um programa equivalente em outra linguagem, é o código-objeto (***object code***), geralmente em **linguagem de**

máquina (binária). Essa transformação permite que o trabalho de desenvolvimento ocorra em dois momentos distintos. Confira na Figura 1 a seguir esses dois momentos.

- **Tempo de compilação** (*compile time*): envolve os eventos que ocorrem **durante o processo de compilação**. Tem como objetivo transformar o código-fonte escrito em uma linguagem de programação em outra linguagem, como a linguagem de máquina (binária).
- **Tempo de execução** (*runtime*): envolve os eventos que ocorrem **durante a execução do programa.**

Figura 1 – Estrutura de funcionamento da JVM



A JVM é um programa executável que carrega e executa os aplicativos Java (bytecode `.class`), convertendo-os em código executável para uma determinada máquina. Fonte: Autores (2020).

Desenvolvendo programas

Antes de compilar, precisamos **escrever** nosso código-fonte, o que pode ser feito em um editor de texto como um bloco de notas. Após a edição do código em uma linguagem de programação, ele poderá ser **compilado** e **executado**. Contudo, se existirem erros de digitação ou comandos incorretos, o código-fonte não compila nem

executa. Apenas com um editor como o bloco de notas, o programador terá que, manualmente, encontrar e acertar os erros, que é uma forma de trabalho lenta e pouco produtiva.

Para facilitar o trabalho de encontrar os erros no código, realizar a compilação e execução de programas, existem os **ambientes de desenvolvimento integrados**, mais frequentemente referenciados pela sua sigla em inglês: **Integrated Develoment Environment** (IDE).

Os IDEs fornecem ferramentas para análise de código, escrita de comandos que se autocompletam (autocomplete), colorização de comandos, chamada do compilador e execução de programas, dentre outras funcionalidades, que auxiliam o programador a escrever códigos melhores e sem erros de compilação, de forma mais rápida e segura. Para o Java, existem três IDEs gratuitos e poderosos (GODOY, 2019):

- Netbeans IDE, da Apache Software Foundation (https://netbeans.apache.org/download/index.html).
- Eclipse, da Eclipse Foundation (https://www.eclipse.org/downloads/).
- IntelliJ, da JetBrains (https://www.jetbrains.com/pt-br/idea/download/).

> Na seção **ambiente de desenvolvimento** você encontrará mais orientações para a **instalação de um IDE**.

Plataforma Java

O Java não é apenas uma linguagem: é uma plataforma que inclui algumas características muito especiais, listadas na Tabela 1.

Tabela 1 - Principais características do Java

Característica	Descrição
Orientada a objetos	Java incorpora a moderna filosofia de programação orientada a objetos.
Neutra em relação à arquitetura	Não possui vínculo com determinado equipamento ou arquitetura de sistema operacional.
Portabilidade	O Java possui Independência de plataforma: "write once run anywhere!" (escreva uma vez, execute em qualquer lugar), pois o código compilado (bytecode) é portável.
Bytecode	Código compilado em um conjunto de instruções otimizadas que devem ser interpretadas em um sistema de runtime (tempo de execução) do Java, como a JVM – ver Figura 2.
JVM	A máquina virtual Java (JVM, ou Java Virtual Machine) é um programa que carrega, interpreta e executa programas Java, convertendo seus bytecode em código executável de máquina. O programa da JVM é feito para um determinado sistema operacional (Windows, Linux, Mac etc.) – ver Figura 2.
Compilada e Interpretada	Programas (arquivos `.java`) são compilados para o formato bytecode (arquivos `.class`) que são interpretados pela JVM – ver Figura 2.
APIs	Aplication Program Interfaces, ou bibliotecas de classes, que facilitam o trabalho com arquivos, interface gráfica, redes entre outras funcionalidades.
Ferramentas e utilidades	Conjunto de documentação, analisador de performance e empacotador de arquivos.

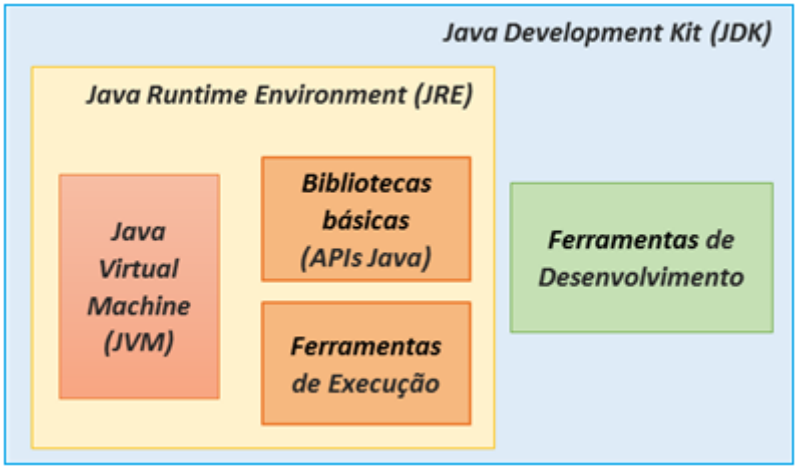
Fonte: GODOY, 2019, p.10.

A característica de **portabilidade** é um dos principais motivos de sucesso do Java há mais de 20 anos, pois permite que o programador escreva o código uma vez e execute em qualquer sistema operacional. Por essa razão, o Java é considerado **multiplataforma**, ou seja, **neutro** em relação à arquitetura do sistema operacional.

