



Java

APOSTILA



São Paulo-2021

Créditos

Organização e Produção

Davi Domingues

Douglas de Oliveira

Fernando Esquírio Torres

Rafael Lopes dos Santos



É proibida a duplicação ou reprodução deste volume, no todo ou em parte, em quaisquer formas ou por quaisquer meios (eletrônicos, mecânico, gravação, fotocópia, distribuição pela internet ou outros), sem permissão prévia do Instituto da Oportunidade Social.

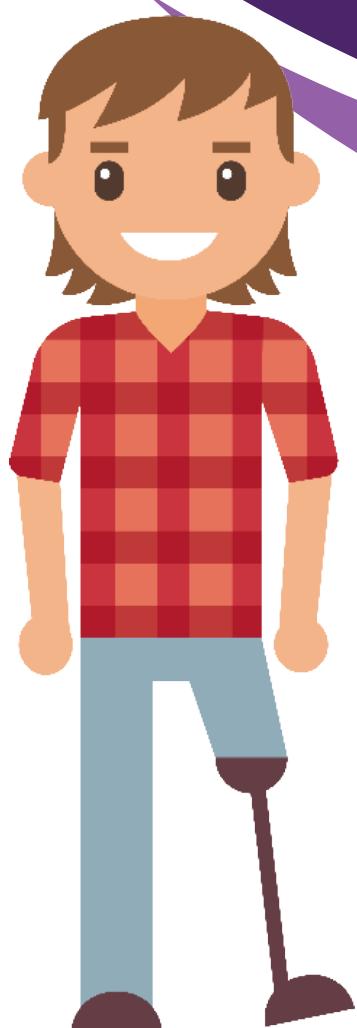
São Paulo
2021

Atenção

Em caso de dúvidas, sugestões ou reclamações. Entre em contato com a equipe de conteúdo.

educacional@ios.org.br

ESSE É SUA APOSTILA!



Ela vai te acompanhar durante todo o período do curso.

Cuide dela com carinho e responsabilidade.

Para que ela chegasse toda cheia de estilo do jeito que você está vendo, muita gente quebrou a cabeça para te entregar a melhor experiência de aprendizagem.

Ótimos estudos!

SUMÁRIO

Introdução ao Java	5
Ambiente de Desenvolvimento	17
Tipos de dados e Operadores.....	26
Desvios condicionais – if/else	44
Desvios condicionais encadeados.....	56
Estruturas de Repetição while e do-while	69
Estrutura de Repetição for	83
Teste de Mesa	92
Vetor.....	99
Strings.....	109
Matriz.....	116



Introdução ao Java

Os objetivos desta aula são:

- Conhecer a história do Java;
- Diferenciar JDK, JRE e JVM;
- Compreender o que é um algoritmo e quais são suas etapas.

Bons estudos!

Sobre o Java

Java é uma **linguagem de programação orientada a objetos**, de **alto nível** e baseada em **classes**, que foi projetada para ter **poucas dependências de implementação**. Ela é uma linguagem de programação de **propósito geral** destinada a permitir que os desenvolvedores programem o código **uma vez** e possam **executar em qualquer lugar**, ou seja, o **código Java compilado pode ser executado em todas as plataformas que suportam Java sem a necessidade de recompilação**.

A linguagem **Java** foi desenvolvida por **James Gosling** e sua equipe da **Sun Microsystems** e lançada em meados de **1995**. Ela é baseada nas linguagens **C** e **C++**. Originalmente, os

compiladores Java, máquinas virtuais e bibliotecas de classes foram lançados pela **Sun** sob **licenças proprietárias**. Atualmente, a Sun refez as licenças e a maioria das suas tecnologias está sob a licença **GPL-2.0** (General Public License, version 2). Isso contribuiu para a popularização do Java e, hoje, de acordo IEEE Spectrum, ele ocupa o segundo lugar no ranking de Top Programming Languages 2021, disponível em: <https://spectrum.ieee.org/top-programming-languages-2021>.



James Gosling

Configuração do Java

Antes de começar a implementar seus programas e começar a desenvolver aplicações em Java, você precisa realizar algumas **configurações no seu computador** para que tudo execute perfeitamente como desejado. A primeira configuração é a **instalação do JDK**, mas antes vamos entender a diferença entre **JDK, JRE e JVM**.

JDK vs JRE vs JVM

JDK (Java Development Kit) é um **kit de desenvolvimento Java** usado para implementar aplicativos e **aplicações em Java**. Ele está disponível para os diversos sistemas operacionais, tais como: **Windows, macOS, Solaris e Linux**. O JDK auxilia os desenvolvedores Java a **codificar e executar programas Java**. Você pode instalar mais de uma versão do **JDK** no seu computador.

Porque usar o JDK?



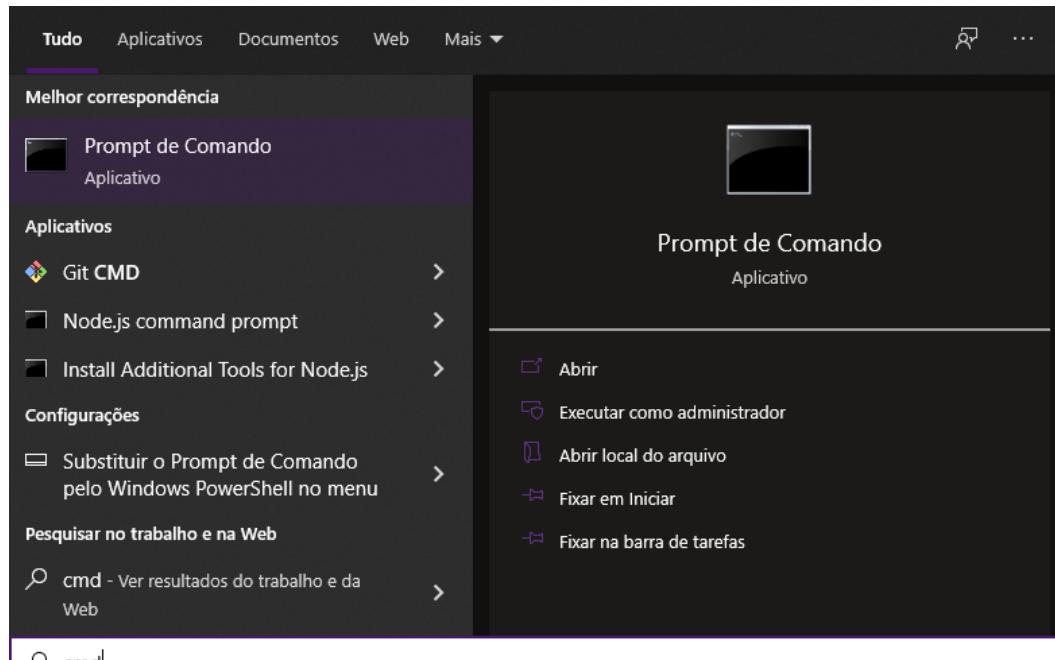
- *JDK têm ferramentas necessárias para escrever programas Java e JRE para executá-los.*
- *Inclui um compilador, Java application launcher, Appletviewer, etc.*
- *O compilador converte o código escrito em Java em código de bytes.*
- *O Java application launcher abre um JRE, carrega as classes necessárias e executa seu método principal.*

Para instalar o JDK, você pode acessar o site da **Oracle**: <https://www.oracle.com/java/technologies/javase-downloads.html> e em caso de dúvidas na instalação seguem dois tutoriais que podem te auxiliar na instalação e na configuração do Java:

Bóson Treinamentos (<https://www.youtube.com/watch?v=KeDhIDXezMs>)

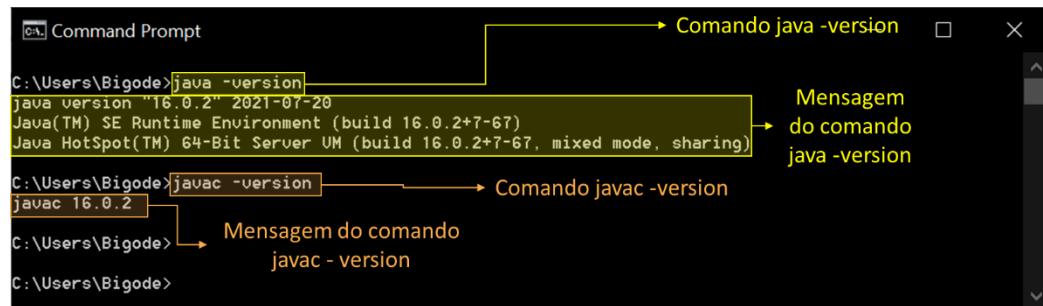
Área Restrita Brasil (<https://www.youtube.com/watch?v=k8SWBprwXyQ>)

No **Windows**, para abrir o terminal, você pode clicar no botão de pesquisa. Então digitar **cmd**, clique em **Prompt** de comando e então digite o comando **java -version** e, depois, o comando **javac -version**.



Prompt de comando.

Se tudo foi instalado corretamente uma mensagem semelhante deverá aparecer no terminal.



C:\Users\Bigode>java -version
java version "16.0.2" 2021-07-20
Java(TM) SE Runtime Environment (build 16.0.2+7-67)
Java HotSpot(TM) 64-Bit Server VM (build 16.0.2+7-67, mixed mode, sharing)
C:\Users\Bigode>javac -version
javac 16.0.2
C:\Users\Bigode>
C:\Users\Bigode>

Terminal mostrando que o JDK foi instalado corretamente.



Dica!

Sempre que você *instalar o JDK* é necessário **verificar se o path foi configurado corretamente**, caso os comandos **java -version** ou o **javac -version** retornem que não foram encontrados, você pode rever o vídeo do *Boson Treinamentos* a partir do minuto 5:45 ou o *Área Restrita Brasil* a partir do minuto 6:10, que você verá como configurar manualmente o caminho do JDK.

JRE

O **JRE (Java Runtime Environment)** é **ambiente de execução Java**, que contém a **JVM**, a **biblioteca padrão do Java**, a **Classloader**, **ferramentas de configuração** e **plug-in** para o **navegador**. Podemos pensar no **JRE** como um pedaço de software desenvolvido para **executar outro software**. Se você pode nunca ter programado em Java, mas já executou aplicações em

Java no seu computador, você precisou de instalar o **JRE** para abrir esses programas.

O JRE está disponível para download em: <https://www.oracle.com/java/technologies/javase-jre8-downloads.html>.

Todas as versões do JDK já vêm incluído em suas ferramentas o JRE.

Porque usar o JRE?



- JRE contém a biblioteca de classes, a JVM e outros arquivos de suporte para executar uma aplicação desenvolvida em Java. Ele não contém as ferramentas de desenvolvimento Java, tais como: debugger, compilador, etc.
- Ele usa importantes módulos de classe, tais como: math, swingetc, util, lang, awt e biblioteca de execução.
- Se você já executou alguma aplicação Java, então o JRE está instalado no seu sistema.

JVM

As aplicações implementadas em **Java** são compiladas em **bytecode**, que podem ser executados em **qualquer máquina virtual Java (JVM)**, **independente da arquitetura**. Então, podemos entender que a **JVM** é a **engine** que fornece o **ambiente de execução (runtime)** para **código em Java** ou aplicações desenvolvidas em **Java**. Ele **converte o Java bytecode** em **linguagem de máquina** para ser executada no computador. A **JVM é parte do JRE e não é instalada separadamente**.

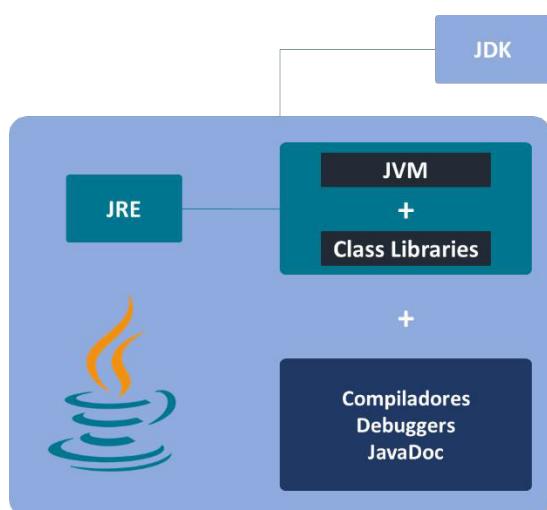
Porque usar o JVM?



- JVM permite executar uma aplicação Java sem depender da plataforma (hardware).
- Ela tem inúmeras bibliotecas, ferramentas e frameworks.
- Uma vez que você executa um programa Java, você pode executá-lo em qualquer plataforma muitas vezes.
- JVM vem com o compilador JIT (Just-in-Time), que converte o código fonte Java em linguagem de baixo nível (linguagem de máquina).

Como o JDK funciona?

Como foi visto, o **JDK** tem as ferramentas necessárias para **implementar** e **executar** programas em **Java**. Veja um diagrama de blocos com alguns recursos do JDK:



Funcionalidade do JDK.

JDK e JRE: O JDK permite programadores criarem programas em Java, que são executados pelo JRE, que inclui a JVM e uma biblioteca de classes.

Class Libraries: é um grupo de biblioteca carregadas dinamicamente, que o programa do Java chama no tempo de execução.

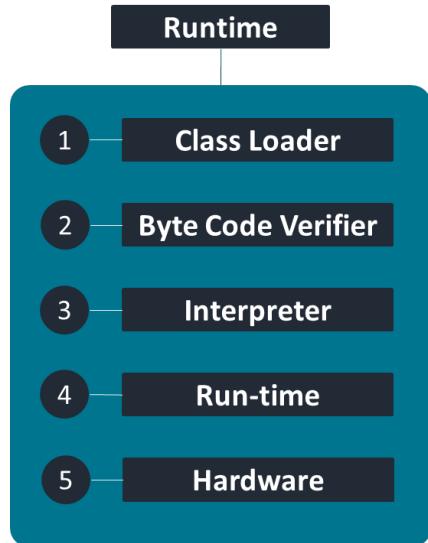
Compiladores: é um programa Java que aceita arquivo de texto (código-fonte) desenvolvidos pelos programadores e compila em um arquivo Java, que geralmente é um arquivo em bytecode.

Debugger: é um programa Java que permite desenvolvedores testar e debuggar (encontrar erros) no código programado.

JavaDoc: é uma documentação criada pela Sun Microsystems pra o Java.

Como o JRE funciona?

Como citado, o **JRE** é responsável por garantir a **execução de uma aplicação Java** no seu computador. Veja os componentes importantes do **JRE**:



Funcionamento do JRE

Class loaders: Class loader carrega várias classes que são necessárias para executar um programa Java. A JVM utiliza três tipos de class loaders: bootstrap class loader, extensions class loader e system class loader.

Byte code verifier: verifica o bytecode (programa compilado), que queremos executar.

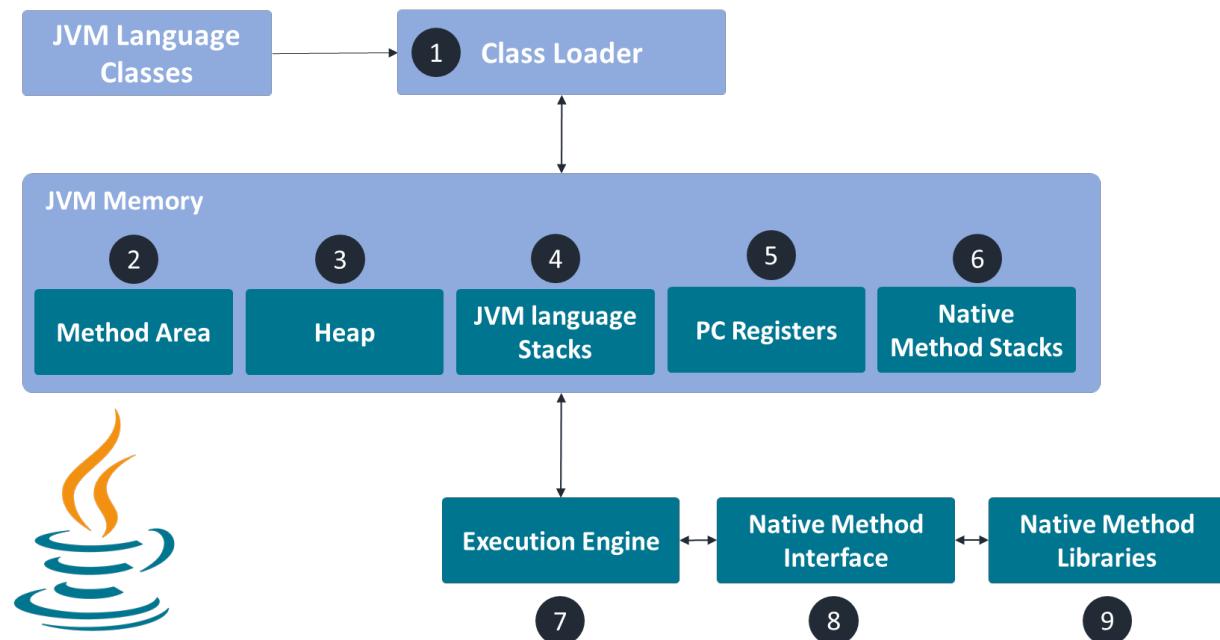
Interpreter (Interpretador): lê linha por linha para executar o programa.

Run-time: é um sistema usado em programação para descrever o período de tempo o qual o programa está executando.

Hardware: é a plataforma para a qual o bytecode foi interpretado, ou seja, é onde o programa irá executar.

Como o JVM funciona?

A imagem a seguir mostra os componentes importantes da **JVM**:



Funcionamento do JVM.

