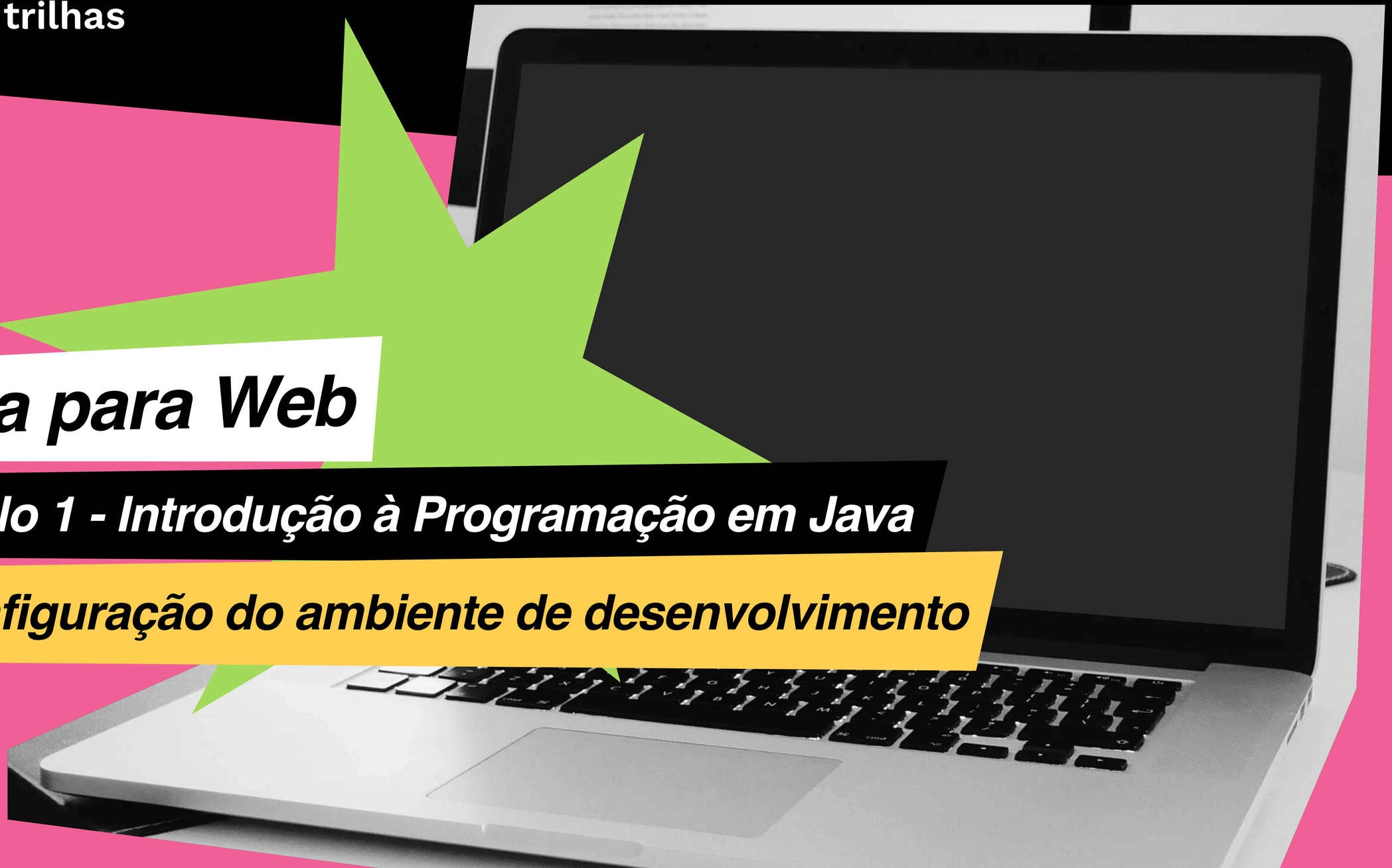
TIC em trilhas

Java para Web

Módulo 1 - Introdução à Programação em Java

e configuração do ambiente de desenvolvimento



TIC em trilhas

Sumário

Aula 1: Básico do Java

Aula 2: Coleções e Estrutura de dados em Java

TIC em trilhas

Aula 1 - Básico do Java

1.1 O que é Java?

História e Sintaxe

O Java, como plataforma de programação, surgiu em 1995 nos laboratórios da empresa Sun Microsystems, como resultado de uma extensa pesquisa científica e tecnológica. A plataforma oferece um ambiente completo para o desenvolvimento e execução de programas, composta por:

- Uma linguagem de programação de alto nível orientada a objetos;
- Máquina Virtual Java (Java Virtual Machine ou **JVM),** que assegura independência de plataforma, permitindo que o código seja executado na máquina virtual e portabilidade para diferentes plataformas, como Windows ou Linux;
- O Java Runtime Environment ou **JRE,** que inclui a máquina virtual e alguns recursos necessários para a execução de aplicações Java;
- O Java Development Kit ou **JDK,** que é um conjunto de utilitários que oferece suporte ao desenvolvimento de aplicações;

Em Java, os programas são escritos em arquivos com a extensão **.java,** os quais são posteriormente compilados para arquivos com a extensão **.class.** Estes últimos contêm os códigos a serem executados na máquina virtual, os bytecodes.