Para viabilizar a portabilidade, a plataforma Java é distribuída em dois pacotes (GODOY, 2019). Veja as diferenças na Figura 2.

- **Java Runtime Environment** (**JRE**): **é o pacote de execução**. Contém apenas a JVM (no Windows, é o executável `java.exe`) e o código compilado das bibliotecas-padrão (por exemplo, o pacote `java.util`, que tem a classe *`string`*). Qualquer pessoa interessada em executar um programa Java, deverá ter a JRE instalada em sua máquina.
- **Java Development Kit** (**JDK**): **é o pacote de desenvolvimento**. Contém o JRE mais o compilador Java (no Windows, o executável **`javac.exe`**), códigos-fonte, documentação e ferramentas. Deve ser instalado pelo desenvolvedor para criar projetos.

Figura 2 – Diferença entre os pacotes JDK e JRE



O JRE (ambiente de execução JAVA) faz parte do JDK (kit de desenvolvimento Java). Se um equipamento apenas executa programas Java, então que ele tenha instalado o JRE. Fonte: Java Development Kit (JKD). **Dicas de Java**. Disponível em: http://dicasdejava.com.br.

Ambiente de desenvolvimento

Como já comentado, o ambiente de desenvolvimento é chamado de IDE. No caso do Java, o IDE também precisa que o JDK já esteja instalado na máquina de trabalho. Juntos, JDK e IDE fornecem um ambiente adequado para desenvolvimento de programas em Java.

Instalação do JDK

Você vai precisar instalar o kit de desenvolvimento Java para criar os programas desta disciplina. Para ajuda com a instalação, siga as orientações deste tutorial passo a passo.

Instalação do IDE

As práticas de todas as Unidades desta disciplina podem ser realizadas tanto com o **Eclipse** quanto com o **IntelliJ**. Veja como realizar as instalações dos IDEs sugeridos – **só é preciso escolher um deles**.

Com o JDK já instalado, escolha um dos IDEs para completar o seu ambiente de desenvolvimento para Java.

Tutorial de instalação de Eclipse IDE

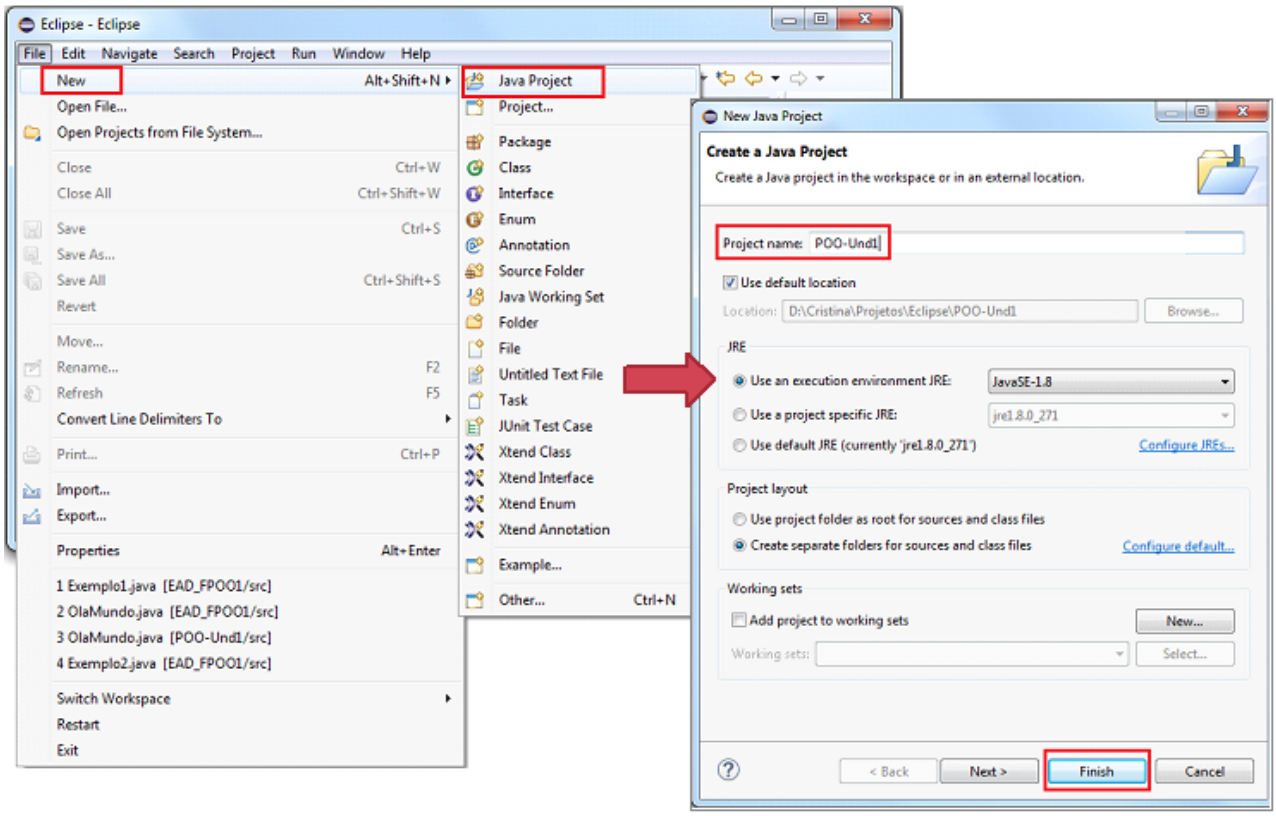
Tutorial de instalação de Intellij IDEA

Meu primeiro programa Java

Muito bem, com um IDE instalado e funcional, vamos começar o nosso primeiro programa. Siga as orientações das figuras a seguir e verifique o funcionamento do programa **OlaMundo**.