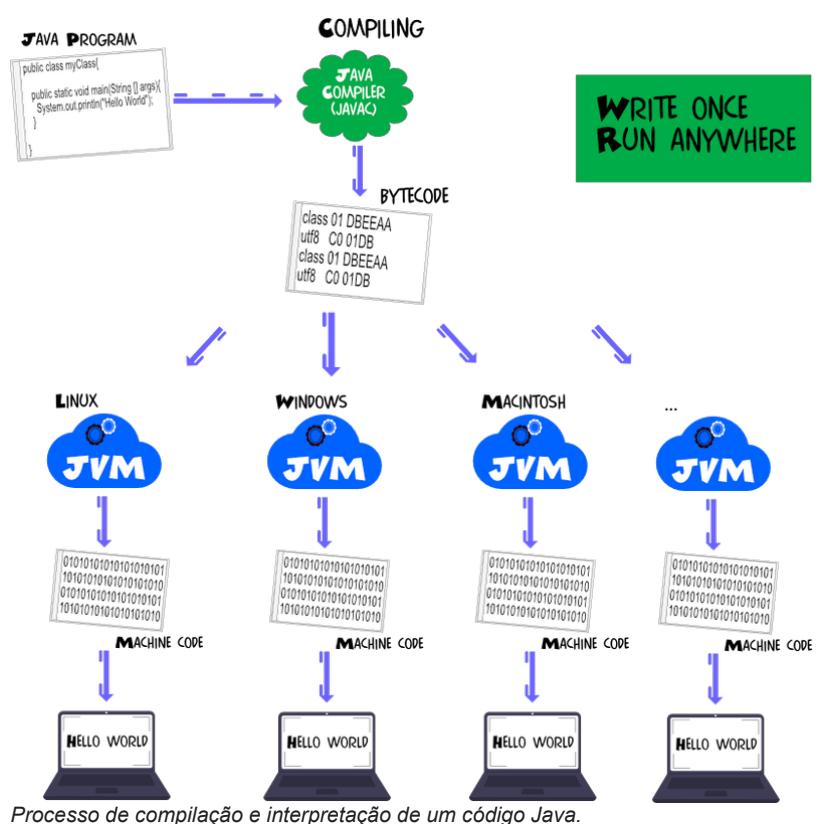
1. **Class Loader**: é um subsistema usado para carregar os arquivos de classes.
2. **Method Area**: JVM Method Area armazena a estrutura das classes, tais como: metadados, o código para os métodos Java e constant pool.
3. **Heap**: todo array de objetos e instância de variáveis são armazenadas na Heap, que é uma parte na memória compartilhada por múltiplas threads.
4. **JVM language Stacks**: armazena variáveis locais e resultados parciais. Cada thread tem sua própria JVM language Stack, que foi criada no momento que a thread é criada.
5. **PC Registers**: armazenam o endereço da instrução, que está sendo executada.
6. **Native Method Stacks**: retêm a instrução do código nativo.
7. **Execution Engine**: é um tipo de, que é usada para testar o software, o hardware ou o sistema completo.
8. **Native Method Interface**: é um framework de programação, que permite o código Java chamar bibliotecas e aplicações nativas, enquanto é executado.
9. **Native Method Libraries**: é uma coleção de bibliotecas nativas (C, C++), que são necessárias para a execução da engine

Compilação e execução do programa Java

A linguagem **Java** é uma linguagem de **alto nível**, que possui uma sintaxe próxima do **inglês** e todo programa feito em Java precisa ser **compilado para a linguagem de máquina**. A linguagem de máquina é a linguagem que os computadores entendem formada de valores **binários (0s e 1s)** e escrever programas nessa linguagem seria **muito complicado**.

Por isso, sugeram as **linguagens de programação**, primeiro vieram as linguagens de **baixo nível**, que estão mais **próximas** da **linguagem de máquina** e geralmente formada de mnemônicos como o **Assembly**. E com o tempo, vieram as **linguagens de programação** de **alto nível**, que é ideal utilizar pois estão mais **próximas do nosso idioma**. Porém usar **linguagens de alto nível requer o uso um tradutor** para estabelecer a **comunicação** entre o **programa** desenvolvido e o **computador**. **Tradutores de linguagem de programação** são também conhecidos como **compiladores** ou **interpretadores**.

O Java é uma linguagem **híbrida** em termos de **compilação**, isso porque ela **possui** um processo de **compilação** e um processo de **interpretação**. A **compilação** é o processo de **traduzir o código na linguagem Java para bytecodes** e a **interpretação** é o processo de **interpretar os bytecodes pela máquina virtual Java (JVM)**. Como o Java é **multiplataforma** o processo de interpretação realizado pela **JVM** é necessário porque **cada sistema operacional precisa ter uma implementação da JVM**.



Introdução à lógica

“Todo mundo deveria aprender a programar um computador porque isso ensina você a pensar.”
Steve Jobs

Lógica pode ser definida com a “**arte de bem pensar**”. Ela está diretamente relacionada à **coerência e racionalidade**, ou seja, está ligado ao que é **correto**. Resumidamente, podemos dizer que lógica é colocar “**ordem no pensamento**”. Quando expressamos pensamentos de

forma **correta**, estamos sendo **lógicos**. Portanto, a lógica se relaciona também com a “**correção do pensamento**”. A lógica, então, determina quais operações **são válidas** ou **não**.

A lógica **não envolve somente números**, ela pode ser um **conjunto de regras racionais para obtenção de um conhecimento**, ou seja, passos a seguir para **aprender algo novo**. A lógica auxilia a formular melhor **as suas ações diante de problemas**. Através da lógica é possível **interpretar um problema** complexo e procurar **entendê-lo melhor**. Em computação o pensamento **lógico é indispensável, sem lógica não há programação**.



Dica!

Para entender melhor um pouco mais sobre lógica, você pode assistir o vídeo do canal Acima da média, disponível no link:

<https://www.youtube.com/watch?v=8kP3vuZgv3k>

Algoritmo

Um **algoritmo** é uma **sequência finita de ações (instruções) encadeadas**, que seguem uma determinada lógica e tem por objetivo **solucionar um problema**. Algoritmos são como uma sequência ordenada de passos, que deve ser seguida para a realização de uma **tarefa**, por exemplo: como ir para a escola, comprar um carro, fazer uma receita de bolo, etc.

Definição: instrução é a informação que indica a um computador uma ação elementar a ser executada.

O exemplo mais clássico de algoritmo é **como fazer um bolo?** Você precisa de diversos ingredientes, tais como: farinha, ovos, manteiga, açúcar, fermento etc., mas o algoritmo te fala como fazer o bolo, ou seja, **qual ordem devemos seguir para ter sucesso na execução do bolo**.

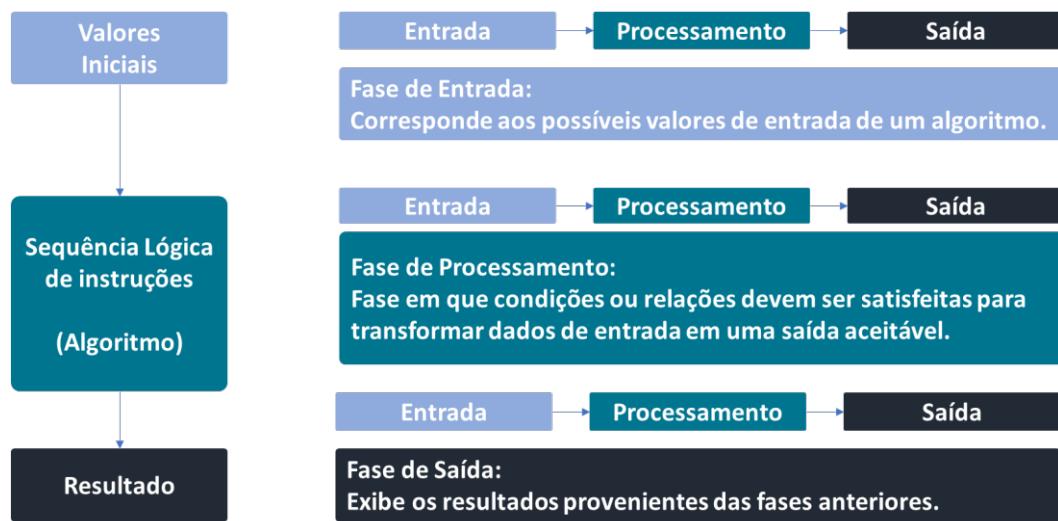
Podemos entender que o algoritmo tem por base o mesmo funcionamento do computador, ou seja, os algoritmos têm uma **estrutura sequencial** como a mostrado abaixo. Essa estrutura sequencial significa que **cada instrução deve ser executada em sequência**. Sendo assim, você tem uma **entrada** que no exemplo abstrato da receita de bolo são os ingredientes, em seguida tem o **processamento**, que seria como o bolo deve ser feito (os passos a serem seguidos), e no **final** da sequência a saída, que é o bolo pronto.



Funcionamento de um algoritmo no computador.

Etapas de um Algoritmo

Um algoritmo possui **três fases**: **Valores iniciais**, **Sequência Lógica de instruções** e **Resultado**. A imagem abaixo, mostra cada uma dessas etapas e descrição do que cada uma corresponde.



Etapas de um algoritmo.



Dica: se você quer saber mais sobre algoritmo, recomendo o vídeo do canal Diolinux, que explica de forma rápida e bem clara o que é um algoritmo. O link está disponível em: <https://www.youtube.com/watch?v=z1XTcKKRbKM>

E se você quiser mais inspiração para começar a programar logo, aconselho assistir o vídeo do Code.org, onde várias personalidades como Bill Gates, Mark Zuckerberg, entre outros falam sobre suas experiências com programação. O vídeo está em inglês, mas possui legendas em português. O link do vídeo é <https://www.youtube.com/watch?v=nKlu9yen5nc>

Glossário

Aqui estão algumas definições e conceitos que você precisa conhecer para entender melhor o que está por trás de um programa em execução no seu computador (Fonte: Wikipédia).

Array: é uma estrutura de dados que armazena uma coleção de elementos de tal forma que cada um dos elementos possa ser identificado por, pelo menos, um índice ou uma chave.

Biblioteca: é uma coleção de subprogramas utilizados no desenvolvimento de software. Bibliotecas contém código e dados auxiliares, que provêm serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular.

Código-fonte: é o conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das linguagens de programação existentes, de maneira lógica.

Compilador: é um programa de computador (ou um grupo de programas) que, a partir de um código fonte escrito em uma linguagem compilada, cria um programa semanticamente equivalente, porém escrito em outra linguagem, código objeto. Classicamente, um compilador traduz um programa de uma linguagem textual facilmente entendida por um ser humano para uma linguagem de máquina, específica para um processador e sistema operacional.

Debug: é o processo de encontrar e reduzir defeitos num aplicativo de software ou mesmo em hardware. Erros de software incluem aqueles que previnem o programa de ser executado e aqueles que produzem um resultado inesperado.

Framework: em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Ao contrário das bibliotecas, é o framework quem dita o fluxo de controle da aplicação, chamado de Inversão de Controle.

Hardware: é um termo técnico que foi traduzido para a língua portuguesa como equipamento, e pode ser definido como um termo geral da língua inglesa, que se refere à parte física de computadores e outros sistemas microeletrônicos.

Heap: em computação, é a forma de gerenciamento de recursos na memória do computador. O gerenciamento de memória envolve entre outros conceitos a liberação de memória e alocação dinâmica de porções de memória para as requisições dos programas.

IDE: do inglês Integrated Development Environment ou Ambiente de Desenvolvimento Integrado, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo. As características e ferramentas mais comuns encontradas nos IDEs são: Editor, Compilador, Linker, Depurador (debugger), Modelagem de dados (modeling), Geração de código, Distribuição (deploy), Testes Automatizados (automated tests), Refatoração (refactoring), entre outras.

Interpretador: é um programa de computador que lê um código fonte de uma linguagem de programação interpretada e o converte em código executável. Seu funcionamento pode variar de acordo com a implementação. Em alguns casos, o interpretador lê o código fonte linha a linha e o converte em código objeto (ou bytecode) à medida que o executa, em outros casos, converte o código fonte por inteiro e depois o executa.

Linguagem de máquina: é a linguagem que o seu computador entende. Um programa em código de máquina consiste em uma sequência de bytes que correspondem a instruções a serem executadas pelo processador. As instruções do processador, chamadas de opcodes, são representadas por valores em hexadecimal.

Linguagem de baixo-nível: trata-se de uma linguagem de programação que segue as características da arquitetura do computador. Assim, utiliza somente instruções que serão executadas pelo processador.

Linguagem de alto-nível: é uma linguagem, que utiliza instruções abstratas. Ela é uma linguagem com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo à linguagem humana.

Registrador: é a memória dentro da própria CPU que armazena n bits. Os registradores estão no topo da hierarquia de memória, sendo assim, é um tipo de memória mais rápida e financeiramente mais custosa. Apesar do alto custo por bit armazenado, sua velocidade de acesso é essencial para o funcionamento dos computadores modernos e, portanto, são incluídos, ainda que em menor capacidade, mesmo em processadores de baixo custo.

Software: trata-se de uma sequência de instruções a serem seguidas e/ou executadas, na manipulação, redirecionamento ou modificação de um dado (informação) ou acontecimento.
Stack (Pilha): é um tipo abstrato de dado e estrutura de dados baseada no princípio de Last In First Out (LIFO), ou seja "o último que entra é o primeiro que sai" caracterizando um empilhamento de dados. Usada para armazenar temporariamente variáveis de uma função.

Thread: é a tarefa que um determinado programa realiza, é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrentialmente.



Ambiente de Desenvolvimento

Os objetivos desta aula são:

- Conhecer as vantagens de utilizar um ambiente de desenvolvimento (IDE);
- Criar um primeiro projeto utilizando o IntelliJ.

Bons estudos!

Ambiente de Desenvolvimento (IDEs)

Existem diversas **IDEs** (**ambiente de desenvolvimento integrado**) para desenvolver seus programas em Linguagens **JVM** (**Java, Kotlin, Scala ou Groovy**). As principais são: **Eclipse**, **Netbeans**, **IntelliJ IDEA**, **BlueJ**, (Oracle) **JDeveloper**, **Xcode**, **Codenvy**, entre outras. As mais populares atualmente (2021) são **Eclipse**, **Netbean** e **IntelliJ IDEA**.



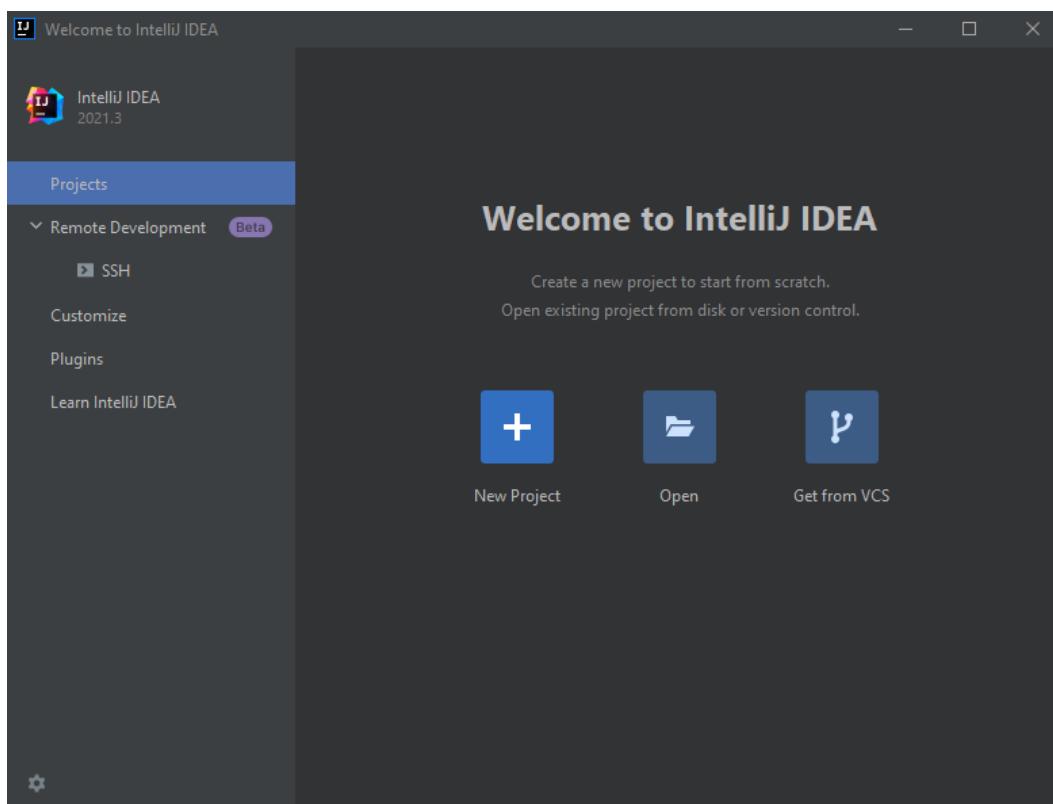
No nosso curso, usaremos o **IntelliJ IDEA**.

O **IntelliJ IDEA** é um **ambiente de desenvolvimento integrado** para linguagem **Java** e foi projetado para maximizar a produtividade da pessoas desenvolvedora. Ela foi desenvolvida pela **JetBrains** e está disponível em **duas versões**: uma **comercial**, que é proprietária, e uma **Community**, que é **gratuita** e possui a licença Apache 2. Ela está disponível para vários **sistemas operacionais**, tais como: **Linux**, **Windows** e **MacOS**. Além das linguagens **JVM** citados acima a **IDE permite desenvolver programas** em linguagens como **Python**, **Ruby**, **PHP**, **SQL**, **Go**, **JavaScript**, **TypeScript** etc. desde que sejam instalados os plugins necessários para trabalhar com essas linguagens.

O arquivo de instalação do IntelliJ IDEA está disponível no site <https://www.jetbrains.com/idea/> e caso você tenha alguma dúvida na instalação pode assistir o vídeo tutorial de instalação do Área Restrita Brasil (<https://www.youtube.com/watch?v=ZMyGkHAcENk>) ou do Bóson Treinamentos (https://www.youtube.com/watch?v=rN_qUZJixKg), que traz a instalação de uma versão mais antiga, mas é bem detalhado na forma de explicar a instalação.

Depois de instalado o **IntelliJ IDEA** no seu computador, você pode abrir o software e uma interface irá aparecer. Nessa interface, você tem algumas opções no menu lateral tais como:

- **Projects**: fornece as opções de criar novo projeto (New Project), abrir um projeto já existente (Open) e buscar projetos de um sistema de versionamento Git (Get from VCS).
- **Customize**: permite a customização da sua IDE, tais como: definir temas de cores, tamanho da fonte do IDE, mapa de caracteres, etc. Apenas para essa apostila, vamos usar o tema IntelliJ Light, pois fica melhor no material impreso.
- **Plugins**: são as extensões disponíveis para instalar na IDE.
- **Learn IntelliJ IDEA**: é um tutorial para aprender a utilizar a IDE.