Na linha do tempo a seguir você conhecerá os principais marcos na história dessa linguagem.

Ilustração feita por upklyak, disponível no Freepik. Adaptada pelo autor.

1991

1991, Junho: O projeto Java é iniciado pelo Time Green da Sun Microsystems, liderado por James Gosling, com o objetivo inicial de desenvolver uma linguagem de programação para dispositivos eletrônicos. A equipe busca criar uma linguagem que seja simples, robusta, portável, independente de plataforma, segura e de alta performance. Gosling e sua equipe trabalham arduamente no desenvolvimento do Java, inspirados pela necessidade de uma linguagem que possa lidar com os desafios emergentes da computação distribuída e da comunicação entre dispositivos digitais.

1995

1995: O projeto Green evolui para se concentrar no desenvolvimento de uma linguagem de programação para a web. Inicialmente chamada de "Greentalk" devido à herança do projeto Green, a linguagem é posteriormente renomeada para "Oak". No entanto, devido a restrições de marca registrada, é finalmente renomeada como "Java".

1995, Maio: O lançamento inicial do Java ocorre. A versão 1.0 do JDK (Java Development Kit) é disponibilizada ao público, marcando o início oficial da era Java. Esta versão inclui a linguagem Java, a Máquina Virtual Java (JVM) e outras ferramentas de desenvolvimento essenciais.

1995, Novembro: A Sun Microsystems anuncia publicamente o Java e suas capacidades revolucionárias. A linguagem é promovida como uma solução para o desenvolvimento de aplicativos multiplataforma e para a criação de conteúdo dinâmico na web.

1996 e 1997

1996, Janeiro: O JDK 1.0 é oficialmente lançado. Este marco é significativo, pois consolida o Java como uma tecnologia madura e pronta para uso em uma variedade de aplicações.

1996, Fevereiro: A Sun Microsystems lança o primeiro navegador web totalmente funcional baseado em Java, chamado "HotJava". Este navegador demonstra as capacidades dinâmicas e interativas do Java na web.

1997: A Sun Microsystems forma a Java Community Process (JCP), um esforço colaborativo para desenvolver e evoluir a plataforma Java de forma aberta e transparente. A JCP permite que empresas e indivíduos contribuam para o desenvolvimento do Java por meio de especificações técnicas e implementações de referência.

2000 até 2010

2000: A Sun Microsystems lança o código-fonte do Java sob a licença GNU General Public License (GPL). Este movimento, conhecido como "OpenJDK", marca um importante passo em direção à abertura da plataforma Java.

2006: A Sun Microsystems lança o Java SE 6, também conhecido como "Mustang". Esta versão introduz várias melhorias de desempenho, segurança e usabilidade, incluindo suporte para scripting dinâmico através da JSR 223.

2010: A Oracle Corporation adquire a Sun Microsystems, tornando-se a mantenedora oficial do Java. A Oracle continua a desenvolver e suportar a plataforma Java, mantendo o compromisso com a inovação e a comunidade Java.

2014 até 2022

2014: A Oracle lança o Java SE 8, uma versão importante que introduz a expressão lambda, o Stream API e outras melhorias significativas na linguagem e na biblioteca padrão.

2017: A Oracle lança o Java SE 9, marcando um importante avanço na evolução do Java. Esta versão introduz o conceito de módulos, que visam simplificar a criação, manutenção e implantação de aplicativos Java.

2020: A Oracle lança o Java SE 14, continuando o ciclo de lançamento regular de novas versões do Java. Esta versão inclui melhorias na linguagem, na JVM e em outras áreas, demonstrando o compromisso contínuo da Oracle com o desenvolvimento e aprimoramento da plataforma Java.

2022: A Oracle lança o Java SE 17: Expressões lambda aprimoradas, switch expressions, APIs para geração de números aleatórios e melhorias em performance e segurança marcam essa versão.

Atualmente

Atualmente: De um início simples, o Java se tornou uma gigante da tecnologia, impulsionando inovações em diversos setores, após anos de criação Java ainda é usado para aplicações seguras e inovadoras.

1.2 Configurando o ambiente (Windows e Linux)

1.2.1 Instalação no Windows

- **1º Passo:** Acesse o site da JetBrains (disponível aqui), role para baixo até encontrar a versão Community.

Confirme que está baixando o IntelliJ IDEA Community. IntelliJ IDEA também tem duas variantes: IntelliJ IDEA Community Edition (grátis) IntelliJ IDEA Ultimate Edition (versão paga com recursos extras).