- Abra o IDE (Eclipse, no exemplo) e crie um projeto vazio.
- Digite o nome do projeto **POO-Und1** no campo indicado e clique em "***Finish***"

Figura 3 – Criação de projeto no Eclipse



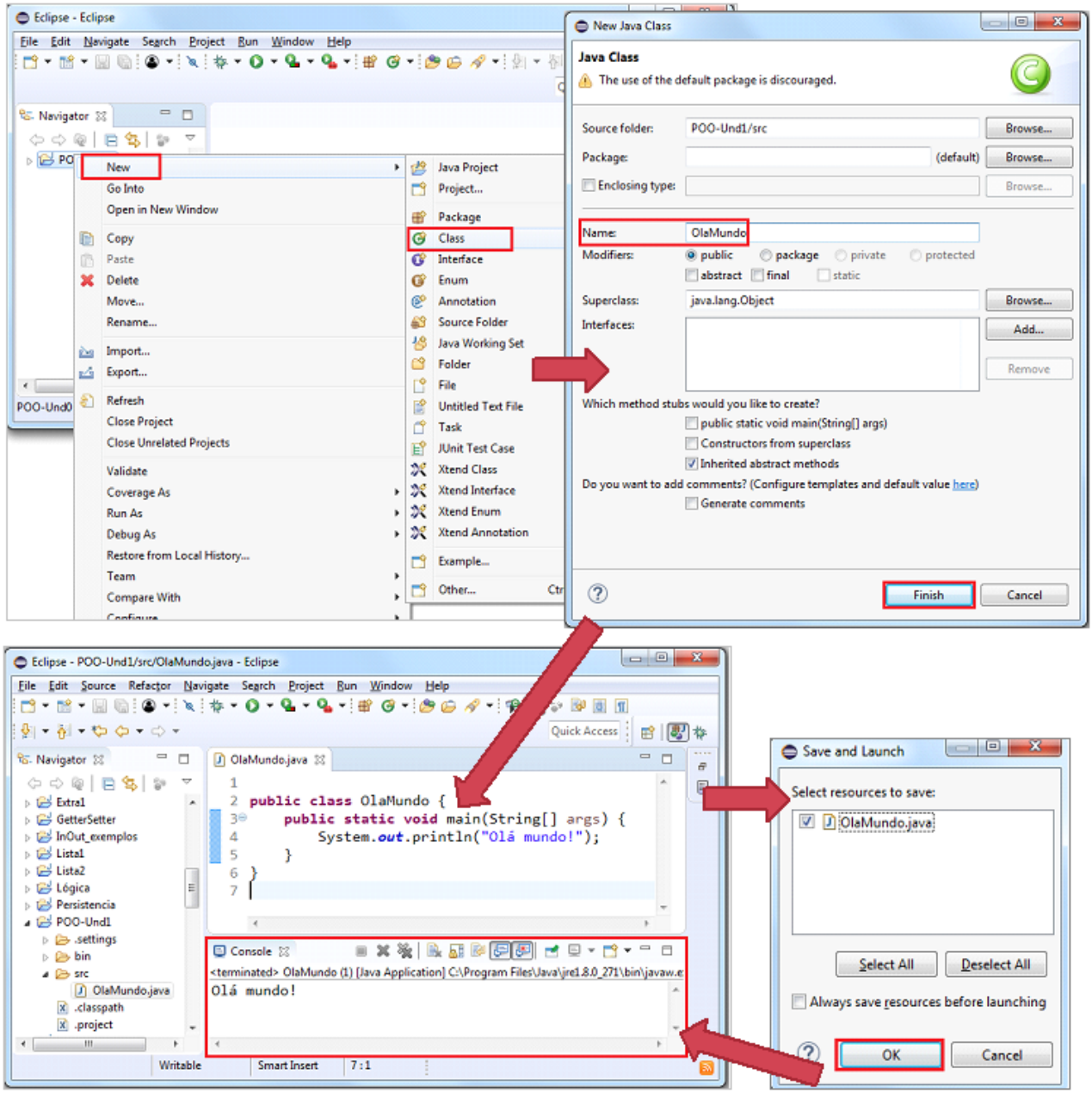
Fonte: Autores (2020).

Observe na Figura 4 a seguir, que no Java tudo se relaciona a **objetos** e **classes**. Seu programa precisou de uma classe, a **OlaMundo**, para poder executar. Na linha 4, precisamos criar um método (função) importante, conhecida como função principal (`main`). Ele sempre será o **ponto inicial de execução** dos nossos programas. Observe também que classe e função principal delimitam seus blocos de código com os caracteres `{` e `}`. Por fim, para conseguirmos imprimir nosso "**Olá mundo**" no *console*, utilizamos o comando **`System.out.println`.** Logo após, nosso programa terminou de executar.

Vamos explorar melhor os conceitos **classe** e **objetos** nas próximas Unidades, mas antes veremos maiores detalhes da **sintaxe** e **conceitos da linguagem Java**, nas seções a seguir.