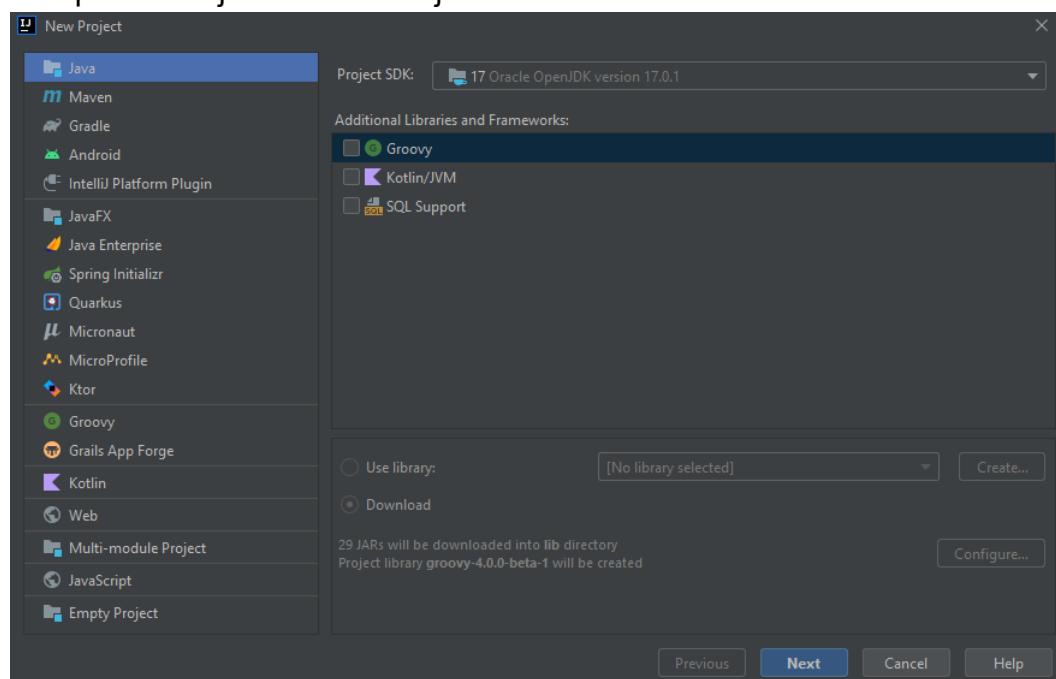


Interface do IntelliJ IDEA.

Criando um projeto em Java

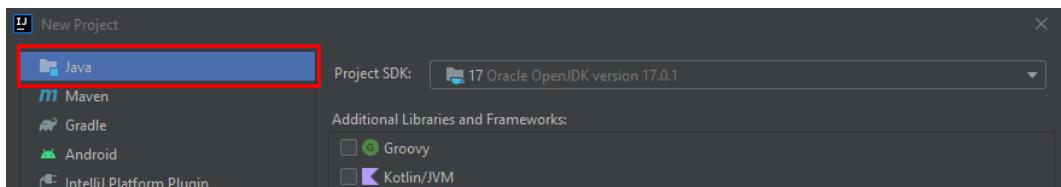
Para criar um **novo projeto**, você deverá seguir os seguintes passos:
Clique no botão **New Project**.

Irá aparecer a janela New Project.

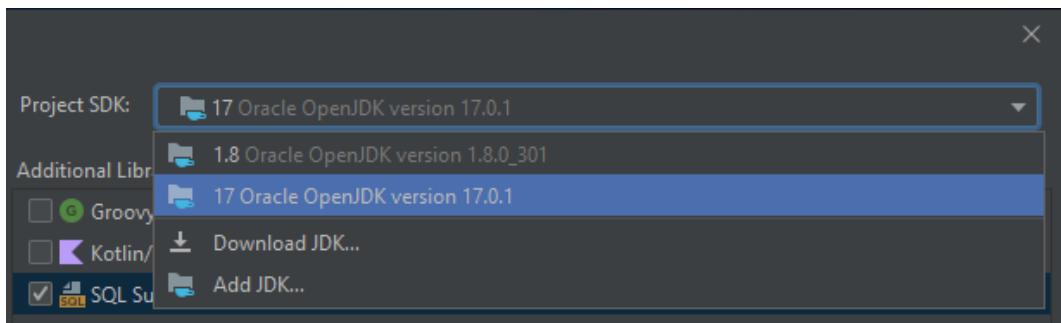


Janela de Novo Projeto.

Na lateral esquerda, destacado de azul, temos as **opções de projetos para serem criados**, no nosso caso criaremos um projeto para programar em **Java**, portanto escolha a opção **Java**.



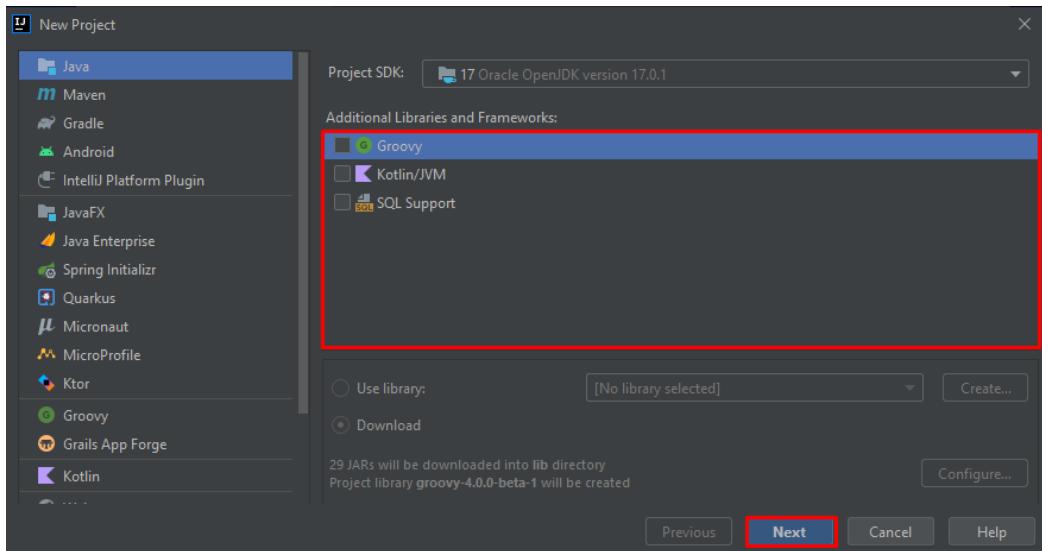
Na parte superior, destacado de vermelho, temos o **JDK**, que foi **instalado anteriormente**. Quando é instalado corretamente, ele **aparece configurado na IDE**. Caso ele não aparecer você pode **fechar a IDE e fazer a instalação** com mostrado nos vídeos indicado no material ou **clicar na seta do campo Project SDK e escolher Download JDK e fazer o download e instalação pela IDE**.



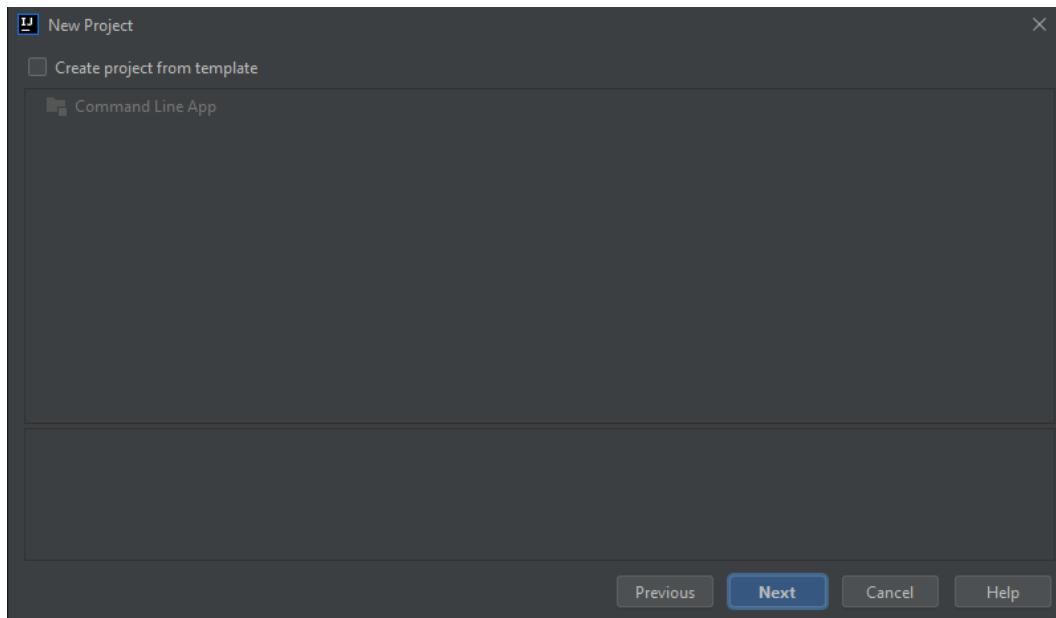
Opções do JDK.

No centro, destacado de vermelho, em **Additional Libraries and Frameworks**, não vamos utilizar **nenhuma biblioteca** ou **framework** adicional no projeto, portanto **não marque nenhuma opção**.

E então clique no botão **Next**.

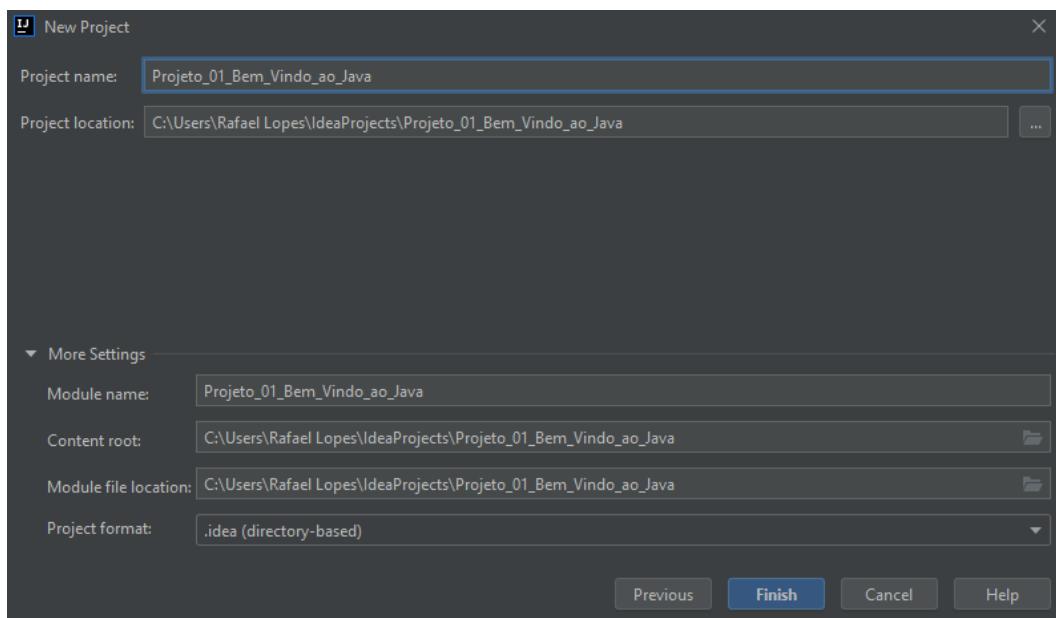


Na janela seguinte, você tem a opção de criar o projeto baseado em algum **template**. Apenas clique no botão **Next**.



Create Project from template.

Na próxima janela, você deverá **atribuir um nome para o projeto**. Se você clicar em **More Settings**, aparecerá **mais opções do projeto**.



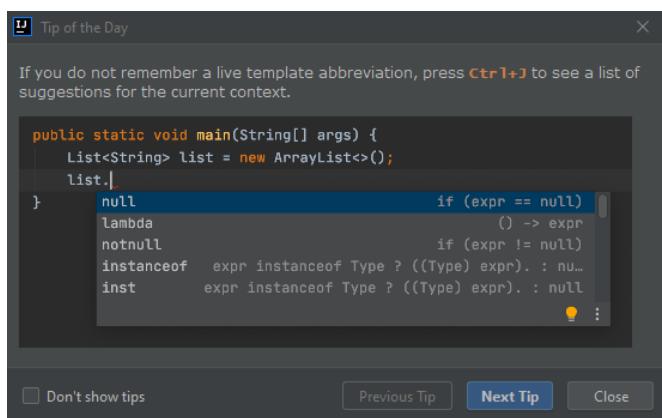
Atribuir um nome para o projeto.



Dica: Sempre coloque um nome representativo no seu projeto, assim facilita caso você precise procurar um determinado código. Evite nomes como Aula_01, Aula_02, pois não dizem o que foi abordado no programa.

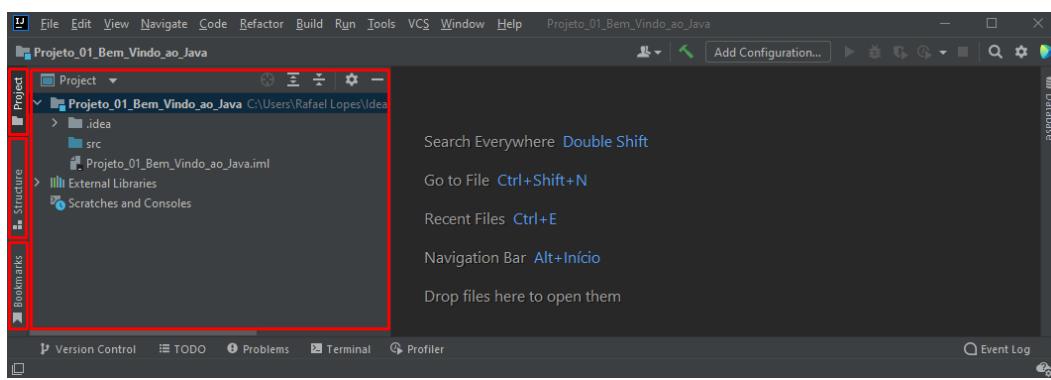
Se você adotar a nomenclatura Projeto_01_nome_do_projeto, Projeto_02_nome_do_projeto, etc. os seus projetos ficarão organizados cronologicamente no diretório. Um dos motivos de fazer isso, é para seus projetos ficarem organizados de acordo a complexidade de assunto, pois uma das formas de aprender programação é começar com programas básicos e ir aumentando a complexidade a cada novo assunto.

Após inserir o **nome do projeto** clique no botão **Finish**. Isso fará com que o **IntelliJ IDEA** faça a configuração automática do projeto. Caso a janela de **Tips (Dicas)** aparecer, você clicar no botão **Close** ou navegar pelas dicas se quiser.



Janela Tip of the Day.

Na tela principal, temos ao lado esquerdo destacado, as abas **Project**, **Structure** e **Favorites**. Na aba **Project**, que é aberto por padrão, temos o diretório **src**, que vem de **source (fonte)**. Nessa pasta é que colocaremos nosso **código-fonte**, ou seja, as **classes** criadas em **Java**.

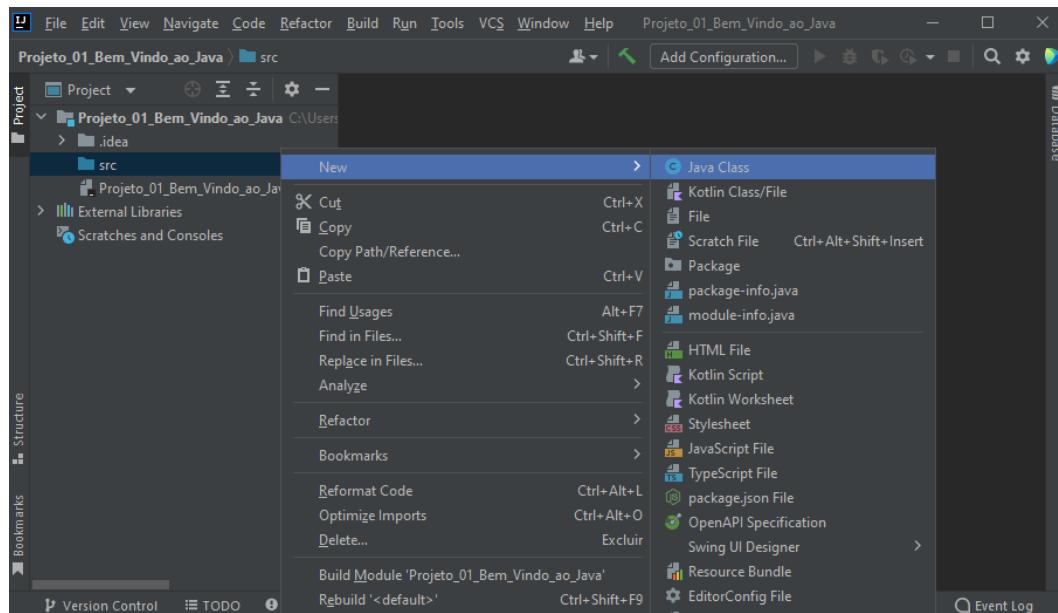


Projeto Java Criado.

Criando a primeira classe

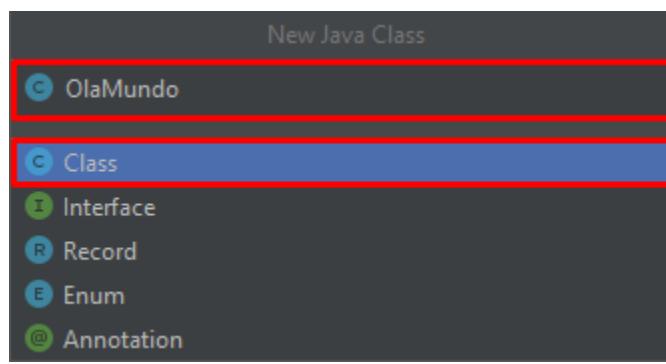
Vamos criar um primeiro programa em **Java**, no momento **não se preocupe com o significado das instruções**, iremos abordar esses detalhes assim que avançarmos no curso. Esse primeiro exemplo é para você aprender os **passos para a criação de um programa em Java**. Portanto, como o projeto criado anteriormente aberto, vamos seguir os passos:

Clique com o botão direito no diretório **src**, seleciona a opção **New** e depois **Java Class**.



Criar um arquivo fonte.

Será solicitado um **nome para sua classe**, que será criado, vamos dar o nome de todo primeiro programa usado em qualquer linguagem de programação: **OlaMundo**. Deixe a **opção de Class selecionada**. Digite o nome no campo destinado para o nome e pressione **Enter**.

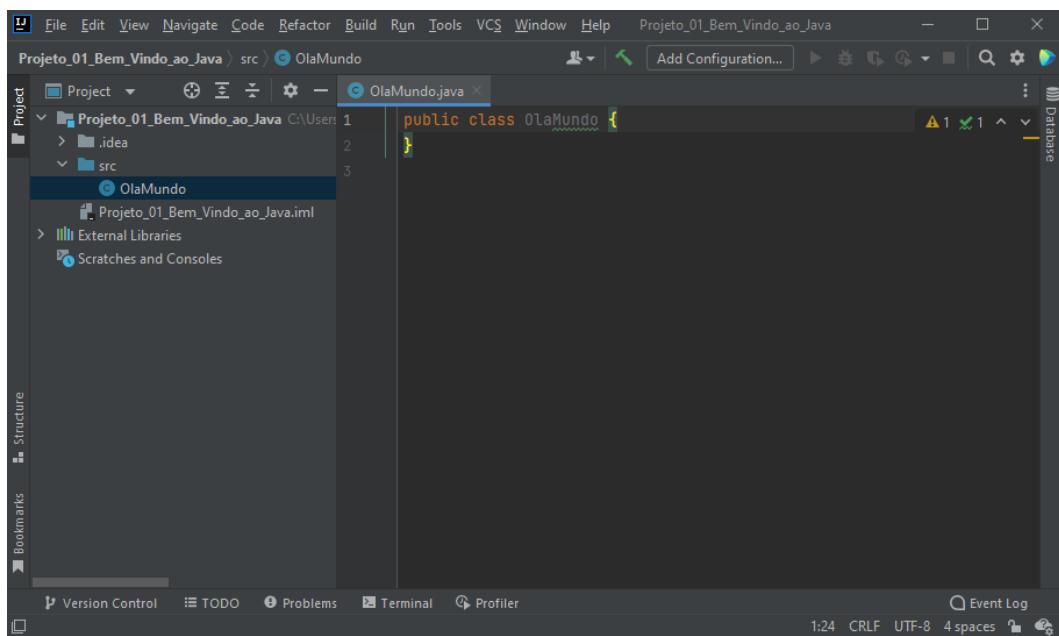


Nome da classe.