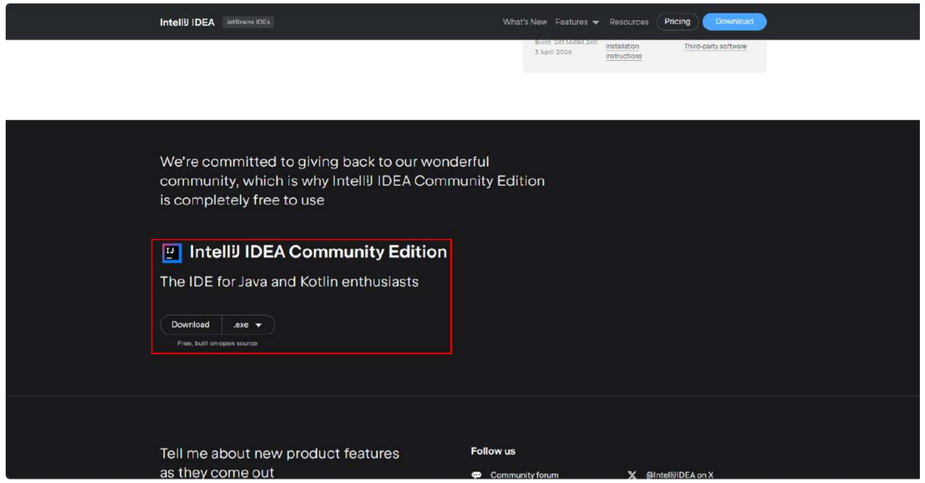


Figura 1: Instalação. Fonte: Print screen do site do IntelliJ IDEA.

- **2º Passo:** Clique em download para ele iniciar.
- **3º Passo:** Clique em “Next” até a tela abaixo aparecer e marque os campos conforme imagem e clique em next:

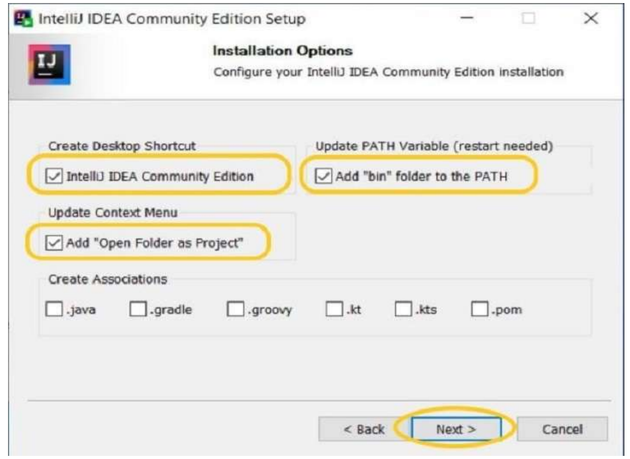


Figura 2: Instalação. Fonte: Print screen da terceira etapa no instalador.

- **4º Passo:** Vá avançando as etapas até a instalação concluir. Por fim, selecione para reiniciar o computador mais tarde e clique em "finish" para concluir a instalação.

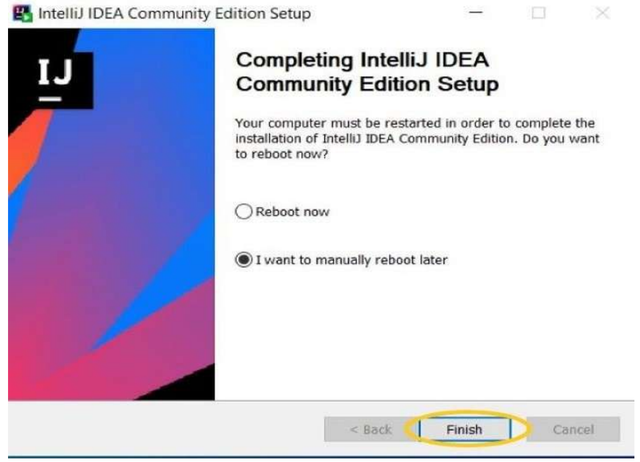


Figura 3: Finalizando instalação. Fonte: Print screen da última etapa no instalador.

1.2.2 Como instalar o IntelliJ no linux

Acesse o link do site da JetBrains aqui

1º Passo: Acesse o site e confira se está com o sistema Linux selecionado conforme a imagem abaixo:

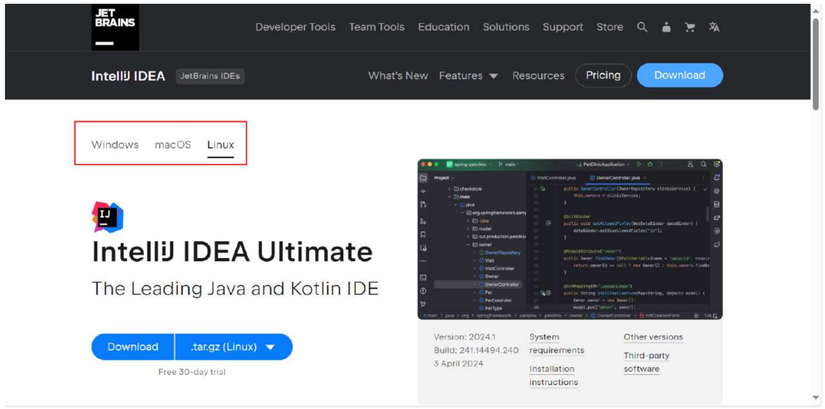


Figura 4: Instalação. Fonte: Print screen do site do IntelliJ com linux selecionado.