1. Com o botão direito do mouse sobre o projeto recém-criado, acesse a opção "***New***" e após, "***Class***".
2. Digite o nome da classe "**OlaMundo**" no campo indicado e clique em "***Finish***".
3. Edite o código-fonte como indicado e execute seu programa na opção "***Run***" de menu, ou no botão "**avançar**".
4. Aceite a opção para salvar e executar o seu programa – janela "***Save and Launch***" – , clicando "**Ok**".
5. Veja a saída do programa na janela "**Console**", abaixo do editor de código.

Figura 4 – Criação de classe, edição e execução de programa



Fonte: Autores (2020).

Sintaxe e conceitos do Java

A linguagem Java herdou sua sintaxe da linguagem C, e o seu modelo de objetos é baseado no modelo do C++. Essa relação existe porque C e C++ , sempre muito populares, iniciaram a era moderna da programação. Assim, ao se basear no legado dessas duas linguagens, o Java facilita sua aprendizagem. Juntas, as linguagens C, C++ e Java definem uma estrutura conceitual comum para o programador profissional, que encontra muitas semelhanças de sintaxe e conceitos quando passam de uma linguagem para outra (SCHIELDT, 2015).

Tipos de dados no Java

O Java é uma linguagem **fortemente tipada**. Isso significa que **todas** as variáveis têm **compatibilidade de tipos** verificada pelo compilador. Logo, quando não há compatibilidade, as operações são consideradas um **erro no programa.** Essa verificação rígida diminui ocorrência de problemas no código e melhora a confiabilidade do código. Por essa razão, todas as variáveis, expressões e valores precisam ter um **tipo de dado.**

O Java tem **tipos de dados orientados a objeto** (que veremos nas próximas Unidades) e **tipos de dados primitivo** (elementares, têm valores binários comuns) – esses últimos são mostrados na Tabela 2.

Tabela 2 – Tipos de dados primitivos do Java

Tipo	Significado	Tamanho em bits	Intervalo
`boolean`	Verdadeiro ou falso	8 bits ou 1 byte	`True` ou `false`
`char`	Caractere (letra ou símbolo)	16 bits ou 2 bytes	`\u0000` a `\uffff`
`float`	Ponto flutuante de precisão simples	32 bits ou 4 bytes	Aproximadamente ±3.40282347E+38 (6 a 7 dígitos decimais significativos)
`double`	Ponto flutuante de precisão dupla	64 bits ou 8 bytes	Aproximadamente ±1.79769313486231570E+308 (15 dígitos decimais significativos)
`byte`	Inteiro	8 bits ou 1 byte	–128 a 127
`short`	Inteiro curto	16 bits ou 2 bytes	–32.768 a 32.767
`int`	Inteiro	32 bits ou 4 bytes	–2.147.483.648 a 2.147.483.647
`long`	Inteiro longo	64 bits ou 8 bytes	–9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Fonte: HORSTMANN; CORNELL, 2010, p. 21.

Um cuidado especial deve ser feito ao **representar números** no Java. Para evitar enganos, devem ser usadas letras especiais, como apresentado na Tabela 3.

Tabela 3 – Sintaxe para tipos de dados com ponto flutuante

Código	Regra	Exemplo	Observações
`float`	Números `float` terminam com `f` ou `F`	`float a = 7.1f;` `float b = 3.1415F;`	Um número com ponto flutuante e sem letra no final é `double` por padrão no Java. Apenas para identificação de tipo, maiúsculas e minúsculas são equivalentes.
`double`	Números `double` terminam com `d` ou `D`	`double c = 7.1d;` `double d = 3.1415D;`	
`long`	Números `long` terminam com `l` ou `L`	`long e = 12l;` `long f = 25L;`	

`int`	Parar representar números em diferentes sistemas de numeração	`int decVal = 26; // 26 em decimal`
		`int hexVal = 0x1A; // 26 em hexadecimal`
		`int binVal = 0b11010; // 26 em binário`	

Fonte: Autores (2020).

O Java também define alguns caracteres especiais na escrita de **textos**, como apresentado na Tabela 4, com exemplos na Figura 5.

Tabela 4 – Código Java para caracteres especiais

Código	Significado
`\b`	*Backspace* (volta um caracter)
`\f`	*Form feed* (nova página)
`\n`	*New line* (nova linha)
`\r`	Carriage *return* (retorno de carro)
`\t`	*Tab* (espaço de tabulação)
`\"`	Escrita do caractere de aspas duplas
`\'`	Escrita do caractere de aspas simples ou apóstrofe
`\\`	Escrita de barra invertida
`\uXXXX`	Escrita de códigos em hexadecimal são precedidos por `\u`

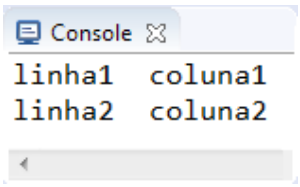
Fonte: JAVA TUTORIALS, 2020.

Figura 5 – Exemplo de código que utiliza caracteres especiais

</>

```
public class OlaMundo {
   public static void main(String[] args) {

System.out.println("linha1\tcoluna1\nlinha2
\tcoluna2");
   }
}
```



Fonte: Autores (2020).