Dica: Não é possível criar uma classe com nome utilizando **espaços em branco**. Então caso o nome da classe tenha **duas palavras**, você pode usar a convenção do **camelCase** ou **CamelCase**, que é utilizar a **inicial de cada palavra com letra maiúscula**. No nosso exemplo, poderíamos usar **olaMundo** ou **OlaMundo**.

O arquivo **OlaMundo.java** será **aberto** e se você **expandir o diretório src**, verá o arquivo **anexo ao projeto**. Caso o arquivo não esteja aberto, clique duas vezes nele para abri-lo.



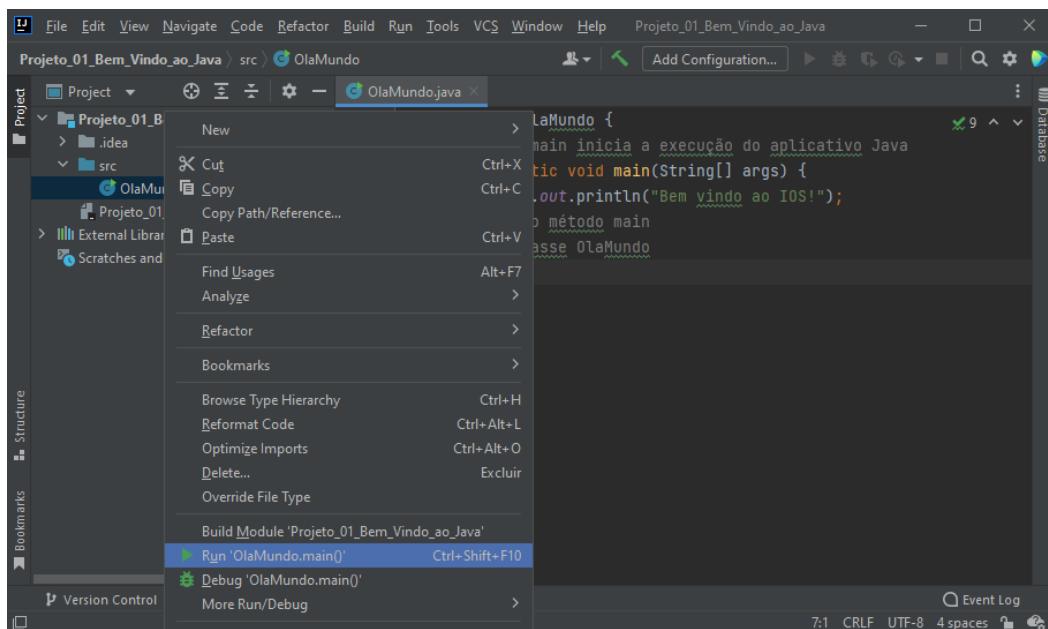
Arquivo **OlaMundo.java** aberto no projeto.

Com o arquivo **criado** e **aberto**, podemos **atualizar o código** no arquivo **OlaMundo.java** com o código mostrado abaixo e salvar o arquivo.

```
1. public class OlaMundo {  
2.     // método main inicia a execução do aplicativo Java  
3.     public static void main(String[] args)  
4.     {  
5.         System.out.println("Bem vindo ao IOS!");  
6.     } // fim do método main  
7. } // fim da classe OlaMundo
```

Primeiro código em Java.

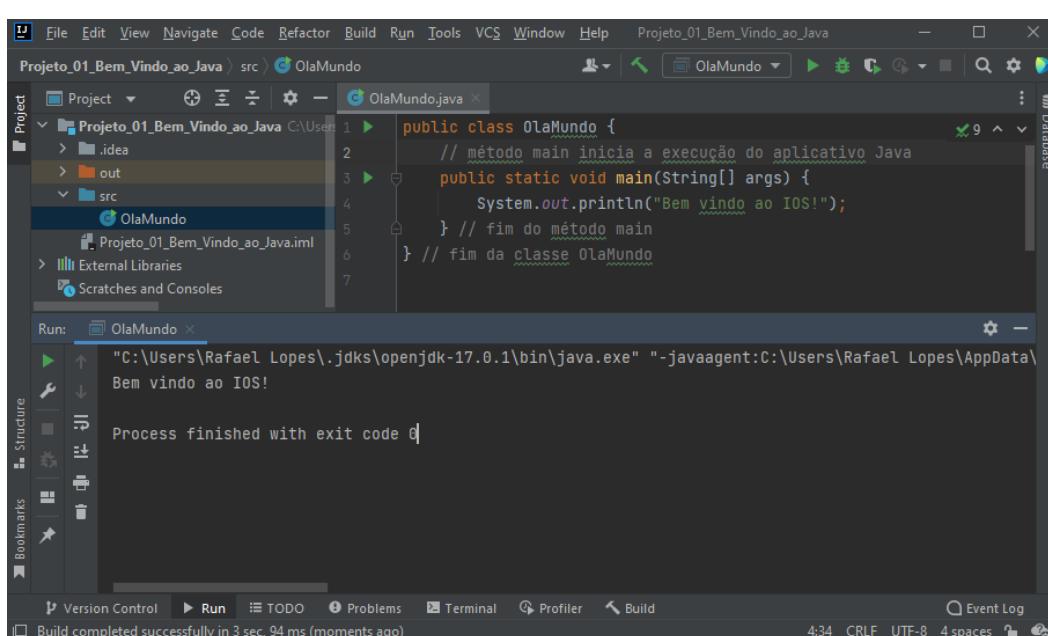
Agora você pode **compilar o código**, para isso você pode clicar com o **botão direito do mouse** no arquivo **OlaMundo.java** e selecionar a opção **Run “OlaMundo.main()”**. Você pode também clique no botão **Run ▶** ou usar o atalho **Ctrl+Shift+F10**.



OlaMundo.java

Opções de compilar o código do arquivo OlaMundo.java

Se tudo estiver configurado corretamente, uma janela na parte inferior da **IDE** irá abrir com a mensagem **Bem vindo ao IOS**, conforme foi programado.



Bem vindo ao IOS!

Process finished with exit code 0

Resultado do código compilado.



Dica: Existe um toolbox do JetBrains para gerenciar todos os programas da empresa que você tem no seu computador. Com ele é possível baixar, atualizar programas e plugins de forma fácil e rápida. O link para baixar o Toolbox é: <https://www.jetbrains.com/toolbox-app/>

Você também pode saber mais sobre o Toolbox assistindo o vídeo: <https://www.youtube.com/watch?v=O-gSdeEbWFo>



Tipos de dados e Operadores

Os objetivos desta aula são:

- Conhecer a sintaxe da linguagem Java;
- Diferenciar algoritmos de descrição narrativa, fluxograma e pseudocódigo;
- Compreender diferentes operadores e sua ordem de precedência.

Bons estudos!

Mais sobre o Java

Como foi dito anteriormente o **Java** é uma linguagem de programação **orientada a objetos**, de **alto nível** e baseada em **classes**, mas ela também pode ser usada para **ensinar programação estruturada**, que é a proposta desse curso. A programação estruturada é um **padrão** ou **paradigma de programação de engenharia de software**, que têm ênfase em **sequência, decisão e iteração** (**sub-rotinas, laços de repetição, condicionais e estruturas em bloco**). Esse tipo de programação é muito comum em linguagens como **C** e **C++**.

Por que estudar Java?

O Java está entre as linguagens de programação mais populares e utilizadas no mundo atual. Hoje, em 2021, **apenas o Python é mais utilizado do que o Java** de acordo com o ranking de **Top Programming Languages 2021** do **IEEE Spectrum**. A popularidade do Java pode ser vista pela **quantidade de vagas disponíveis para profissionais que possuem esse conhecimento**. Além disso, o Java possui uma grande comunidade ativa de **projetos open-source**, por exemplo a **Apache Software Foundation** mantém mais de **200 projetos de Java**, que vão desde ferramentas de build a servidor como o Tomcat.

Outro ponto importante a se destacar é que o **Java está presente em diversos projetos que já estão em produção em diversas empresas**, sendo assim um desenvolvedor precisa entender como **programar Java para criar funcionalidade ou dar manutenção nesses sistemas**. E por fim, o Java é uma linguagem **moderna que está em evolução** e aderente aos novos **paradigmas de programação**.

Por que estudar Java nesse curso?

O ensino de Java na forma de programação estruturada pode ser um grande passo para a familiarização com a linguagem e com estruturas mais simples de programação. No contexto de programação web, o Java está presente no backend de serviços de internet. E para o aluno que deseja continuar a sua formação como programador web, esse conteúdo pode ser o conhecimento fundamental para complementar os seus estudos.

Nosso primeiro programa em Java

Na aula passada, nós fizemos a nossa **primeira classe**. Ela foi o **primeiro passo no estudo de Java estruturado**, mas vamos aprender a função dessas instruções e assim entender o que esse pequeno programa faz.

```

1. public class OlaMundo {
2.     // método main inicia a execução do aplicativo Java
3.     public static void main(String[] args)
4.     {
5.         System.out.println("Bem vindo ao IOS!");
6.     } // fim do método main
7. } // fim da classe OlaMundo

```

Primeiro código em Java.

Declarando uma classe

O **Java** é uma linguagem **baseada em classes**, portanto **todo programa Java** consiste em pelo menos **uma classe que a pessoa programadora define**. Assim, a primeira instrução é a declaração de uma classe:

```
public class OlaMundo
```

A palavra-chave **class** introduz uma **declaração de classe** e é **imediatamente seguida** pelo nome da **classe (OlaMundo)**.



Importante!

Palavras-chave (**às vezes chamadas de palavras reservadas**) são **reservadas** para uso pelo **Java** e sempre escritas com todas as **letras minúsculas**. As palavras-chave do Java são:

Palavras reservadas em Java				
abstract	do	import	return	try
assert	double	instanceof	short	void
boolean	else	int	static	volatile
break	extends	interface	strictfp	while
byte	false	long	super	
case	final	native	switch	
catch	finally	new	synchronized	
char	float	null	this	
class	for	package	throw	
const	goto	private	throws	
continue	if	protected	transiente	
default	implements	public	true	
abstract	do	import	return	
assert	double	instanceof	short	
boolean	else	int	static	
break	extends	interface	strictfp	
byte	false	long	super	

Cada classe que definimos **inicia** com a palavra-chave **public**. Uma classe **public** deve ser inserida em um arquivo com um nome na forma **NomeDaClasse.java**, assim a classe **OlaMundo** é armazenada no arquivo **OlaMundo.java**. **Public** é um **modificador de acesso** e uma declaração com o modificador **public** pode ser **acessada de qualquer lugar** e por **qualquer entidade** que possa **visualizar a classe a que ela pertence**.

Comentários

Comentários são **anotações inseridas no código fonte** com o objetivo de descrever alguma **lógica, instrução** ou um **lembrete TO-DO**. Por exemplo, você pode usar comentários para: **lembrar algo importante quando o código foi desenvolvido**, criar **seções para organização** e ainda para criar um **cabeçalho do código**. Comentários são **ignorados pelo compilador** na verificação da sintaxe do código. Os comentários mais comuns para programadores iniciantes em Java são:

Comentário de linha, que é iniciado por **//**

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

Comentário de bloco, que é iniciado por **/*** e finalizado por ***/**

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

Para **programadores iniciantes**, é interessante **incluir um comentário de linha** depois de um **fechamento de chave** que termina uma declaração de método e após uma chave de fechamento que termina uma declaração de classe. Isso ajuda iniciantes a **visualizarem o início de fim do bloco de comandos de um método ou classe**.

```
} // fim do método main
} // fim da classe OlaMundo
```

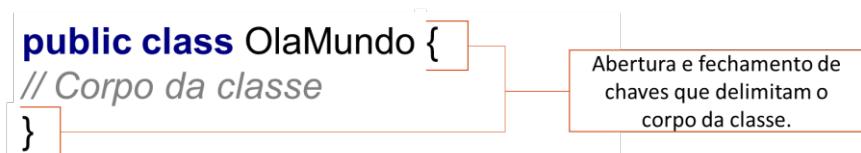
O Java fornece comentários de um **terceiro tipo**: comentários **Javadoc**. Esses são delimitados por **/**** e ***/**. Os comentários no estilo **Javadoc** permitem **incorporar a documentação do programa** diretamente aos **seus programas**. O programa utilitário **javadoc** (parte do **JDK**) lê comentários **Javadoc** e os usa para **preparar a documentação do programa no formato HTML**.



Dica: Algumas organizações exigem que todo programa comece com um comentário que informa o objetivo e o autor dele, a data e a hora em que foi modificado pela última vez.

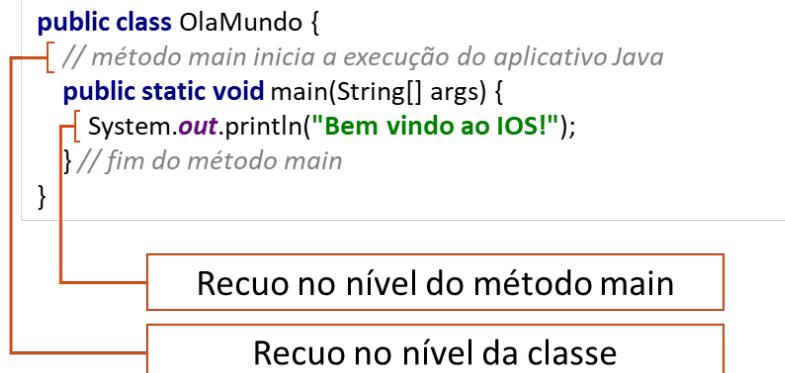
Corpo da classe

O corpo de uma classe, que pode também ser chamado de **bloco de comandos da classe**, é definido pela **abertura** e **fechamento** de **chaves** logo após a **declaração da classe**. Tudo que está dentro dessas chaves **faz parte do código da classe**.



Dica: Faça do **reculo de parágrafo** para dar **legibilidade** a seu código e ficar bem claro a **hierarquia de níveis do programa**. É comum dois ou quatro espaços. Qualquer que seja o estilo que você escolher, utilize-o de modo consistente.

```
public class OlaMundo {  
    // método main inicia a execução do aplicativo Java  
    public static void main(String[] args) {  
        System.out.println("Bem vindo ao IOS!");  
    } // fim do método main  
}
```



A diagram illustrating code indentation. It shows the code above with indentation levels indicated by orange brackets. One bracket groups the entire class definition. Another bracket groups the `main` method definition. Callout boxes point to these brackets with the text: "Reculo no nível da classe" (for the class level) and "Reculo no nível do método main" (for the method level).

Declarando um método

Um **método em Java** é equivalente a uma **função**, **sub-rotina** ou **procedimento** em outras linguagens de programação.

A **instrução**:

```
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**. Desse modo, o comando

```
System.out.println("Bem vindo ao IOS!");
```

instrui o computador para **imprimir** na tela a **string “Bem vindo ao IOS!”**, que foi passado **entre aspas duplas**. Uma string é uma **sequência de caracteres** e 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**. Toda instrução em **Java** deve terminar com **ponto-e-vírgula**.

Comando de saída

A **saída** de um programa é o **resultado do seu processamento**, pode ser armazenando em uma **variável**, um **banco de dados** ou **exibição na tela**. Em Java, **System.out.println()** é uma instrução para **imprimir o argumento passado dentro dos parênteses**. O método **println()** exibe o **resultado na saída padrão (monitor)** e realiza a **quebra de linha**, isto é, o cursor vai para a **próxima linha da tela**. **System** é o nome de uma **classe em Java** e **out** é uma **instância** dessa **classe**.

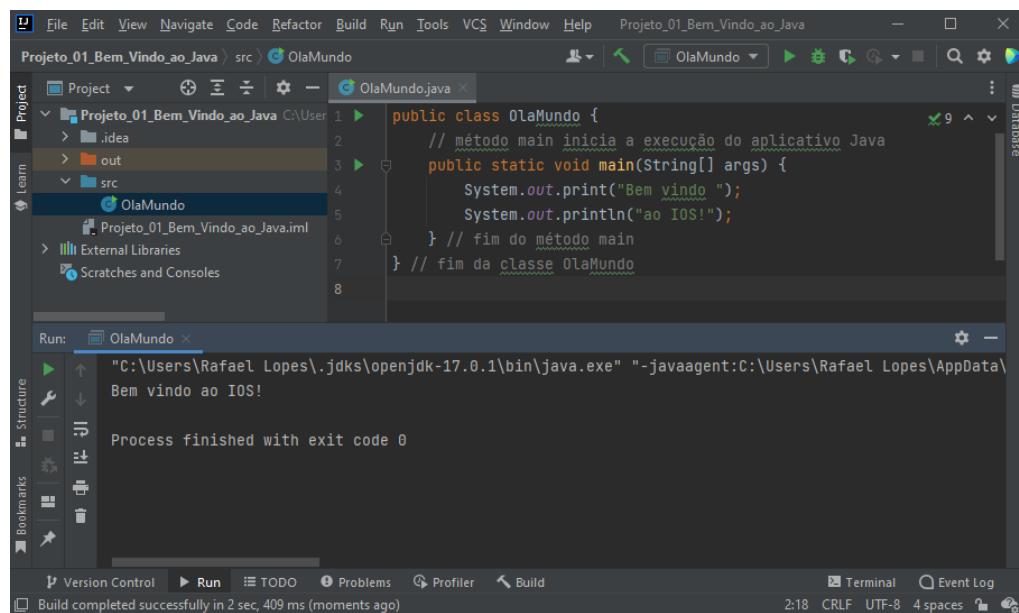
Modificando nosso primeiro programa

Vamos utilizar o método **print()** para **imprimir uma string e sem quebra de linha**. Você pode abrir o projeto anterior e **modificar o código** para ficar igual ao mostrado a seguir:

```
1. public class OlaMundo {
2.     // método main inicia a execução do aplicativo Java
3.     public static void main(String[] args) {
4.         System.out.print("Bem vindo ");
5.         System.out.println("ao IOS!");
6.     } // fim do método main
7. } // fim da classe OlaMundo
```

Código Java modificado.

Com o seu **código atualizado e salvo**, você pode **compilar e executar o código** acessando o botão **Run** e ver o resultado **impresso no console**. A imagem a seguir **mostra** que o **resultado** é o **mesmo do código anterior**. Isso porque o método **print()** não provoca a **quebra de linha** e também tomamos o cuidado para **colocar um espaço no final da primeira string** para não apresentar o **resultado todo junto**.



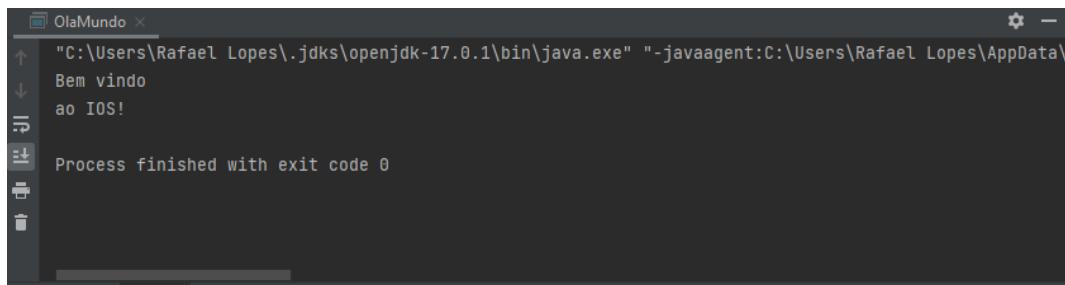
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** Shows "Projeto_01_Bem_Vindo_ao_Java > src > OlaMundo".
- Toolbars:** Standard Java development tools like File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help.
- Left Sidebar:** Shows the project structure with "Projeto_01_Bem_Vindo_ao_Java" expanded, containing ".idea", ".out", "src", and "OlaMundo". "OlaMundo" contains "OlaMundo.java".
- Code Editor:** Displays the Java code for "OlaMundo.java" with the modifications shown above.
- Terminal:** Shows the command run: "C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" "-javaagent:C:\Users\Rafael Lopes\AppData\Local\Temp\IDEA145\javaagent.jar" "Projeto_01_Bem_Vindo_ao_Java" and the output: "Bem vindo ao IOS!" followed by "Process finished with exit code 0".
- Bottom Status Bar:** Shows "Build completed successfully in 2 sec, 409 ms (moments ago)" and "2:18 CRLF UTF-8 4 spaces".

Resultado do código modificado compilado.

Se usarmos o método **println()** nas **duas instruções de saída de dados** como mostrado a seguir, o resultado é as **strings** impressas em **linhas distintas**:

```
1. public class OlaMundo {  
2.     // método main inicia a execução do aplicativo Java  
3.     public static void main(String[] args) {  
4.         System.out.println("Bem vindo ");  
5.         System.out.println("ao IOS!");  
6.     } // fim do método main  
7. } // fim da classe OlaMundo
```



Método **println()** nas duas instruções de saída de dados e resultado da compilação e execução do código.

Representações de Algoritmos (Descrição narrativa, Fluxograma e Pseudocódigo).

Descrição narrativa

Existem várias formas de **representar** um **algoritmo**, entre elas podemos citar: **descrição narrativa**, **fluxograma** e **algoritmo estruturado**. A descrição narrativa é usar uma **linguagem natural** (ex. **português**) para escrever um **enunciado** e os **passos a serem seguidos para a resolução do problema**. O algoritmo a seguir, mostra a **descrição narrativa** de um programa para **receber dois números, realizar a multiplicação** desses números e **exibir o resultado na tela**.

Multiplicação de dois números:

- Passo 1: Receber os dois números que serão multiplicados
- Passo 2: Multiplicar os números
- Passo 3: Mostrar o resultado obtido na multiplicação

Descrição narrativa do algoritmo

Fluxograma

O **Fluxograma** é um **tipo de diagrama**, que representa **esquematicamente** um **processo** ou um **algoritmo**. Muitas vezes, ele construído através de **gráficos** que **ilustram** de forma **descomplicada** a transição de informações entre os elementos que o compõem. O fluxograma

utiliza alguns **símbolos padrões** que possuem um **significado dentro do diagrama**. Veja os símbolos mais comuns de um fluxograma para descrever um algoritmo em programação.

Símbolo	Significado	Descrição
	Terminal	Representa o início ou o fim de um fluxo lógico. Em alguns casos definem as sub-rotinas.
	Entrada manual	Determina a entrada manual dos dados, geralmente através de um teclado.
	Processamento	Representa a execução de ações de processamento.
	Exibição	Mostra o resultado de uma ação, geralmente através da tela do computador.
	Decisão	Representa os desvios condicionais nas operações de tomada de decisão e laços condicionais para a repetição de alguns trechos do programa.
	Preparação	Representa a execução de um laço incondicional que permite a modificação de instruções do laço.
	Processo predefinido	Define um grupo de operações relacionadas a uma sub-rotina.
	Conector	Representa pontos de conexão entre trechos de programas, que podem ser apontados para outras partes do diagrama de bloco.
	Linha	Representa os vínculos existentes entre os símbolos de um diagrama de blocos

Símbolos comuns do fluxograma.

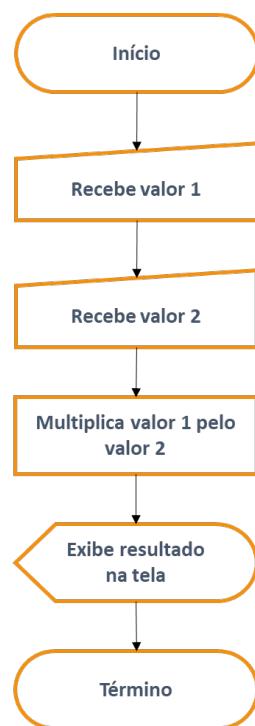
Voltando ao nosso exemplo de **multiplicação de dois números**, veja a **representação em fluxograma do programa**.

Inicialmente, temos a **entrada dos dois valores**, em seguida o **processamento da multiplicação** e no **final a exibição na tela**.

O **algoritmo estruturado**, também conhecido como **pseudocódigo**, é uma maneira intermediária entre a linguagem natural, no nosso caso o **Português** e uma linguagem de programação para **representar um algoritmo computacional**. Os pseudocódigos são estruturados de forma que qualquer pessoa que programe, consiga **reproduzi-lo na linguagem de programação** escolhida.

Voltando ao nosso exemplo de multiplicação de dois números, a mostra a representação em algoritmo estruturado.

Veja o **pseudocódigo** para o programa de multiplicação de dois números. Inicialmente, temos a **entrada** dos dois valores, em seguida o **processamento** da multiplicação e no final a **exibição** na tela.



ALGORITMO

```

DECLARE N1,N2,M NUMÉRICO
ESCREVA "Digite dois Digitos"
LEIA N1, N2
M ← N1 * N2
ESCREVA "Multiplicação = ", M
FIM DO ALGORITMO
  
```

Pseudocódigo do algoritmo de multiplicação de dois números.



Importante!

Algoritmo estruturado não é a mesma coisa do que portugol. O **algoritmo estruturado** é uma representação descrição em forma de pseudocódigo de um programa. Ele é importante na documentação de projetos open-source, em artigos científicos, etc. O **portugol** é uma linguagem usada para fins didáticos e serve para ensinar lógica de programação. **Portugol não tem utilização na indústria.**

Tanto na **descrição narrativa**, **fluxograma** e **pseudocódigo** pode um ou mais caminhos para serem percorridos. Veja abaixo a descrição **narrativa**, o **fluxograma** e o **pseudocódigo** para um programa de **divisão de números**.

Descrição narrativa

Divisão de dois números:

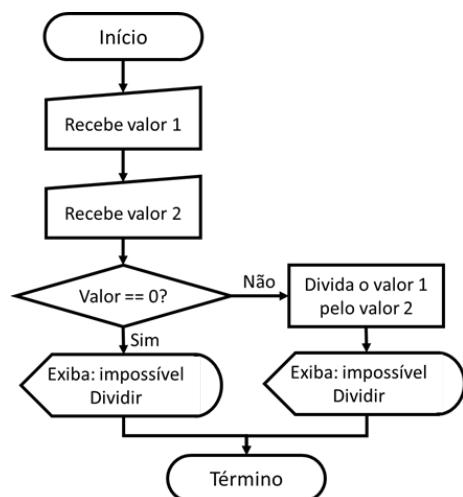
Passo 1: Receber os dois números que serão divididos

Passo 2: Verificar se o segundo número é igual a zero

Passo 3: Se segundo número não for igual a zero, a divisão poderá acontecer, caso contrário a divisão não pode ser realizada.

Passo 4: Exibir resultado da se possível, caso não exibir que não possível dividir

Fluxograma



Pseudocódigo

ALGORITMO

```

DECLARE N1, D2, D NUMÉRICO
ESCREVA "Digite dois números:"
LEIA N1, N2
SE N2 = 0 ENTÃO
  ESCREVA "Impossível dividir."
SENÃO
  INÍCIO
    D ← N1/N2
    ESCREVA "Divisão = ", D
  FIM
FIM ALGORITMO
  
```

Constantes e tipos de dados

Constantes são **valores fixos** em um **algoritmo** e podem ser classificados como **literal** (**caractere** ou **sequência de caracteres**), **numérico** (**inteiro** ou **real**) ou **lógico** (**verdadeiro** ou **falso**). Já o **tipo de dado** indica a **capacidade** e o tipo de **conteúdo** de um **valor** ou **constante**. Veja os principais **tipos de dados do Java**, bem como a sua **Descrição**:

Tipos de dados		Definição
literal – Também conhecido como texto ou caractere.	char/String	Poderá receber letras, números e símbolos.
inteiro	int	Poderá receber números inteiros positivos ou negativos.
real	float/double	Poderá receber números reais, isso é, com casas decimais, positivos ou negativos.
lógico	boolean	Poderá receber verdadeiro (1) ou falso (0).

Tipos de dados no Java

Vamos ver alguns exemplos e revisar esse assunto:

false	lógico	boolean
“O resultado é:”	literal	String
“true”	literal	String
21	inteiro	int
3.1415	real	double
‘h’	literal	char

Operadores

Os operadores **específam** uma **avaliação/ação** a ser executada em **um** ou **mais operandos** e podem ser **aritméticos**, **relacionais** ou **lógicos**. É através de operandos que os programas em **Java executam cálculos aritméticos, lógicos e relacionais**.

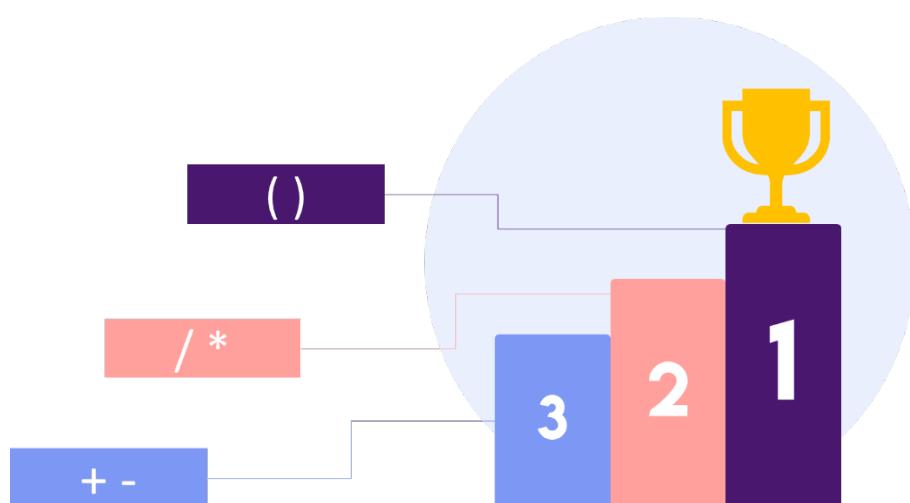
Operadores aritméticos

Os operadores aritméticos são o conjunto de símbolos que representam as **operações básicas da matemática**. No Java, os operadores aritméticos são:

Operação	Operador	Expressão	Resultado
Adição	+	6+4	10
Subtração	-	7-9	-2
Multiplicação	*	12*3	36
Divisão	/	44/2	22
Módulo (Resto da divisão)	%	10%3	1

Operadores aritméticos.

A ordem de precedência dos operandos aritméticos é:



Desse modo a operação:

3 * 5 + 2

O programa primeiro faz a multiplicação e com o resultado dessa multiplicação realiza a soma. Os parênteses têm a maior precedência, portanto a operação:

3 * (5 + 2)

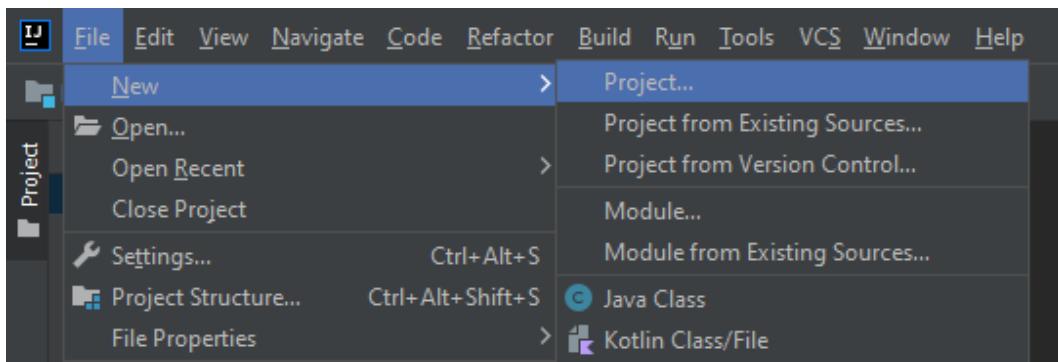
Primeiro é realizado a soma dentro dos parênteses e com o resultado dessa soma realiza a multiplicação.



Vamos praticar!

Siga os passos para escrever o programa utilizando **operadores aritméticos**:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.



Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_02_Oper_Arit**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **OperArit** e insira o código abaixo.

```

1. public class OperArit {
2.     // método main inicia a execução do aplicativo
3.     public static void main(String[] args)
4.     {
5.         System.out.println("Precedência: " + (10/2+3));
6.         System.out.println("Alterando a Precedência: " + 10/(2+3));
7.
8.         System.out.println("Divisão inteira: " + 9/2);
9.         System.out.println("Divisão real: " + 9/2.0);
10.        System.out.println("Divisão real: " + 9.0/2);
11.        System.out.println("Divisão real: " + 9.0/2.0);
12.
13.        System.out.println("Resto da divisão inteira: " + 9%2);
14.        System.out.println("Resto da divisão inteira: " + 8%2);
15.        System.out.println("Resto da divisão inteira: " + 9%5);
16.    } // fim do método main
17. } // fim da classe OperArit

```

Código de operadores aritméticos.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

```
Run: OpenArit ×
C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users\Thiago\IdeaProjects\OperArit\target\javaagent\OperArit.jar
Precedência: 8
Alterando a Precedência: 2
Divisão inteira: 4
Divisão real: 4.5
Divisão real: 4.5
Divisão real: 4.5
Resto da divisão inteira: 1
Resto da divisão inteira: 0
Resto da divisão inteira: 4

Process finished with exit code 0
```

Version Control Run TODO Problems Terminal Profiler Build

Build completed successfully in 4 sec, 799 ms (a minute ago)

Resultado da execução do código.

A instrução na **linha 5** da mostra que a **divisão tem precedência em relação à soma**, pois o resultado impresso na tela **foi 8**, ou seja, $10/2 = 5 + 3 = 8$. Os **parênteses** do lado de fora da operação são necessários para que o **Java entenda o símbolo + é uma soma e não uma concatenação de strings**. Já a instrução na linha 6 da mostra como os **parênteses podem alterar a precedência da operação**.

Importante!



Em programação, **concatenar** é um termo usado para indicar a ação de **unir duas ou mais strings**. Por exemplo, considerando as strings "**casa**" e "**mento**" a concatenação da primeira com a segunda gera a string "**casamento**".

Na linha 8, você pode ver a instrução que realiza a **divisão inteira** entre **dois números inteiros**. **Divisão inteira** é aquela que **resulta um número inteiro se casas decimais**. Caso você deseje que o resultado seja **real**, pelo menos um dos **dois números** deve ser escrito com o **ponto decimal** como mostrado nas instruções seguintes.

Importante!



Em programação, **números reais**, que são **números com casas decimais**, são representados com **ponto decimal** e **não com vírgula** como estamos acostumados no Brasil.

Nas linhas **13, 14 e 15**, você pode observar como o **operador de resto da divisão**. O operador **%** retorna o **resto de uma divisão inteira**, por exemplo, a **divisão inteira** de **5** por **3** tem quociente **1** e resto **2**.

Operadores relacionais

Os **operadores relacionais** são utilizados na **realização de comparação** entre valores do mesmo **tipo primitivo**, ou seja, eles podem ser representados por **variáveis, constantes** e até mesmo em **expressões aritméticas**. No Java, os operadores relacionais são:

Operador	Representação	Exemplo
Maior que	>	$a > b$: Se o valor de a for MAIOR do que o valor de b , retorne TRUE. Senão, retornará FALSE.
Maior ou igual a	\geq	$a \geq b$: Se o valor de a for MAIOR OU IGUAL do que o valor de b , retorne TRUE. Senão, retornará FALSE.
Menor que	<	$a < b$: Se o valor de a for MENOR do que o valor de b , retorne TRUE. Senão, retornará FALSE.
Menor ou igual a	\leq	$a \leq b$: Se o valor de a for MENOR OU IGUAL do que o valor de b , retorne TRUE. Senão, retornará FALSE.
Igual a	$=$	$a == b$: Se o valor de a for IGUAL do que o valor de b , retorne TRUE. Senão, retornará FALSE.
Diferente de	\neq	$a != b$: Se o valor de a for DIFERENTE do que o valor de b , retorne TRUE. Senão, retornará FALSE.

Operadores relacionais.



Vamos praticar!

Siga os passos para escrever o programa utilizando operadores relacionais:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_03_Oper_Relac**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **OperRelac** e insira o código abaixo.