2º Passo: Desça na página até achar a Community edition

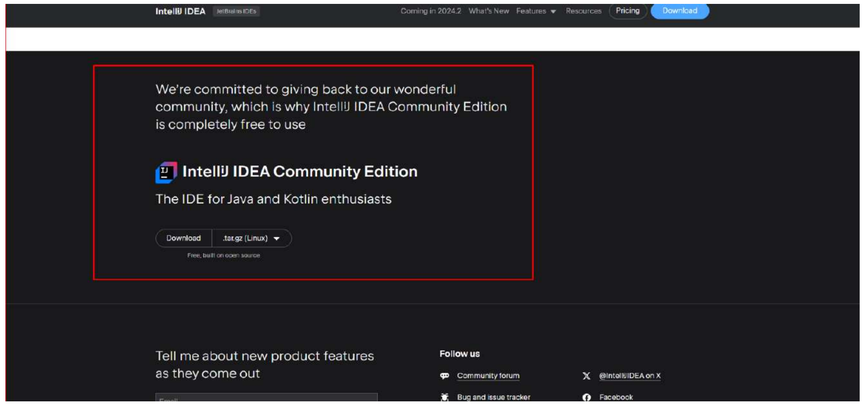


Figura 5: Instalação. Fonte: Print screen da parte do download correto no site da IntelliJ .

- **3º Passo:** Concluído o download, abra um terminal e "entre" na mesma pasta onde você baixou o pacote *.tar.gz* do IntelliJ IDEA. Caso esteja na pasta "**Downloads**" (por exemplo), basta executar o comando:

```
cd /home/$USER/Downloads
```

- **4º Passo:** Descompacte o tarball e renomeie a pasta resultante para "intellij-idea" com o comando:

```
tar -xvzf idea*.tar.gz && mv idea*/ intellij-idea
```

- **5º Passo:** Mova a pasta "intellij-idea" para o diretório "/opt" executando:

```
sudo mv intellij-idea /opt
```

- **6º Passo:** Crie também um link simbólico do IntelliJ IDEA no sistema com:

```
sudo ln -s /opt/intellij-idea/bin/idea.sh /usr/bin/idea
```

Feito isso, execute o comando **idea** ou **sh /opt/intellij-idea/bin/idea.sh** para abrir o IntelliJ IDEA pela primeira vez.

1.3 Operadores Lógicos, Relacionais e Aritméticos

1.3.1 Tipos Primitivos

Você já ouviu falar sobre tipos primitivos?
Os tipos primitivos são os tipos de dados básicos, como inteiros, números quebrados, caracteres e booleanos.

Eles ocupam uma quantidade fixa na memória e esses dados são armazenados diretamente na variável. Basicamente, ele serve para definir o espaço de memória que vamos usar.

Os tipos primitivos tem tamanhos diferentes que servem para guardar informações diferentes como podemos ver na imagem a seguir.

Figura 6: Quadro de tipos primitivos e seus respectivos tamanhos. Fonte: Feito pelo autor.

- **Tipos Integrais**

São como os números que usamos para contar objetos. Eles podem ser positivos ou negativos, mas não têm partes quebradas, como um bolo inteiro.

- **Tipos de Ponto Flutuante**

São como os números que usamos para realizar medidas, como a altura de uma pessoa. Eles podem ter partes quebradas, como se pudesse aceitar não só bolos inteiros, mas também as fatias de um bolo.

- **Tipo Booleano**

São como interruptores que podem estar ligados (True) ou desligados (False). São usados para tomar decisões, como decidir se você vai sair ou ficar em casa.

- **Tipo de Caractere**

São as letras e símbolos que temos no nosso teclado do computador. Chamamos de caractere (char) quando temos apenas um desses símbolos sozinhos.

1.3.2 Tipos compostos

São construídos a partir dos tipos primitivos e são formados de múltiplos elementos desses mesmos tipos. Alguns dos mais utilizados são:

- **Arrays:** Coleções ordenadas de elementos do mesmo tipo de dado;
- **Strings:** É um Array que representa uma sequência de caracteres;
- **Listas:** São como Arrays só que mais poderosos, possuindo mais funcionalidades;
- **Conjuntos:** São como as listas, porém não aceitam valores repetidos;
- **Mapas:** Também são como listas, porém armazenam os dados no formato chave-valor.