Variáveis

No Java, declaramos uma variável informando seu **tipo de dados**, seguido do seu **nome**. Veja também alguns exemplos na Figura 6.

```
tipo nomeDaVariavel;
tipo nomeDaVariavel = valor;
```

O Java também possui a capacidade de deduzir o tipo de dados de acordo com a primeira atribuição. Isso não significa que a variável não tenha tipo de dado, ou que esse tipo possa mudar, é apenas uma conveniência. Para declarar uma variável com esse recurso, utilizamos a palavra-chave **var**.

```
var nomeDaVariavel = valor;
```

Ao declarar uma variável, é importante saber que:

- O **nome de variáveis** é ***case-sensitive***, pois existe a diferenciação entre nome escritos em letras maiúsculas e minúsculas.
- **Não é possível começar o nome de uma variável** com um dígito (número) ou utilizar uma palavra reservada da linguagem.
- Por convenção, **iniciamos o nome de variável com letra minúscula**.
- Para a **atribuição de valores** a variáveis no Java, utilizamos o símbolo = .

IMPORTANTE

Não confundir atribuição (=) com comparação de igualdade (==).

Constantes

Uma **constante** é declarada quando precisamos lidar com **dados que não devem ser alterados durante a execução do programa**. É comum utilizar a palavra reservada **`final`** para representar a constante. Com ela, a variável só pode ser inicializada uma única vez, virando constante. Analise os exemplos a seguir.

Figura 6 – Exemplo de código para declaração e atribuição de variáveis e constantes

```
</>
```

```
// Exemplos de declaração e atribuição de variável
    int    numero;
    float  altura = 1.70f;
    String nome;

    // Exemplos de declaração e atribuição de constante
    final float  PI          = 3.1416F;
    final String NOME_PAGINA = "home";
```

Fonte: Autores (2020).

Expressões aritméticas

As **expressões aritméticas** do Java envolvem **operadores** aritméticos e **operandos**, que são variáveis ou constantes do tipo **numérico**, indicados na tabela a seguir.

Tabela 5 – Operadores aritméticos

Operador	Função	Exemplos
+	*Soma*	`3 + 4; x + y;`
-	*Subtração*	`10 – 5; a – b;`
*	*Multiplicação*	`5 * 3; m * n;`
/	*Divisão*	`12 / 6; x / y;`
%	*Resto da divisão inteira*	`9 % 2;  // Obs: o resultado desta operação é igual a 1.`

Precedência = sequência de resolução da fórmula:	
1º	*Parênteses:* **(** *e* **)**
2º	*Multiplicação e divisão:* ***** *e* **/**
3º	*Soma e subtração:* **+** *e* **-**

Exemplo:	Resultado:

`int x = 3 + 4 - 8 / 4 * 3 - 5;`
`x = 3 + 4 – 2 * 3 – 5`
`x = 3 + 4 – 6 – 5`
`x = 7 – 6 –5`
`x = 1 – 5`
`x = –4`
	`x = –4`

`int x = 3 + (4 - 8) / 4 * (3 - 5);`
`x = 3 + (–4) / 4 * (–2)`
`x = 3 + (–1) * (–2)`
`x = 3 + 2`
`x = 5`
	`x = 5`

Quando houver mais de um operador com a mesma precedência: realizar a operação que aparecer **primeiro**, na leitura da esquerda para a direita.

Fonte: Autores (2020).

Expressões lógicas

As **expressões lógicas** envolvem **operadores lógicos** ou **operadores relacionais**, e os **operandos** são relações ou variáveis ou constantes do tipo **lógico**.

Tabela 6 – Operadores lógicos

Negação / Não logico: ! (not)

x	!x
true	false
false	true

E lógico: && (and)

x	y	(x && y)
true	true	true
true	false	false
false	true	false
false	false	false

OU lógico: || (or)

x	y	(x || y)
true	true	true
true	false	true
false	true	true
false	false	false

Precedência = sequência de resolução da fórmula:	
1º	*Parênteses:* **(e)**
2º	*Negação:* **!**
3º	*E lógico:* **&&**
4º	*OU lógico:* **||**

Exemplo:	Resultado:

`boolean p = (8 > 4) && !((7 >= 10) || (5 != 2));`
`p = true && !(false || true)`
`p = true && !(true)`
`p = true && false`
`p = false`
	`p = false`

`boolean p = (8 > 4) && !(7 >= 10) || (5 != 2);`
`p = true && !false || true`
`p = true && true || false`
`p = true || false`
`p = true`
	`p = true`

Quando houver mais de um operador com a mesma precedência: realizar a operação que aparecer **primeiro**, na leitura da esquerda para a direita.

Fonte: Autores (2020).

Tabela 7 – Operadores relacionais

Operador	Função	Exemplos	
==	Igual a	`3 == 3; x == y;`	O resultado de uma operação relacional sempre é um valor lógico, ou no Java **boolean:** **true** ou **false**.
>	Maior que	`5 > 4; x > y;`	
<	Menor que	`3 < 6; x < y;`	
>=	Maior ou igual a	`5 >= 3; x >= y;`	
<=	Menor ou igual a	`3 <= 5; x <= y;`	
!=	Diferente de	`8 != 9; x != y;`	

Fonte: Autores (2020).

Neste exemplo, são apresentados códigos Java para declaração de variável, atribuição, utilização de operadores aritméticos, lógicos e relacionais.

Os **comentários** são um recurso importante da linguagem, pois são trechos do código ignorados pelo compilador, servem para auxiliar outros programadores a lerem e compreenderem o seu código e para você recordar o que pretendia fazer quando escreveu o seu programa.