```

1. public class OperRelac {
2.     // método main inicia a execução do aplicativo
3.     public static void main(String[] args)
4.     {
5.         System.out.println("Teste1: " + (3==2));
6.         System.out.println("Teste2: " + (3!=2));
7.         System.out.println("Teste3: " + (3>=2));
8.         System.out.println("Teste4: " + (3<=2));
9.     } // fim do método main
10. } // fim da classe OperRelac

```

Código de operadores relacionais.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

```
Run: OpenArit x
▶ ↑ "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users\...
Teste1: false
Teste2: true
Teste3: true
Teste4: false

Process finished with exit code 0
```

Version Control Run TODO Problems Terminal Profiler Build

Build completed successfully in 4 sec, 433 ms (a minute ago)

Resultado da execução do código.

O resultado de uma operação utilizando operadores relacionais é um tipo **booleano**: **true** ou **false**.

Operadores Lógicos

Seguindo o raciocínio lógica **booleana** utilizamos os **operadores lógicos** para representar situações que **não são representadas por operadores aritméticos**. O resultado dessas expressões é sempre um **valor lógico**: **true (verdadeiro)** ou **false (falso)**. Operadores lógicos são usados em programação para **concatenar expressões** que estabelecem uma **relação de comparação entre valores**.

Sempre utilizamos uma **tabela verdade** para **representar todas as possíveis combinações** das **variáveis** de **entrada** de uma **função** e os seus respectivos **valores de saída**.

No Java, os operadores lógicos são:

Operador E (&&)

Somente resulta em verdadeiro (true), se todas as expressões condicionais forem true.

Operador Ou (||)

Se pelo menos uma expressão condicional for true, o resultado é verdadeiro (true).

Operador Não (!)

Se a expressão condicional for false, o resultado é true.

Se a expressão condicional for true, o resultado é false.

A tabela verdade para os operadores **&&** e **||** é

- &&** → Somente resulta em verdadeiro quando todas as sentenças avaliadas forem verdadeiras;
- ||** → Somente resulta em falso quando todas as sentenças avaliadas forem falsas.
- !** → O operador ! faz a negação de uma sentença

Exemplo:

A	B	&&		!A	!B
V	V	V	V	F	F
V	F	V	V	F	V
F	V	V	V	V	F
F	F	V	F	V	V



Vamos praticar!

Siga os passos para escrever o programa utilizando operadores relacionais:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_04_Oper_Log**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **OperLog** e insira o código abaixo.

```

1. public class OperLog {
2.     // método main inicia a execução do aplicativo
3.     public static void main(String[] args)
4.     {
5.         System.out.println("Teste1: " + (true && false));
6.         System.out.println("Teste2: " + (true || false));
7.         System.out.println("Teste3: " + (!true));
8.     } // fim do método main
9. } // fim da classe OperLog

```

Código de operadores lógicos.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

```
Run: [ ] OperArit ×
▶ ⬆ "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users
Teste1: false
Teste2: true
Teste3: false
[ ] ⏪
[ ] 🖨
[ ] 📁
[ ] 🗑
[ ] 🔍
Process finished with exit code 0
```

Version Control Run TODO Problems Terminal Profiler Build

Build completed successfully in 2 sec, 607 ms (moments ago)

Resultado da execução do código.



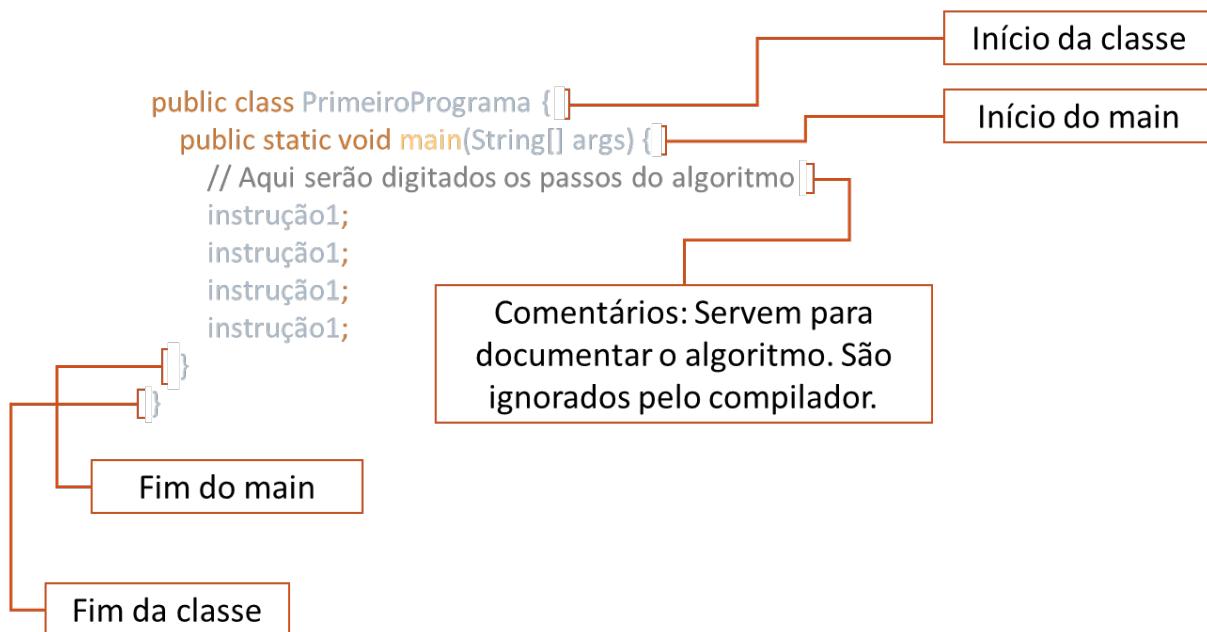
Os objetivos desta aula são:

- Conhecer as estruturas do comando if;
- Compreender os desvios condicionais.

Bons estudos!

Estrutura de um programa em Java

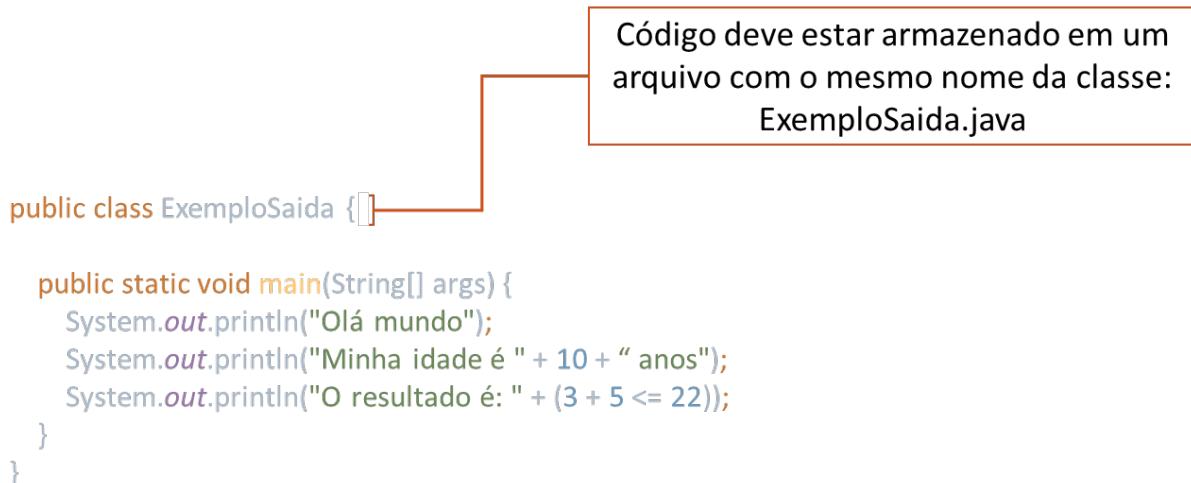
O **Java** é uma **linguagem orientada a objeto**, isto é, todo o código de um programa deve estar definido dentro de uma **classe**. Além disso, o código deve estar definido dentro de um **método especial chamado main()**. O método **main()** é o **ponto de partida** na **execução de um programa** em Java.



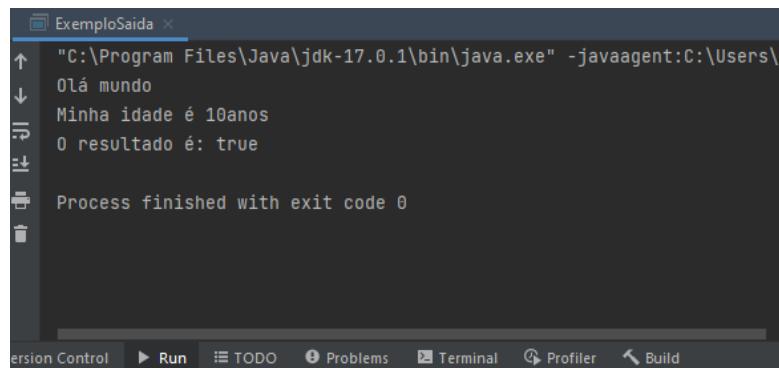
No **início do aprendizado** de uma linguagem de programação, geralmente, o **resultado do processamento** de um algoritmo pode ser **armazenado em uma variável** e/ou pode ser **exibido diretamente em um dispositivo de saída**. O monitor (**console**) é o dispositivo de **saída padrão** de um projeto em **Java**. Como foi visto o comando de **saída** utilizado para escrever dados e resultados na saída padrão (**monitor/console**) é:

```
System.out.println("");
```

Por exemplo, o código abaixo: bloco



Produc o seguinte resultado mostrado na tela do computador.



```
ExemploSaída ×
↑ "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users\↓
↓ Olá mundo
Minha idade é 10anos
0 resultado é: true
↓
↓ Process finished with exit code 0
↓

Version Control Run TODO Problems Terminal Profiler Build
```

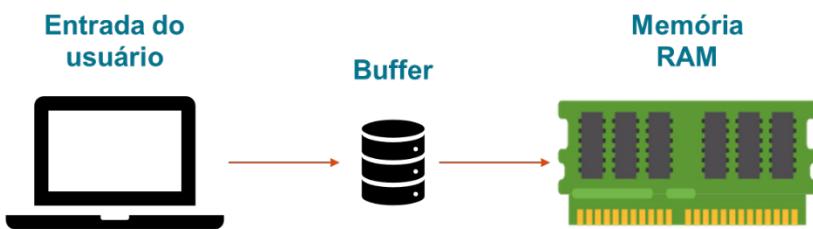
Comando de entrada

Além de **imprimir os dados e resultado** na **saída padrão** do computador, podemos também **ler dados digitado de um teclado** e armazená-los na **memória do computador** por meio de **variáveis**. Para isso, é necessário criar um **buffer** para **guardar os dados digitados** no teclado de depois **transferi-los para a memória (variável)**.



Saiba mais!

Buffer é uma região de memória utilizada para armazenar temporariamente os dados.



Para criar o **buffer** é necessário utilizar a classe **Scanner**:

```
Scanner entrada = new Scanner(System.in);
```

É o buffer do teclado

Após criar o **buffer** do teclado basta **transferir esse dado para a variável**. A leitura de um dado é realizada associando o **tipo de entrada** ao **tipo da variável** que receberá o dado.

Tipo de dado	Usar
String	entrada.nextLine();
Int	entrada.nextInt();
Double	entrada.nextDouble();
Float	entrada.nextFloat();
Char	entrada.next().charAt(0);
Boolean	entrada.nextBoolean();

Para usar a classe **Scanner** é necessário fazer um **import de um pacote antes da definição da classe.**

```
import java.util.Scanner;

public class Exemplo {
    public static void main(String[] args) {
        Scanner exemplo = new Scanner(System.in);
    }
}
```

Vamos fazer um programa em que **solicitaremos** para o usuário que informe o **nome** e o **valor** de **duas notas de um aluno**. O programa deverá **calcular a média dessas notas** e depois **imprimir a média calculada**. Nesse exemplo, precisaremos de **três variáveis duas** para **armazenar as notas digitadas e outra para armazenar a média calculada**. Portanto, siga os passos para escrever o nosso programa para calcular a média das notas:



Vamos praticar!

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

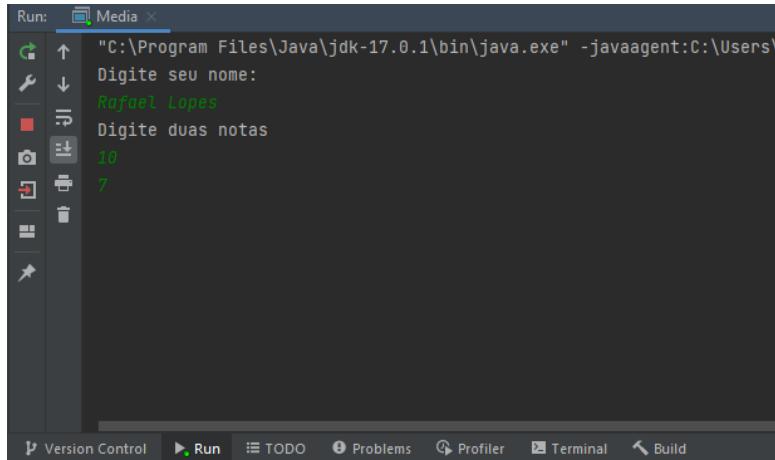
Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_05_Commando_Estrada**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Media** e insira o código abaixo.

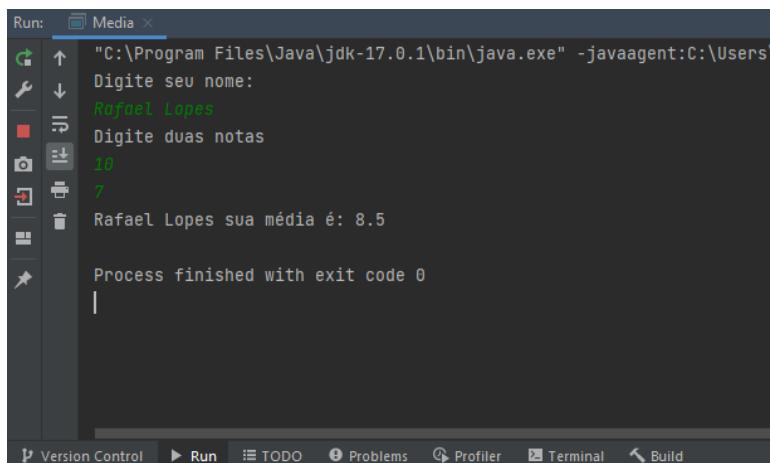
```
1. import java.util.Scanner; // Import necessário para utilizar a classe
2.
3. public class Media {
4.     public static void main (String [] args) {
5.         // Intância do objeto (buffer) utilizando a classe Scanner
6.         Scanner entrada = new Scanner(System.in);
7.         // Declaração de variáveis
8.         String aluno;
9.         double nota1 , nota2 , media;
10.        System.out.println ("Digite seu nome: ");
11.        aluno = entrada.nextLine();    // Recebe o nome do aluno
12.
13.        System.out.println ("Digite duas notas");
14.        nota1 = entrada.nextDouble(); // Recebe a primeira nota
15.        nota2 = entrada.nextDouble(); // Recebe a segunda nota
16.
17.        media = ( nota1 + nota2 ) / 2; // Calcula a média das notas
18.
19.        System.out.println(aluno + " sua média é: " + media); // Exibe a média
20.
21.        entrada.close(); // Encerra a instância
22.    } // fim do método main
22. } // fim da classe Media
```

Código do programa para ler dados digitados no teclado.

Ao executar o programa, ele irá solicitar o **nome** do aluno e as **duas notas**.



E o resultado do **cálculo da média** é exibido em seguida.



```
Run: Media ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:/Users/
Digite seu nome:
Rafael Lopes
Digite duas notas
10
7
Rafael Lopes sua média é: 8.5
Process finished with exit code 0
```

Version Control Run TODO Problems Profiler Terminal Build

A instrução na **linha 6** da cria uma **instancia** da classe **Scanner**, que irá receber os dados digitados no teclado.

Na **linha 11**, a instrução **lê o nome e armazena na variável aluno**. Como o dado esperado é uma **string**, então utilizamos **entrada.nextLine()** para ler o valor como uma **string**.

Na **linha 14**, a instrução **lê a primeira nota e armazena na variável nota1**. Como o dado esperado é um **double** (**valor numérico com casas decimais**), então utilizamos **entrada.nextDouble()** para ler o valor como um **double**.

Na **linha 15**, a instrução **recebe a segunda nota e armazena na variável nota2**. Como o dado esperado é um **double** (**valor numérico com casas decimais**), então também utilizamos **entrada.nextDouble()** para ler o valor como um **double**.

Na **linha 17**, a instrução **media = (nota1 + nota2) / 2** calcula a **média das notas recebidas** e o resultado é **impresso** através do comando na **linha 19**.

Por fim, na **linha 20**, o método **close()** encerrar a **instância da classe**.

Estruturas condicionais

Geralmente, as **instruções** de um programa são executadas **sequencialmente** uma após a outra, na ordem em que foram escritas. Porém muitas vezes é necessário **alterar a sequência da execução de um programa** de acordo com o resultado de um **teste lógico** ou até mesmo repetir um **determinado trecho de código por inúmeras vezes**. Nesse contexto, as **estruturas condicionais** permitem **decidir o fluxo de execução de programa**, selecionando as instruções que serão executadas de acordo com um **teste lógico (Desvio condicional)**.

O **Desvio Condisional** é usado quando **existe a necessidade de verificar condições** para **executar uma instrução** ou um **bloco de instruções**. Toda condição testada **pode retornar dois valores lógicos: true ou false**. Vejamos alguns exemplos de condições:

(x > y)

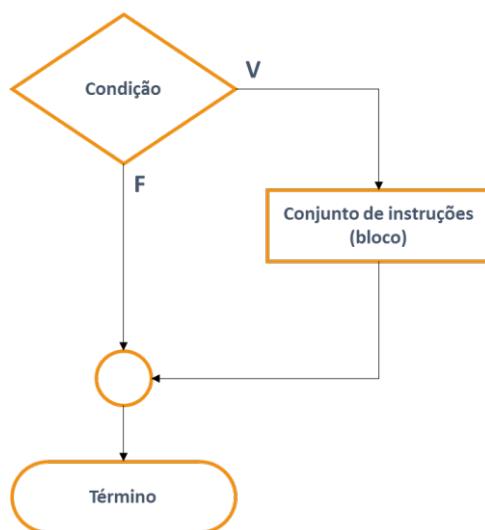
(peso < 50.0)

```
( (x > 0) && (x < 8) )
( (x == 5 && y == 2) || y == 3)
```

O desvio condicional pode ser: **simples (if)**, **composto (if/else)** ou **encadeados (aninhados)**.

Desvio condicional simples (if)

O **desvio condicional simples** é usado para verificar se uma dada **condição é atendida**. Caso a condição seja **atendida** um determinado **conjunto de instruções deverá ser executado**. **Caso contrário**, a condição não for atendida, o fluxo de execução do algoritmo **seguirá após o fim do bloco de decisão**.



A sintaxe do comando if é:

```
if (condicao){
    instrucao1;
    instrucao2;
    instrucao3;
}
proximalInstrucao
```

Início do bloco if

Fim do bloco if

Fique **atento** as **chaves de início e fim do bloco de instruções do desvio condicional if**.

Importante!

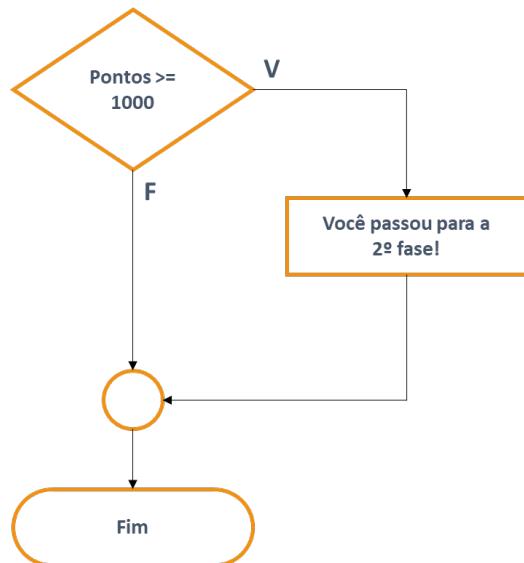


Se o bloco de instruções do **if** tiver apenas **uma instrução**, o **uso de abertura e fechamento de chaves é opcional**.



Vamos praticar!

Vamos criar um programa para **verificar o número de pontos de um jogador**. Caso o número de pontos for **maior ou igual a 1000**, o programa deverá **imprimir uma mensagem** que o jogador **passou para a segunda fase**.



Siga os passos para escrever o programa utilizando o **desvio condicional simples**:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_06_Devio_Cond_if**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Jogo** e insira o código abaixo.