1.3.3 Variáveis

Para começar, é dessa forma que declaramos uma variável em Java:

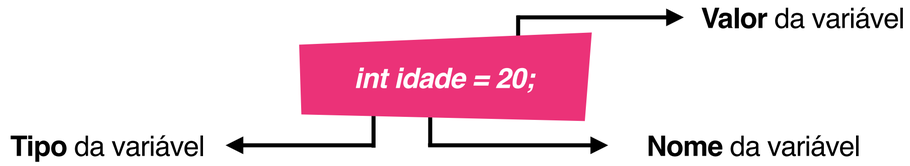


Figura 7: Exemplo de declaração de uma variável. Fonte: Feito pelo autor.

A variável serve para salvar algum valor na memória, de forma que fique mais fácil para acessar e/ou modificar ele quando quisermos.

- **Tipo da variável:** Vai ser de um dos tipos primitivos ou compostos;
- **Nome da variável:** É como um rótulo para que fique mais fácil encontrar e reutilizar ela depois;
- **Valor da variável:** É o valor que nossa variável vai ter.

“Extra”: O sinal de igual (=) é utilizado em Java para atribuir valor à alguma variável ou seja, estamos atribuindo o que está à direita do sinal no que está à esquerda dele.

Para os tipos compostos, será utilizado da mesma forma, como demonstrado na variável amigos no exemplo a seguir:

```
public class Main {
    public static void main(String[] args) {
        String nome = "Giancarlo";
        int idade = 22;

        String[] amigos = {"Joaquim", "Jeison", "Diego"};

        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
        System.out.println("Amigos: " +
Arrays.toString(amigos));
    }
}
```

Sinta-se à vontade para copiar e colar esse exemplo no IntelliJ, para rodar, modificar e testar livremente.

1.3.4 Operadores Lógicos, Relacionais e Aritméticos

- **Operadores Lógicos:**

São importantes para tomar decisões com base em condições lógicas. Principais exemplos:

&& (E): Se você quer saber se a pessoa gosta tanto de sorvete quanto de chocolate, usaria o "E".

|| (OU): Agora, se você só precisa saber se a pessoa gosta de pelo menos um dos dois, você usaria o "OU".

! (NÃO): Por fim, o "NÃO" é um pouco diferente. É como se fosse uma inversão. Se você perguntar "Você NÃO gosta de sorvete?", a resposta seria o oposto do que a pessoa disser. Se ela disser "sim", isso significa que ela não gosta de sorvete. Se ela disser "não", significa que ela gosta.

- **Operadores Relacionais ou de Comparação**

São utilizados para comparar dois valores. Veja os principais:

Operador	Função
>	Verifica se o valor da esquerda é **Maior que** o valor da direita
<	Verifica se o valor da esquerda é **Menor que** o valor da direita
>=	Verifica se o valor da esquerda é **Maior ou Igual que** o valor da direita
<=	Verifica se o valor da esquerda é **Menor ou Igual que** o valor da direita
'=='	Verifica se o valor da esquerda é **Igual ao** valor da direita
!=	Verifica se o valor da esquerda é **Diferente que** o valor da direita

- **Operadores Aritméticos**

São utilizados para realizar operações matemáticas entre variáveis numéricas. Veja os principais:

Operador	Função	Cálculo	Resultado esperado
+	Operador de adição	1+1	2
-	Operador de subtração	3-1	2
*	Operador de multiplicação	3*4	12
/	Operador de divisão	8/2	4
%	Operador de módulo (ou resto da divisão)	5%2	1

Com o exemplo abaixo, podemos ver no código esse mesmo teste e como cada um desses sinais se comporta:

```
public class Main {
    public static void main(String[] args) {
        System.out.println(1 + 1);
        System.out.println(3 - 1);
        System.out.println(3 * 4);
        System.out.println(8 / 2);
        System.out.println(5 % 2);
    }
}
```

1.4 Estruturas de controle (condições, repetições)

1.4.1 Estrutura if, else if e else

Essas estruturas servem para tomar decisões no nosso código.

Neste exemplo temos 1 variável e verificamos:

- Se a idade for maior que 18, imprimimos “É maior de idade.”;
- Se a idade for exatamente 18, imprimimos “Tem exatamente 18 anos.”;
- Se nenhum dos dois for verdade, imprimimos “É menos de idade.” ;

```
public static void main(String[] args) {
    int idade = 20;

    if (idade > 18) {
        System.out.println("É maior de idade.");
    } else if (idade == 18){
        System.out.println("Tem exatamente 18 anos.");
    } else {
        System.out.println("É menor de idade.");
    }
}
```