- **Comentários** feitos na mesma linha começam com //
- **Bloco de comentários** são delimitados por /* e */

Figura 7 – Exemplo de código para declaração e atribuição

</>

```
public class Exemplo1 {

public static void main(String[ ] args) {

int x, y;
// declaração de variáveis inteiras: x e y

int X;
// declaração de outra variável inteira X
(ex. de case sensitive)

// não é boa prática começar nome de
variável com maiúscula!
boolean p, q;
// declaração de variável lógica
float a = 10.5f;
// declaração de variável do tipo float,
que já recebe valor;

x = 10;
// atribuição de valor à variável
y = x - 38;
// atribuição do valor de - 28 para a
variável

X = 9 % 2;
// X recebe o resultado da operação, que é
igual a 1.

p = (3 >= 5);
// p recebe o valor de false
q = (true || false);
// q recebe o valor de true

System.out.println("x = x);
// imprime x = 10 na console
System.out.println("y = " + y);
// imprime y = -28 na console
System.out.println("X = "+ X);
// imprime X = 1 na console
System.out.println("p = " + p);
// imprime p = false na console
System.out.println("q = " + q);
// imprime q = true na console
System.out.printf ("a = %. 2f", a*3);
// imprime a = 31,5 na console
     }
}
```

```
Console
x = 10
y = -28
X = 1
p = false
q = true
a = 31,50
```

Fonte: Autores (2020).

Leitura e escrita de dados no dispositivo-padrão

Um programa geralmente **interage com o usuário** para **obter dados** e para **exibir o resultado** do seu processamento na tela do computador. Essas operações básicas são:

- A **entrada de dados** (**INPUT ou leitura de dados**) é feita quando o programa recebe em uma variável o conteúdo digitado no **teclado**, que é o **dispositivo-padrão** para obtenção de dados.
- A **saída de dados** (**OUTPUT ou escrita de dados**) é feita quando o programa envia o resultado de um processamento para o *console*, ou tela, que é o **dispositivo-padrão** para exibição de dados.

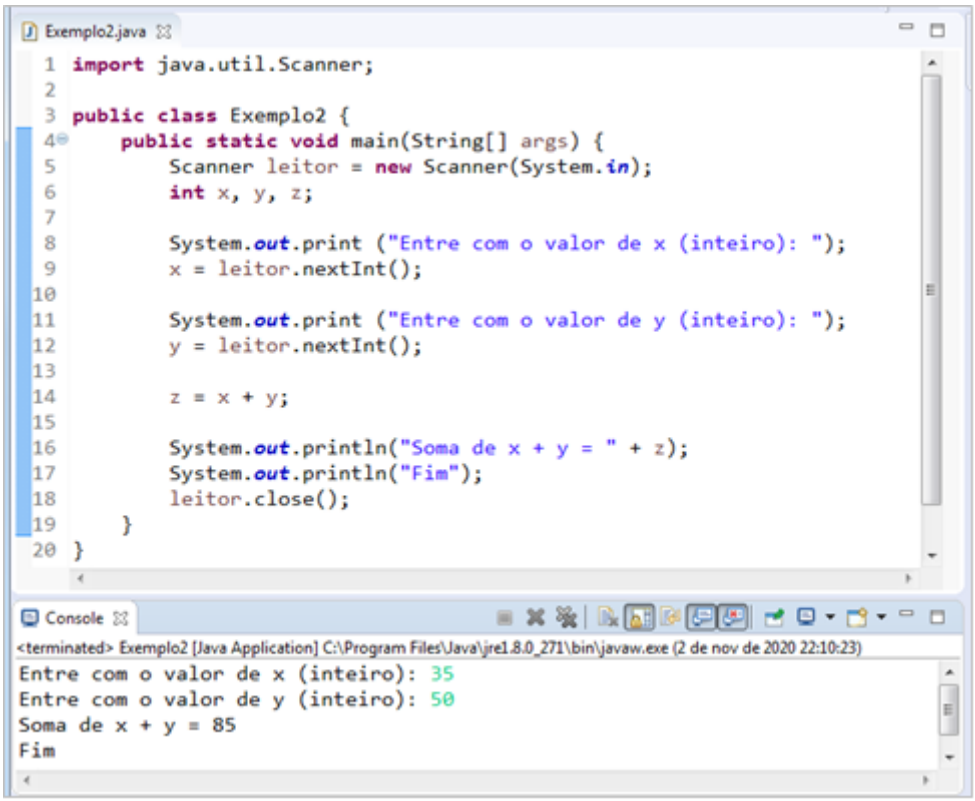
No Java existe uma biblioteca de funcionalidades disponibilizadas no pacote de classes **`java.util`**, no qual a classe **Scanner** implementa operações de **entrada de dados pelo teclado**. Esta classe possui diversos métodos para obter diferentes tipos de dados, sendo os mais utilizados:

- **`String Scanner.next()`** → retorna uma cadeia simples de caracteres, **sem** espaço.
- **`int Scanner.nextInt()`** → retorna um valor do tipo **`int`**.
- **`double Scanner.nextDouble()`** → retorna um valor do tipo **`double`**, com ponto flutuante.
- **`String Scanner.nextLine()`** → retorna uma cadeia de caracteres, **com** espaço.