```

1. import java.util.Scanner; // Import necessário para utilizar a classe
2.
3. public class Jogo {
4.     public static void main (String [] args) {
5.         // Intância do objeto (buffer) utilizando a classe Scanner
6.         Scanner entrada = new Scanner(System.in);
7.         int pontos = 0;
8.         System.out.println ("Digite o número de pontos do jogador: ");
9.         pontos = entrada.nextInt(); // Recebe o número de pontos
10.
11.        if (pontos >= 1000) {
12.            System.out.println ("Você passou para a fase 2!!!!");
13.        }
14.
15.        System.out.println("proxima instrucao");
  
```

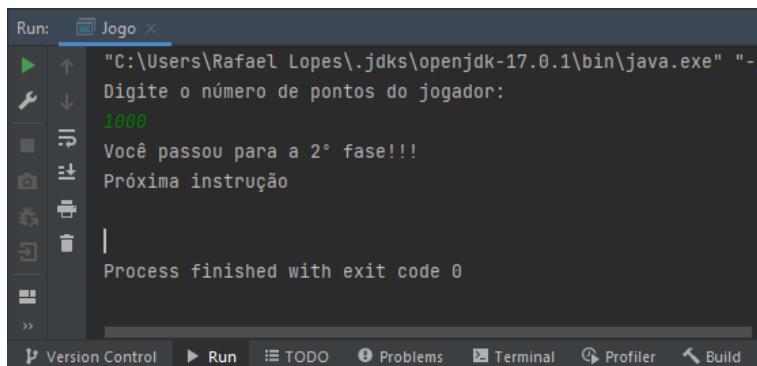
```

16.         entrada.close();
17.     } // fim do método main
18. } // fim da classe Jogo

```

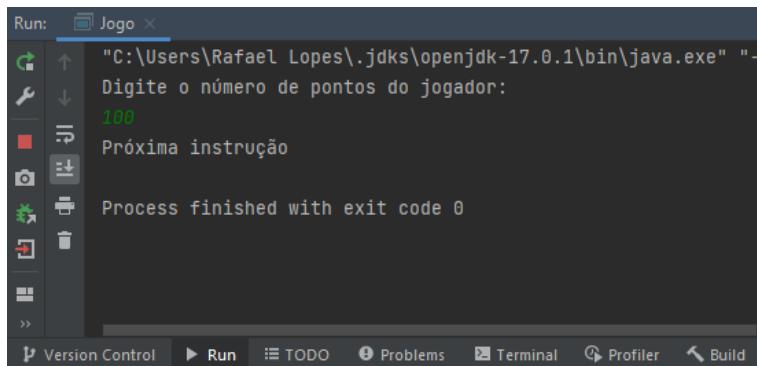
Código utilizando o desvio condicional simples.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.



Resultado da execução do código.

Na **linha 11**, o comando **if** testa se o **valor digitado pelo usuário** é **maior ou igual a 1000**. Caso a condição seja verdadeira, a mensagem da segunda fase é mostrada. Caso contrário, a condição seja **falsa**, apenas a mensagem de próxima instrução é **impressa**.



Nesse exemplo, o desvio condicional **if** tem apenas **uma instrução**, portanto o uso de chaves é **opcional**:

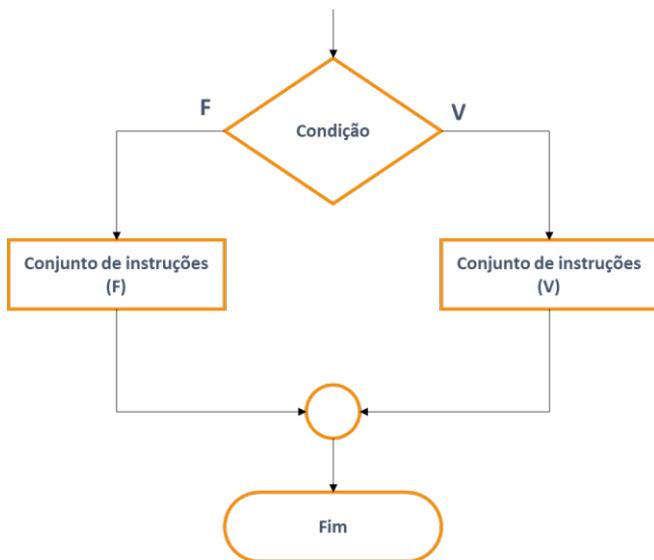
```

if (pontos >= 1000) {
    System.out.println("Você passou para a 2º fase!!!");
}

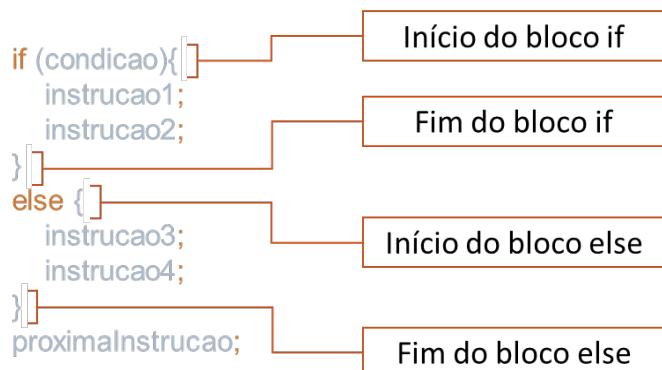
```

Desvio condicional composto (if/else)

O **desvio condicional composto** prevê **dois conjuntos de instruções** para serem executados de acordo com a **avaliação da condição**. Nesse caso, esse um conjunto de instruções para serem executados se a condição for **verdadeira** e um outro conjunto de instruções para serem executados se a condição for **falsa**.



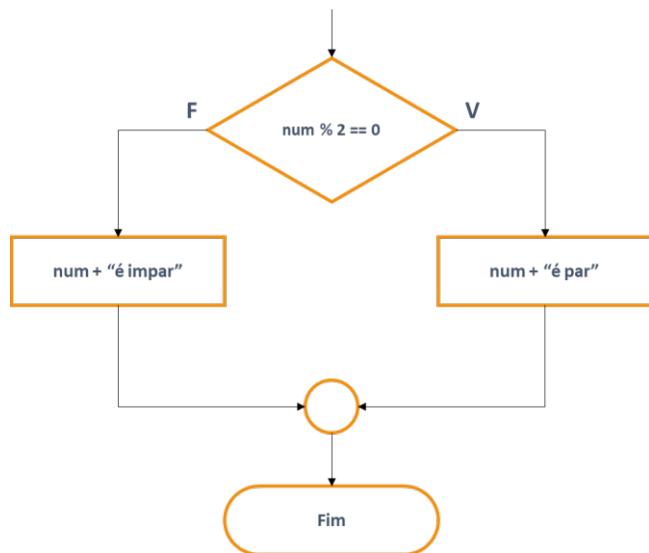
O comando **if/else** possui a seguinte sintaxe:



Nesse caso, temos uma **abertura e fechamento de chaves** delimitando o bloco de instruções **do if**, que serão **executadas** caso a **condição testada seja verdadeira**, e outra **abertura e fechamento de chaves** delimitando o bloco de instruções **do else**, que serão executadas caso a condição testada seja falsa.

Vamos praticar

Vamos criar um programa para **verificar se um número digitado** pelo teclado é **par** ou **ímpar**. **Caso o número seja par**, o programa deverá **imprimir** a mensagem **que ele é par** e, caso o **número seja ímpar**, o programa deverá **imprimir** a mensagem **que ele é ímpar**.



Vamos praticar!

Siga os passos para escrever o programa utilizando operadores relacionais:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_07_Devio_Cond_if_else**

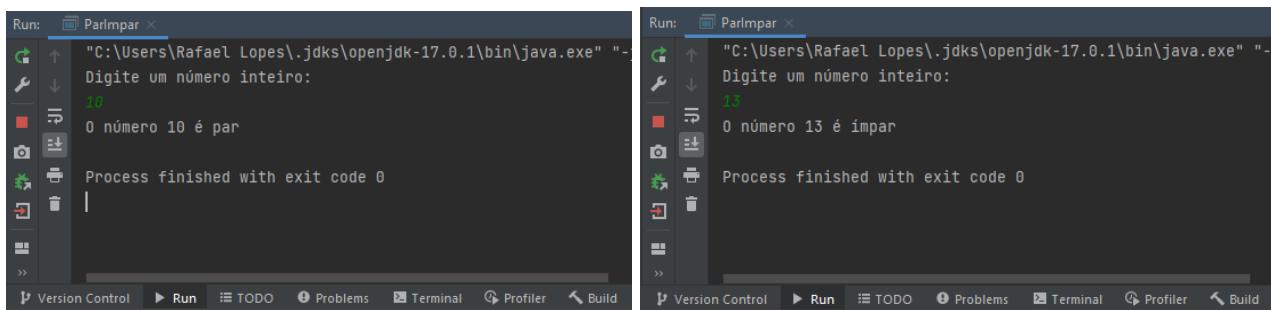
No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **ParImpar** e insira o código abaixo.

```

1. import java.util.Scanner; // Import necessário para utilizar a classe
2.
3. public class ParImpar {
4.     public static void main (String [] args) {
5.         // Intância do objeto (buffer) utilizando a classe Scanner
6.         Scanner entrada = new Scanner(System.in);
7.         int numero;
8.         System.out.println ("Digite um número inteiro: ");
9.         numero = entrada.nextInt(); // Recebe o número
10.
11.        // Verifica se o número é par
12.        if ((numero % 2) == 0) {
13.            System.out.println ("O número " + numero + " é par");
14.        }
15.        else {
16.            System.out.println ("O número " + numero + " é ímpar");
17.        }
18.        entrada.close();
19.    } // fim do método main
20. } // fim da classe ParImpar
  
```

Código utilizando o desvio condicional composto.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.



```
Run: ParImpar
"C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" -
Digite um número inteiro:
10
O número 10 é par
Process finished with exit code 0
```



```
Run: ParImpar
"C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" -
Digite um número inteiro:
13
O número 13 é ímpar
Process finished with exit code 0
```

Resultado da execução do código.

Na **linha 12**, o comando **if** testa se o **resto da divisão** do valor digitado **por 2 é igual a 0**. Caso a condição seja **verdadeira**, o **número é par**, então o bloco de instruções do comando **if** deverá ser **executado** e a **mensagem de número par** **deverá ser impressa**. Caso contrário, a condição seja **falsa**, o **número é ímpar**, então o bloco de instruções do comando **else** deverá ser executado e a **mensagem de número ímpar** **deverá ser impressa**.

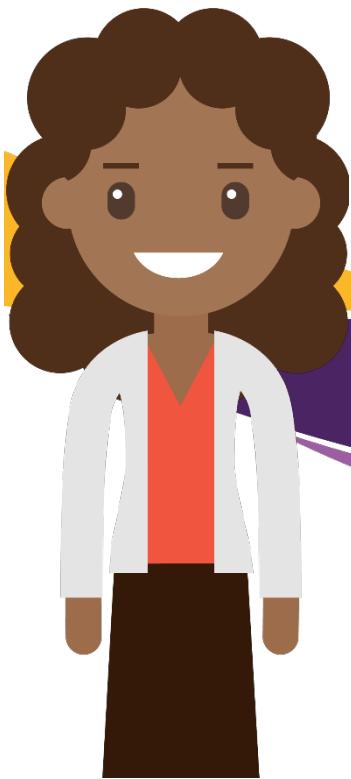
Nesse exemplo, o desvio condicional **if/else** tem apenas uma instrução, portanto o uso de chaves é opcional:

```
if ((numero % 2) == 0)
    System.out.println ("O número " + numero + " é par");
else
    System.out.println ("O número " + numero + " é ímpar");
```

Você também poderia usar o **operador lógico de negação na condição testada** e desse modo não precisaria colocar o operador relacional de comparação **==**.

```
if (!(numero % 2))
    System.out.println ("O número " + numero + " é par");
else
    System.out.println ("O número " + numero + " é ímpar");
```

O resultado da operação **numero % 2** só poderá **ser zero (número par)** ou **um (número ímpar)**. Como queremos que seja **verdadeira** a condição quando o **número for par**, então colocamos o **operador lógico de negação !** para **inverter o valor booleano**. Portanto, se o resultado for **zero**, o operador **inverte para um** e assim é **verdadeira a condição se o número é par**. Se o resultado for **um**, o operador **inverte para zero** e assim é **falsa a condição se o número é ímpar**.



Desvios condicionais encadeados

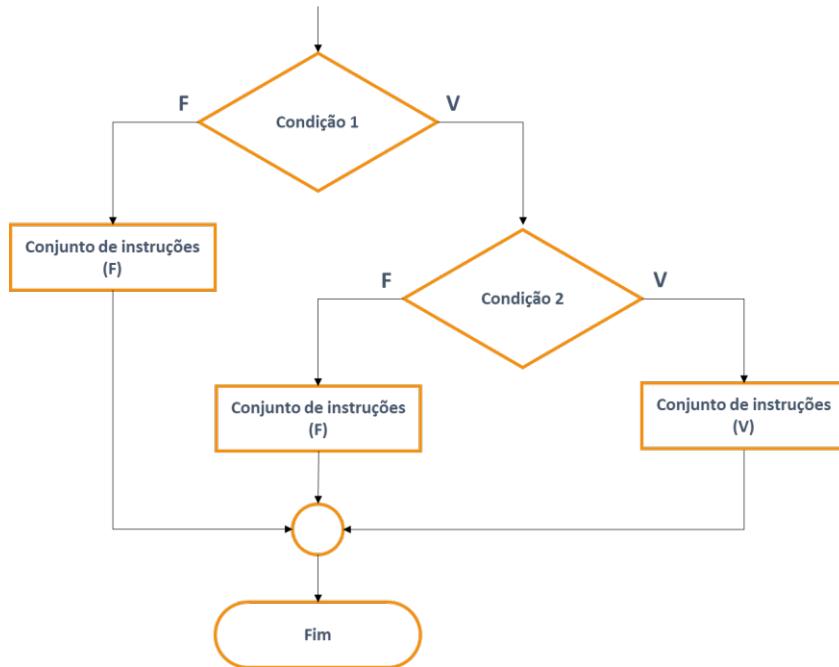
Os objetivos desta aula são:

- Compreender os desvios condicionais encadeados;
- Conhecer o uso e sintaxe do comando switch-case.

Bons estudos!

Desvio condicional encadeado

Muitas vezes, em **programação**, precisamos **testar diversas condições** e **cada condição depende do resultado da condição, que foi testada anteriormente**. Esse tipo de desvio condicional também pode ser chamado de **ifs aninhados**, que basicamente é colocar um **if dentro de if**.



A sintaxe de um desvio condicional encadeado é:

```

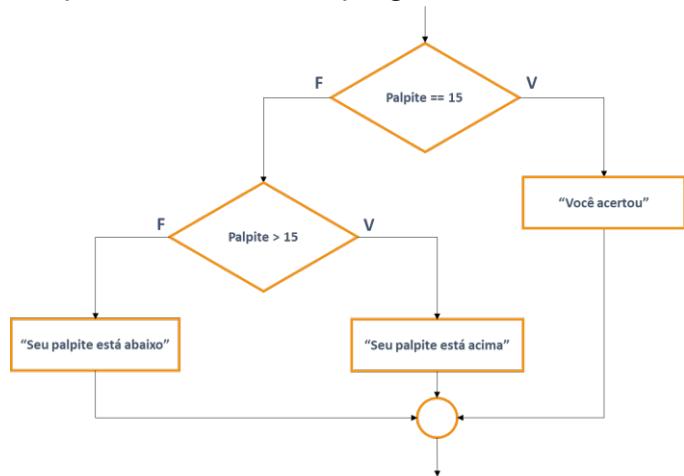
if (condicao1) {
    instrucao1;
    instrucao2;
} else {
    if (condicao2) {
        instrucao3;
        instrucao4;
    } else {
        instrucao5
    }
}
proximalInstrucao
    
```

Fique atento as **aberturas** e **fechamento** de **chaves** para cada bloco do desvio condicional encadeado.



Vamos praticar!

Vamos fazer um programa do **Jogo de adivinhação**. O objetivo do jogo é **adivinar** o número **15**, predeterminado no programa. O usuário irá digitar



Siga os passos para escrever o programa utilizando operadores relacionais:

Crie um novo projeto no **Interllij IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_08_Adivinha**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **JogoAdivinha** e insira o código abaixo:

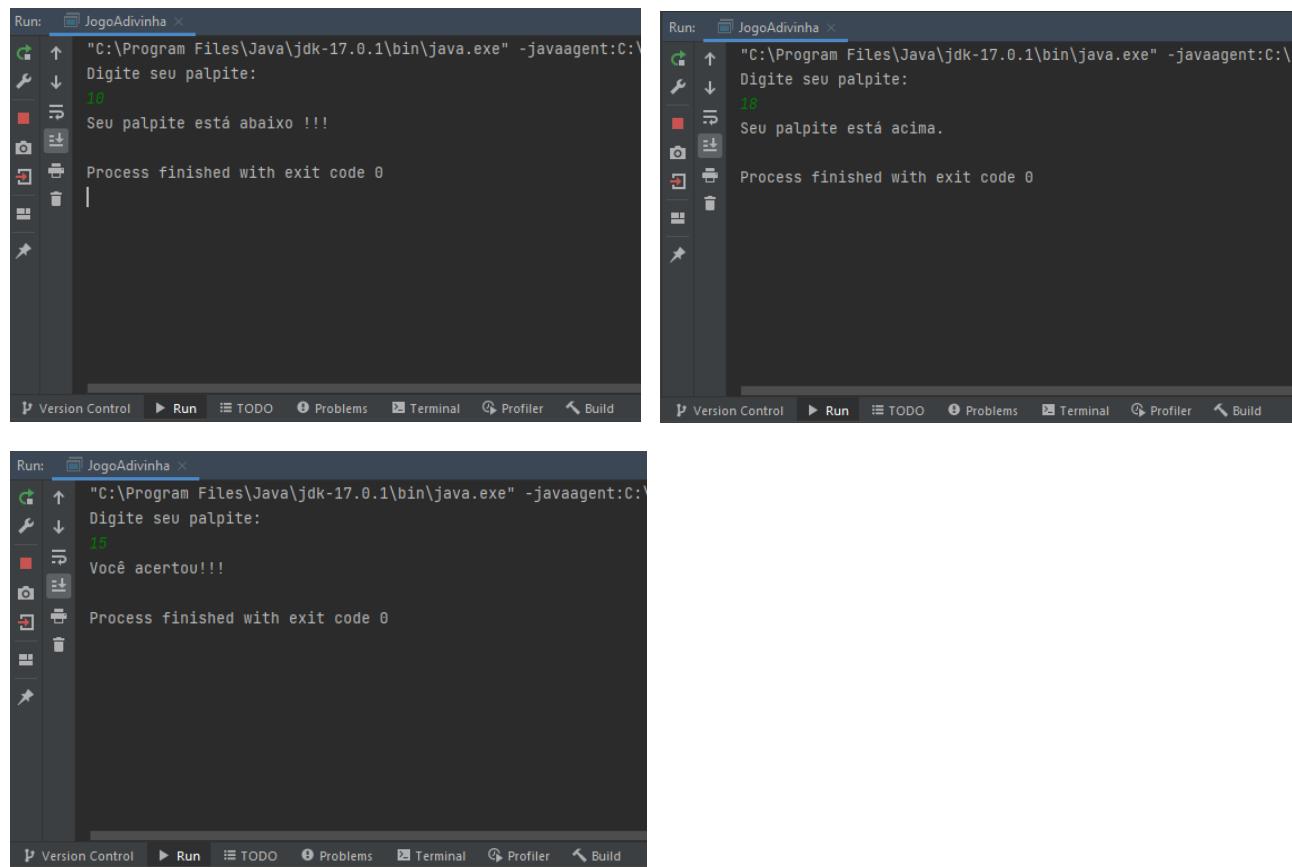
```

1. import java.util.Scanner; // Import necessário para utilizar a classe
2.
3. public class JogoAdivinha {
4.     public static void main (String [] args) {
5.         // Intância do objeto (buffer) utilizando a classe Scanner
6.         Scanner entrada = new Scanner(System.in);
7.         int palpate;
8.         System.out.println("Digite seu palpate:");
9.         palpate = entrada.nextInt();
10.        if(palpate == 15) {
11.            System.out.println("Você acertou!!!\"");
12.        }
13.        else {
14.            if(palpate < 15) {
15.                System.out.println("Seu palpito está abaixo !!!\"");
16.            } else {
17.                System.out.println("Seu palpito está acima.");
18.            }
19.        }
20.        entrada.close();
21.    } // fim do método main
22. } // fim da classe JogoAdivinha
  
```

Código do programa para o jogo de adivinhação.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, ele irá solicitar que o usuário **digite um palpite** e então o programa testará as condições dos **if encadeados** e **imprimirá uma das mensagens** de acordo com a condição **verdadeira**.