1.4.2 Estrutura switch-case

Temos também a opção de usar o switch-case, que é ideal para situações com muitas opções, onde você pode escolher um caso específico para executar.

```
public static void main(String[] args) {
    int opcao = 2;

    switch (opcao) {
        case 1:
            System.out.println("Opção 1 selecionada.");
            break;
        case 2:
            System.out.println("Opção 2 selecionada.");
            break;
        case 3:
            System.out.println("Opção 3 selecionada.");
            break;
        default:
            System.out.println("Opção inválida.");
    }
}
```

1.4.3 Estruturas de Loops: for, while e o do-while

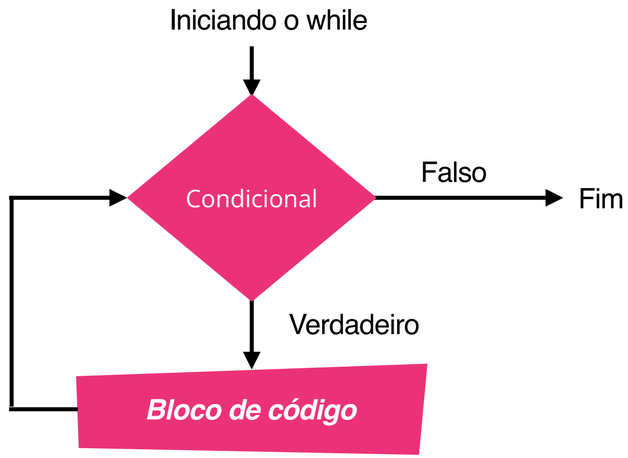


Figura 8: Estrutura básica de um loop. Fonte: Feito pelo autor.

Para execução de um bloco de código de multiplas vezes usamos os loops, também conhecidos como estruturas de repetição que permitem executar um conjunto de instruções repetidamente enquanto uma determinada condição é atendida.

- **For:** Você o utiliza quando sabe exatamente quantas vezes deseja repetir uma ação.

```
for (int i = 0; i < 5; i++) {
    System.out.println("Iteração #" + i);
}
```

- **While:** Executa um bloco de código enquanto uma condição for verdadeira.

```
int contador = 0;

while (contador < 5) {
    System.out.println("Iteração #" + contador);
    contador++;
}
```

- **Do-while**: O bloco de código é executado pelo menos uma vez, mesmo que a condição seja falsa inicialmente.

```
int contador = 0;

do {
    System.out.println("Iteração #" + contador);
    contador++;
} while (contador < 5)
```

Com essas estruturas estamos prontos para a automatizar e controlar o fluxo da nossa aplicação. Permitindo que ele tome decisões e transforme tarefas repetitivas em tarefas simples.

TIC em trilhas

Foto de Clément Hélardot, disponível na Unsplash. Adaptada pelo autor.

Foto de Volodymyr Hryshchenko, disponível na Unsplash. Adaptada pelo autor.

Aula 2 - Coleções e Estrutura de dados em Java

2.1 Enum

Um enum (enumeração) em Java é um tipo de dado usado para representar um conjunto fixo de valores nomeados. Pense em um cardápio de uma sorveteria com opções de sabores: Chocolate, Baunilha, Morango.

Características

- Define constantes nomeadas;
- Limita as opções a um conjunto fixo;

Use enums quando tiver um conjunto predefinido de valores que não mudarão;

Útil para melhorar a legibilidade do código e evitar erros de digitação;

Podemos ver isso no exemplo a seguir:

```
public static void main(String[] args) {
   System.out.println("Você escolheu o sabor " +
SaboresDeSorvete.CHOCOLATE);
}

enum SaboresDeSorvete {
   CHOCOLATE, BAUNILHA, MORANGO;
}
```