Para **saída de dados**, usamos os métodos:

- **`System.out.println(`**`"Texto"`**`);`** → escreve `Texto` no console e vai para a **linha de baixo.**
- **`System.out.print (`**`"Texto"`**`);`** → escreve `Texto` no console e permanece na **mesma linha**.

Figura 8 – Exemplo de código para leitura e escrita de dados



Fonte: Autores (2020).

Comandos de desvio e repetição

Desvio com o comando "*If-else*"

Em algumas situações, precisamos executar ou desviar de um trecho de código, dependendo de uma condição. O Java apresenta duas estruturas de decisão importantes, o `if - else` e o **operador ternário**. Veja os exemplos na figura a seguir.

O `if` executa uma instrução, ou bloco de código, caso a uma condição seja **verdadeira**. Opcionalmente, podemos usar a instrução else, que será executada caso o `if` seja **falso**.

Este programa exibe no console o texto “menor de idade”, pois a condição (`idade < 18`), presente na linha 3, é verdadeira.

Se o valor da variável `idade` for alterado para 18, o programa imprime “maior de idade”, pois a expressão (`idade < 18`) agora é falsa.

O **operador ternário**, na linha 1 do segundo exemplo, tem o mesmo resultado da condição do primeiro exemplo.

Figura 9 – Exemplo de código de ***if*** para desvio dependente de condição (decisão)

```java
int idade = 15;

if (idade < 18) {
    System.out.println("Menor de idade");
} else {
    System.out.println("Maior de idade");
}
```

```java
System.out.println(idade < 18 ? "Menor de idade" : "Maior de idade");
```

Fonte: Autores (2020).

Desvio com o comando "*switch*"

Em outras situações, precisamos desviar o fluxo de execução do nosso programa baseado em um conjunto de valores. Nesses casos, o *switch* é mais indicado. Vejo o código-exemplo na Figura 10.

Observe que, na linha 2, o comando `switch` de baseia no valor da variável inteira `escolha` – que também poderia ser uma `string`.

Em cada ***case***, colocamos o valor esperado seguido de dois pontos (***case*** 3:) e o trecho de código que será executado se o ***case*** corresponder ao valor da variável testada. No caso do exemplo, `escolha` tem valor igual a 1, portanto, será exibida no *console* a frase`"Escolha 1 selecionada"`.

O código dentro de cada ***case*** executa até que um ***break*** seja executado. Se não houver um comando ***break***, o código prosseguirá até o próximo ***case***. Se o valor de escolha fosse igual a 2, seriam exibidas no console as frases:

> `"Escolha 2 selecionada"`.
> `"Escolha 2 selecionada"`.

Por fim, existe o bloco opcional ***default***, executado se as condições anteriores não forem atendidas.

Figura 10 – Exemplo de código de switch para desvio dependente de condição (decisão)

</>

```
int escolha = 1;
   switch (escolha) {
   case 1:
      System.out.println("Escolha 1 selecionada");
      break;
   case 2:
      System.out.println("Escolha 2 selecionada");
   case 3:
      System.out.println("Escolha 3 selecionada");
      break;
   default:
      System.out.println("Escolha opção selecionada");
      break;
}
```

Exemplo de código de decisão utilizando a estrutura *switch*. Fonte: Autores (2020).

Repetição com os comandos "*do*" e "*do-while*"

Os comandos de repetição executam um conjunto de instruções, ou bloco de comandos, enquanto uma condição for **verdadeira**. Veja os exemplos a seguir.

Figura 11 – Exemplo de código de while e do-while para repetição dependente de condição (decisão)

</>

```
// -- exemplo WHILE -------------------
   int i    = 1;
   int soma = 0;
   while (i <= 3) {
      soma = soma + i;// acumula o valor de
i em soma
      i++;            // aumenta o valor de
i de 1 em 1
   }
   System.out.println("Soma = " + soma); //
6 = 1 + 2 + 3

// -- mesmo exemplo anterior com DO-WHILE --
--------------
   i    = 1;
   soma = 0;
   do {
      soma = soma + i;// acumula o valor de
i em soma
      i++;            // aumenta o valor de
i de 1 em 1
   }while (i <= 3);
   System.out.println("Soma = " + soma); //
6 = 1 + 2 +
```

```
Console
Soma = 6
Soma = 6
```

Exemplo de códigos de repetição que geram o mesmo resultado apresentado no console. Fonte: Autores (2020).

Repetição com o comando "*for*"

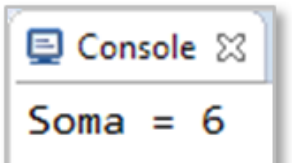
A instrução `for` é formada por três partes, conforme apresentadas no exemplo a seguir.

a. **Inicialização:** permite a criação e inicialização de variáveis.
b. **Condição:** mantém o `for` repetindo seu bloco de comandos enquanto verdadeira.
c. **Operação:** executada ao final de cada iteração (repetição) do `for`.

Figura 12 – Exemplo de código de for para repetição dependente de condição (decisão)

</>

```
   int soma = 0;
   for (int i = 1; i <= 3; i++) {
      soma = soma + i; // acumula o valor de
i em soma
   }
   System.out.println("Soma = " + soma); //
6 = 1 + 2 + 3
```



Exemplo de código de repetição. Fonte: Autores (2020).

Interrupção de repetição com o "*continue*" e com o "*break*"

Em algumas situações, precisamos interromper um comando de repetição. Para estes casos, o Java disponibiliza dois comandos, exemplificados na Figura 13.

- ***Continue***: interrompe a repetição e executa a condição novamente.
- ***Break:*** interrompe a repetição incondicionalmente.

Figura 13 – Exemplo de código com interrupção de repetição

</>

```
// i começa em 1 e aumenta de 3 em 3 até <=
20
   for (int i = 1; i <= 20; i+=3) { // 1,
4, 7, 10, 13, 16, 19
      if (i % 2 == 1) { // se é número
ímpar: resto de i por 2 = 1
         continue;      // interrompe e vai
para a condição
      }
      System.out.println(i + " ");
   }
   System.out.println("Fim 1\n\n");

// i começa em 1 e aumenta de 3 em 3 até <=
20
   for (int i = 1; i <= 20; i+=3) { // 1,
4, 7, 10, 13, 16, 19
      if (i % 5 == 0) { // se múltiplo de
5: resto de i por 5 = 0
         break;         // interrompe e sai
da repetição
      }
      System.out.println(i + " ");
   }
   System.out.println("Fim 2");
```

```
Console
10
4
10
16
Fim 1


1
4
7
Fim 2
```

Exemplo de código de repetição com interrupção. Fonte: Autores (2020).

Estrutura de dados: vetores e matrizes

Muitas vezes, precisamos trabalhar com listas de valores. Para isso, o Java permite definir estruturas conhecidas como vetores (*arrays*). Elas associam um conjunto de valores a um índice, iniciado em 0. Podemos declarar um *array* utilizando o operador de [ ]. Confira alguns exemplos nas figuras a seguir.

Figura 14 – Exemplo de código para manipular **vetor** com ***for***

</>

```
int [] vet = new int[] {10,2,30,-5};

   for (int i = 0; i < 4; i++) {
   // Imprime o conteúdo do vetor na posição
i
      System.out.print(vet[i] + " ");
   }
   System.out.println("\nfim");
```

```
Console
10 2 30 -5
fim
```

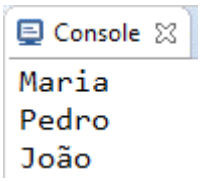
Exemplo de código de com vetor de inteiro `vet` .Fonte: Autores (2020).

Utilizar um laço de repetição para manipular estruturas como vetores é uma ação tão comum que um comando ***for*** foi criado apenas para isso, chamado ***for-each*** (ou para-cada). Veja o exemplo a seguir.

Figura 15. Exemplo de código para manipular***vetor*** com o iterador ***for-each***

```
String [] estudantes = new String[]
{"Maria", "Pedro", "João"};

   for (String estudante :  estudantes) {
         System.out.println(estudante);
   }
```



Exemplo de código de com vetor de inteiro `vet` .Fonte: Autores (2020).

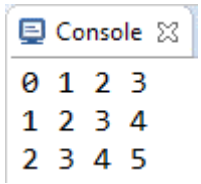
No exemplo a seguir, uma estrutura de matriz de valores inteiros é criada, em que as variáveis i e j são usadas como índices de linha e coluna, respectivamente. Um laço de repetição duplo (um comando ***for*** dentro de outro comando ***for***) permite percorrer cada elemento da `mat[i][j]`, que recebe (atribuição) o valor da variável `valor`.

Figura 16 – Exemplo de código para manipular **matriz**

```
final int LIN = 3; // Constante para número
de linhas
   final int COL = 4; // Constante para
número de colunas

   int [][] mat = new int[LIN][COL];
   int valor;

   for (int i = 0; i < LIN; i++) {
      for (int j = 0; j < COL; j++) {
         valor = i + j;
         mat[i][j] = valor;
         System.out.print(mat[i][j] + " ");
      }
      System.out.println();
```



Exemplo de código de com a matriz de inteiros `mat`.Fonte: Autores (2020).

EXPERIMENTE

Considerações finais

Nesta Unidade, vimos os principais conceitos de programação básica usando a linguagem Java, sendo que a prática desses conceitos apenas foi possível após a instalação completa de um IDE, ou ambiente de desenvolvimento, que será utilizado em todas as Unidades desta disciplina.

Com os recursos básicos de programação trabalhados, tais como declaração e atribuição de variável, leitura e escrita de dados nos dispositivos-padrão (teclado e tela do computador), utilização de comandos de desvio e repetição, e utilização de estruturas de dados como vetores e matrizes, estamos prontos para prosseguirmos explorando os conceitos da orientação a objetos nas próximas Unidades.

Referências

ALVES, G. F. O. Qual a diferença entre JDK, JRE e JVM. **Dicas de Java**. Disponível em: https://dicasdejava.com.br/qual-a-diferenca-entre-jdk-jre-e-jvm/. Acesso em: 1 nov. 2020.

GODOY, V. **Programação orientada a objetos I**. Curitiba: IESDE, 2019.

HORSTMANN, C. S.; CORNELL, G. **Core Java – volume I**. 8. ed. São Paulo: Pearson, 2010.

ORACLE. Primitive Data Types. **ORACLE**, The Java™ Tutorials. Disponível em: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html. Acesso em: 1 nov. 2020.

SCHILDT, H. **Java para Iniciantes**. Porto Alegre: Bookman, 2015.

TIOBE. Programming Community Index. **TIOBE Index for February 2021**., out. 2020. Disponível em: *https://www.tiobe.com/tiobe-index/*. Acesso em: 1 nov. 2020.

© PUCPR - Todos os direitos reservados.