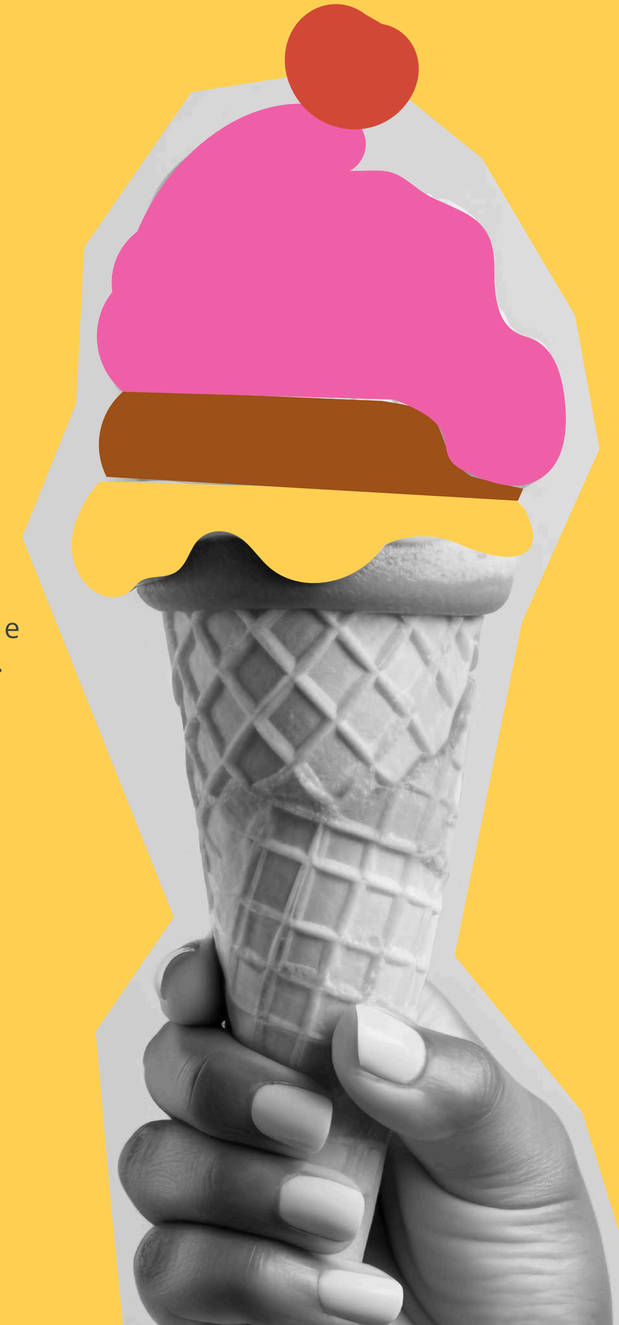


Foto de Freepik disponível na Freepik. Adaptada pelo autor.

É importante frisar que os enums são única e exclusivamente para ajudar o programador. Eles não fazem diferença nenhuma para o usuário, serve para evitar erros na hora de desenvolver nossa aplicação e deixar mais legível caso tenhamos que fazer alguma manutenção futura.

2.2 Arrays e Arraylist

2.2.1 Arrays

Arrays, ou vetores, são coleções simples de coisas. Neles, é possível armazenar vários objetos do mesmo tipo para melhor organizar os dados que você precisa manipular dentro do seu programa.

Podemos usar o loop for para percorrer cada elemento do array de forma fácil:

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

for (int indice = 0; indice < nomes.length; indice++) {
    String nomeDaVez = nomes[indice];
    System.out.println(nomeDaVez);
}
```

Um detalhe importante: usando o **.length**, garantimos que, independentemente do tamanho do array, passaremos por ele inteiro.

2.2.2 Arraylist

Arraylist são como vetores, porém seu tamanho varia de forma dinâmica, ou seja, conforme precisamos de mais ou menos espaço, ele aumenta ou diminui automaticamente.

Características :

- O ArrayList não precisa de um tamanho para começar e gerencia o tamanho da coleção conforme necessário;
- Ele permite que adicione, remova, pesquise, ordene itens, dentre outros recursos, que o array básico não sabe fazer.

Por exemplo, é assim que declaramos um Arraylist e adicionamos nomes nele:

```
ArrayList<String> nomes = new ArrayList<String>();

nomes.add("Gian");
nomes.add("Joaquim");
nomes.add("Diego");
nomes.add("Jeison");
nomes.add("Vanessa");
nomes.add("Alicia");
```

Repare que foi declarado dizendo o tipo da lista usando a sintaxe **<String>**.

O único ponto em que o ArrayList perde para o array tradicional é em velocidade, porque ele tem funcionalidades adicionais e isso consome recursos.

Você também pode utilizar ele dentro dos loops que já utilizamos antes:

```
ArrayList<String> nomes = new ArrayList<>();

nomes.add("Gian");
nomes.add("Joaquim");
nomes.add("Diego");
nomes.add("Jeison");
nomes.add("Vanessa");
nomes.add("Alicia");

for (int indice = 0; indice < nomes.size(); indice++) {
   String nomeDaVez = nomes.get(indice);
   System.out.println(nomeDaVez);
}
```

Nesse caso, a diferença é que usamos o método **.size()** para saber o tamanho da lista e o **.get(indice)** para pegar os valores dentro dela.

Por fim, além destes, abaixo estão alguns métodos do ArrayList comuns que podem ser utilizados para facilitar sua vida:

- **addAll()** para adicionar coleções inteiras nessa lista;
- **remove(indice)** para remover itens em posição específica;
- **clear()** para limpar a lista, entre outros.

É dessa forma então que podemos armazenar e manipular diversos valores do mesmo tipo de forma simples em nossos programas.

Para facilitar um pouco a visualização, é dessa forma que podemos ilustrar uma lista ou array:

Figura 9: Estrutura de um array/lista com índice. Fonte: Feito pelo autor.

2.3 Conjuntos (Set), HashSet e Mapas

2.3.1 HashSet

Um HashSet é uma lista, porém, os itens **não podem se repetir** e a ordem não importa! A vantagem é que, mesmo que você tente adicionar algo duplicado, não será possível e você não cometerá erros.

Para entender melhor, vamos ver um exemplo:

```
public static void main(String[] args) {
    var listaDeMercado = new HashSet<String>();

    listaDeMercado.add ("Maçã");
    listaDeMercado.add ("Farinha");
    listaDeMercado.add ("Laranja");
    listaDeMercado.add ( "Beterraba");
    listaDeMercado.add ("Laranja");
    listaDeMercado.add ("Laranja");

    System.out.println(listaDeMercado);
}
```

Veja que, nesse exemplo, mesmo tendo adicionado a laranja três vezes utilizando o método **.add()**, ela só aparece uma vez. Isso quer dizer que o próprio HashSet ignorou a segunda e a terceira entradas.

Outro ponto importante é que, assim como as listas, é necessário colocar o argumento de tipo do conjunto, usando o **<>**, e pode ser um conjunto de qualquer objeto. No nosso caso, utilizamos **HashSet<String>**.

2.3.2 HashMap

Map, ou mapa, é uma estrutura que é composta de uma lista de elementos que têm **chave e valor**. Isso permite que a gente tenha entradas com uma chave única e um valor associado a ela, que não precisa ser necessariamente único.

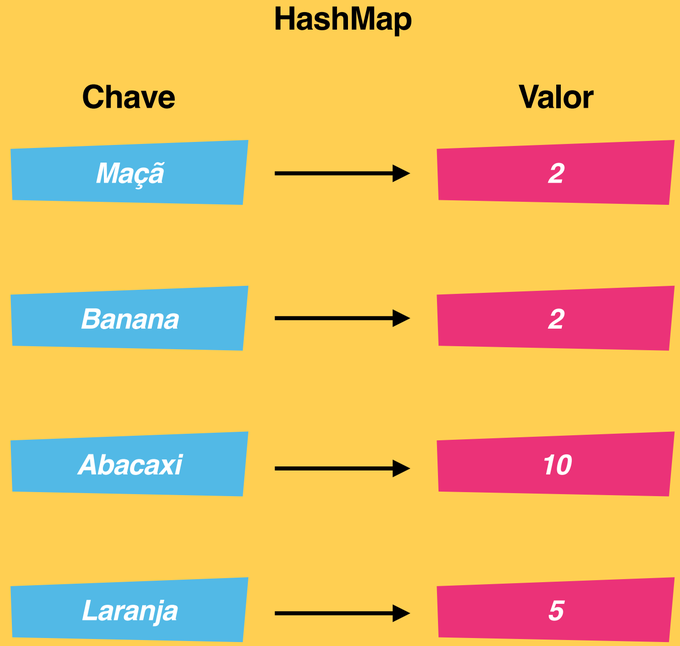


Figura 10: Exemplo de estrutura de um HashMap. Fonte: Feito pelo autor.

Veja um exemplo:

```
public static void main(String[] args) {
    var listaDeMercado = new HashMap<String, Integer>();

    listaDeMercado.put("Maçã", 12);
    listaDeMercado.put("Farinha", 30);
    listaDeMercado.put("Laranja", 20);
    listaDeMercado.put("Beterraba", 5);
    listaDeMercado.put("Laranja", 30);
    listaDeMercado.put("Laranja", 40);

    System.out.println(listaDeMercado);
}
```

Repare que, mesmo adicionando a laranja várias vezes utilizando o método **.put()**, o HashMap mantém o valor associado à **última vez que você adicionou** aquela chave.

O **HashMap** também precisa de parâmetros de tipo. Mas, neste caso, usamos um parâmetro para a chave e outro para o valor: **HashMap<String, Integer>.**

- **O conceito de HASH**

O termo "hash" diz que o java usa um algoritmo que serve como uma **etiqueta única** para não só garantir que algum valor não vai se repetir, mas também para encontrar, remover e adicionar os itens dessa lista mais rápido.

Explore mais!

Para saber um pouco mais sobre o famoso código “Olá mundo!” e suas partes, confira neste link.

Para se desafiar e se aventurar no mundo da programação, uma listinha de exercícios, focados no básico de programação em Java. Bora se aventurar? Comece por aqui!

Para explorar o mundo das constantes em Java, acesse o site.

Ilustração feita por upklyak, disponível no Freepik. Adaptada pelo autor.

Referências bibliográficas

História do Java. JavaTpoint, [s.d.]. Disponível em: <https://www.javatpoint.com/pt/hist%C3%B3ria-do-java>. Acesso em: 15 abr. 2024.

Loops (Repetições) em java. Javatpoint, [s.d.]. Disponível em: <https://www.javatpoint.com/pt/loops-em-java>. Acesso em: 16 abr. 2024.

OSÓRIO, Victor. Java 101: Collections!?!? Listas, conjuntos e mapas... as classes mais usadas do Java! 2022. Disponível em: <https://blog.vepo.dev/posts/java-101-collections>. Acesso em: 16 abr. 2024.