```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\junit-5.8.1.jar
Digite seu palpite:
16
Seu palpite está abaixo !!!
Process finished with exit code 0
```



```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\junit-5.8.1.jar
Digite seu palpite:
18
Seu palpite está acima.
Process finished with exit code 0
```



```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\junit-5.8.1.jar
Digite seu palpite:
15
Você acertou!!!
Process finished with exit code 0
```

A instrução na **linha 10** verifica se o palpite digitado é **igual a 15**. Caso **verdadeiro** a condição o programa exibirá a mensagem que **você acertou**. Caso a condição seja **falsa**, o programa irá verificar a condição do **if** na **linha 14**, caso seja **verdadeira**, ele imprimirá que o palpite está **abaixo do valor 15**. Caso a condição seja **falsa**, o programa executa o bloco de comandos do **else** na **linha 17** e imprimirá a mensagem que o **palpite está acima do valor 15**.



Vamos praticar!

Vamos deixar esse programa um pouco mais difícil utilizando a **classe Random** para gerar um **número aleatório de 1 a 10** para que possamos tentar adivinhar.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_09_Adivinha_Random**

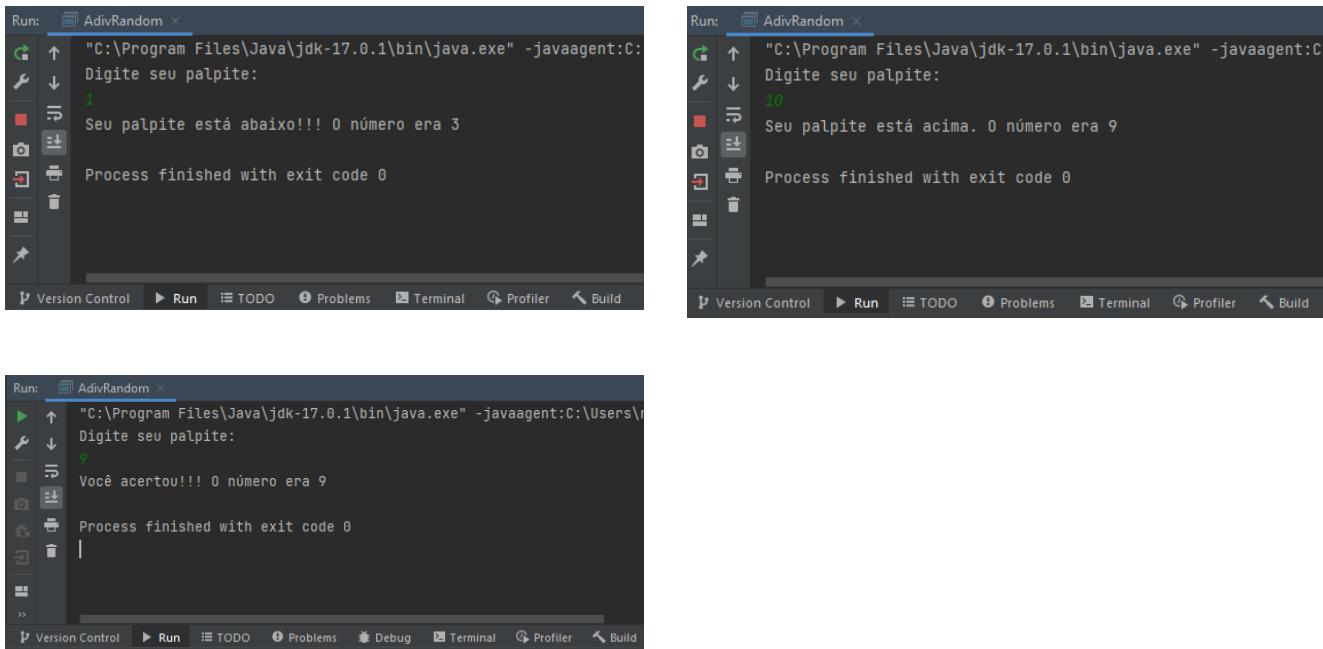
No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **AdivRandom** e insira o código abaixo.

```
1. import java.util.Scanner; // Import necessário para utilizar a classe Scanner
2. import java.util.Random; // Import necessário para utilizar a classe Random
3.
4. public class AdivRandom {
5.     public static void main (String [] args) {
6.         // Intância (buffer) utilizando a classe Scanner
7.         Scanner entrada = new Scanner(System.in);
8.         // Intância utilizando a classe Random
9.         Random rand = new Random();
10.        int palpate, num = 0;
11.        num = rand.nextInt(10) + 1;
12.        System.out.println("Digite seu palpate:");
13.        palpate = entrada.nextInt();
14.        if(palpate == num) {
15.            System.out.println("Você acertou!!! O número era " + num);
16.        }
17.        else {
18.            if(palpate < num) {
19.                System.out.println("Seu palpate está abaixo!!! O número era " +
num);
20.            } else {
21.                System.out.println("Seu palpate está acima. O número era " + num
);
22.            }
23.        }
24.        entrada.close();
25.    } // fim do método main
26. } // fim da classe AdivRandom
```

Código do programa para o jogo de adivinhação com a classe Random.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, ele irá solicitar que o usuário digite um palpate e então o programa testará as condições dos if encadeados e imprimirá uma das mensagens de acordo com a condição verdadeira.



```

Run: AdivRandom ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users\...\.gradle\caching\org\javatutorial\javatutorial\1.0-SNAPSHOT\javatutorial-1.0-SNAPSHOT.jar
Digite seu palpite:
1
Seu palpite está abaixo!!! O número era 3
Process finished with exit code 0

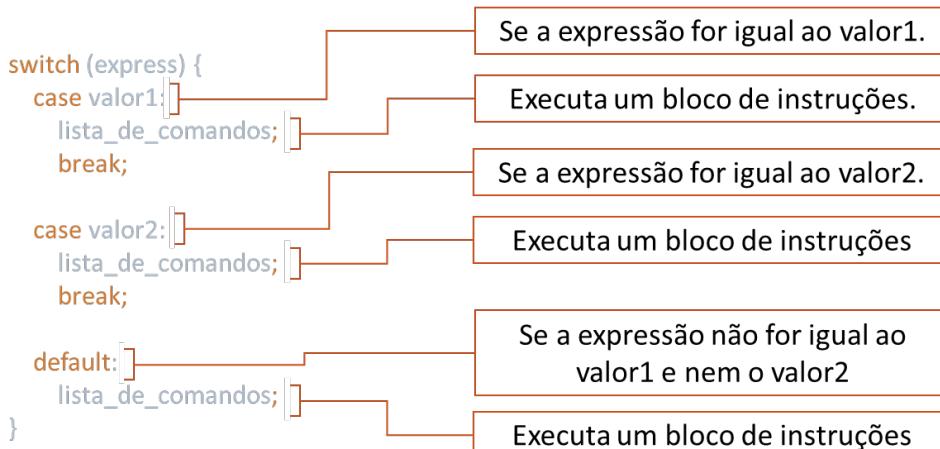
Run: AdivRandom ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users\...\.gradle\caching\org\javatutorial\javatutorial\1.0-SNAPSHOT\javatutorial-1.0-SNAPSHOT.jar
Digite seu palpite:
10
Seu palpite está acima. O número era 9
Process finished with exit code 0

Run: AdivRandom ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users\...\.gradle\caching\org\javatutorial\javatutorial\1.0-SNAPSHOT\javatutorial-1.0-SNAPSHOT.jar
Digite seu palpite:
9
Você acertou!!! O número era 9
Process finished with exit code 0
  
```

A instrução da **linha 9** criar uma instância da classe **Random**, que nos permite “**sor-tear**” um **valor aleatório**. Na **linha 11**, o método **nextInt(10)** sorteia um valor **inteiro** de **0** a **9**, por isso precisamos **somar 1** para que a **faixa dos números possíveis a serem sorteados fiquem entre 1 e 10**.

Switch-case

O comando **switch-case** é utilizada para trabalhar com situações mutualmente exclusivas. Situações mutualmente exclusivas são aquelas que **se uma situação for executada, as demais não poderão ser executadas**. O **switch-case** sempre verifica o valor de uma **expressão** para decidir **o que fazer**, ou seja, **qual(is) instrução(ões) será(ão) verificada(s)**. A **expressão** será comparada com o valor de cada comando **case**. Sempre que a **expressão casar** com o valor de um **case** a lista de comando desse **case** é **executada** e o **switch finaliza** ao encontrar o comando **break**. A **expressão** comparada pode ser do tipo **int, char ou String**. A sintaxe do **switch-case** é:



O caso **default** é executado quando **nenhum case for satisfeito**. O comando **switch** não pode ter **dois ou mais** comandos **case** com **valores iguais**.

Se compararmos o **switch-case** com os ifs encadeados veremos que eles têm **muitas semelhanças**.

```
switch (express) {
    case valor1:
        lista_de_comandos;
        break;

    case valor2:
        lista_de_comandos;
        break;

    default:
        lista_de_comandos;
}
```

```
if (variavel == valor1) {
    lista_de_comandos;
}
else if (variavel == valor2){
    lista_de_comandos;
}
else{
    lista_de_comandos;
}
```



Vamos praticar!

Vamos criar um programa para **exibir um menu de opções** para o usuário **escolher**. O programa inicialmente deverá mostrar o seguinte texto na tela:

```
===== Menu de opções =====
1 - Cadastrar Produtos
2 - Listas Produtos
3 - Sair do Sistema
===== Escolha uma opção =====
```

O Usuário deverá digitar **uma** das **três opções**:

Caso digite 1 deverá aparecer:

- “Você escolheu o menu 1”
- “Que é a opção Cadastrar Produtos”

Caso digite 2 deverá aparecer:

- “Você escolheu o menu 2”
- “Que é a opção Listar Produtos”

Caso digite 3 deverá aparecer:

- “Item de menu inválido”

Se digitar qualquer outra coisa:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_10_Switch_case**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Menu** e insira o código abaixo:

```

1. import java.util.Scanner; // Import necessário para utilizar a classe Scanner
2.
3. public class Menu {
4.     public static void main ( String [] args ) {
5.
6.         System.out.println("===== Menu de Opcoes =====");
7.         System.out.println(" 1 - Cadastrar Produtos ");
8.         System.out.println(" 2 - Listas Produtos ");
9.         System.out.println(" 3 - Sair do Sistema ");
10.        System.out.println("===== Escolha uma opcao =====");
11.
12.        Scanner entrada = new Scanner(System.in);
13.        int menu = entrada.nextInt(); // Lê a opção digitada
14.
15.        switch (menu) {
16.            case 1:
17.                System.out.println("Voce escolheu o menu 1");
18.                System.out.println("Que eh a opcao Cadastrar Produtos");
19.                break;
20.            case 2:
21.                System.out.println("Voce escolheu o menu 2");
22.                System.out.println("Que eh a opcao Listar Produtos");
23.                break;
24.            case 3:
25.                System.out.println("Voce escolheu o menu 3");
26.                System.out.println("Que eh a opcao sair do Sistema");
27.                break;
28.            default:
29.                System.out.println("Item de menu invalido");
30.        }
31.        entrada.close();
32.    } // fim do método main
33. } // fim da classe Menu

```

Código utilizando o switch-case.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

```
Run: Menu ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\lib\jdwp-agent.jar=transport=dt_socket,address=8000,server=y,suspend=n
===== Menu de Opcoes =====
1 - Cadastrar Produtos
2 - Listas Produtos
3 - Sair do Sistema
===== Escolha uma opcao =====
1
Voce escolheu o menu 1
Que eh a opcao Cadastrar Produtos

Process finished with exit code 0
```

```
Run: Menu ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\lib\jdwp-agent.jar=transport=dt_socket,address=8000,server=y,suspend=n
===== Menu de Opcoes =====
1 - Cadastrar Produtos
2 - Listas Produtos
3 - Sair do Sistema
===== Escolha uma opcao =====
2
Voce escolheu o menu 2
Que eh a opcao Listar Produtos

Process finished with exit code 0
```

```
Run: Menu ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\lib\jdwp-agent.jar=transport=dt_socket,address=8000,server=y,suspend=n
===== Menu de Opcoes =====
1 - Cadastrar Produtos
2 - Listas Produtos
3 - Sair do Sistema
===== Escolha uma opcao =====
3
Voce escolheu o menu 3
Que eh a opcao sair do Sistema

Process finished with exit code 0
```

```
Run: Menu ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\lib\jdwp-agent.jar=transport=dt_socket,address=8000,server=y,suspend=n
===== Menu de Opcoes =====
1 - Cadastrar Produtos
2 - Listas Produtos
3 - Sair do Sistema
===== Escolha uma opcao =====
5
Item de menu invalido

Process finished with exit code 0
```

Na **linha 15**, o **switch** recebe a variável **menu**, que contém o **valor da opção escolhida pelo usuário**. Caso o usuário digitar **1**, o switch testa o case 1 e verifica que ele **casa com a opção escolhida pelo usuário**, então o bloco de instruções do **case 1 será executado**:

```
case 1:
    System.out.println("Voce escolheu o menu 1");
    System.out.println("Que eh a opcao Cadastrar Produtos");
    break;
```

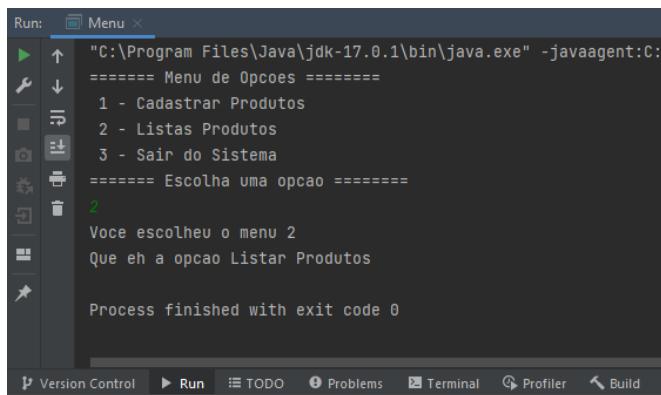
```
Run: Menu ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\lib\jdwp-agent.jar=transport=dt_socket,address=8000,server=y,suspend=n
===== Menu de Opcoes =====
1 - Cadastrar Produtos
2 - Listas Produtos
3 - Sair do Sistema
===== Escolha uma opcao =====
1
Voce escolheu o menu 1
Que eh a opcao Cadastrar Produtos

Process finished with exit code 0
```

E quando o programa encontra a instrução **break** na **linha 19**, ele encerra o **switch e não testa as outras opções**.

Caso o usuário digitar **2**, o switch testa o **case 1**, como ele **não casa com a opção escolhida**, o **switch testa o case 2** e verifica que ele **casa com a opção escolhida pelo usuário**, então o bloco de instruções do **case 2 será executado**:

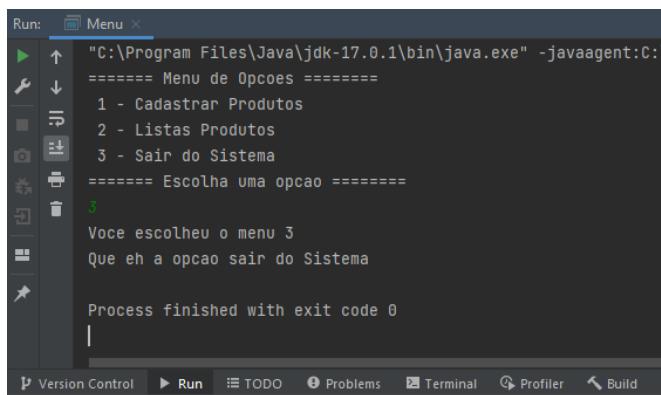
```
case 2:
    System.out.println("Voce escolheu o menu 2");
    System.out.println("Que eh a opcao Listar Produtos");
    break;
```



E quando o programa encontra a instrução **break** na **linha 23**, ele encerra o **switch e não testa as outras opções**.

Caso o usuário digitar **3**, o switch testa o **case 1**, como ele **não casa com a opção escolhida**, o **switch testa o case 2**, como ele **não casa com a opção escolhida**, o **switch testa o case 3** e verifica que ele **casa com a opção escolhida pelo usuário**, então o bloco de instruções do **case 3 será executado**:

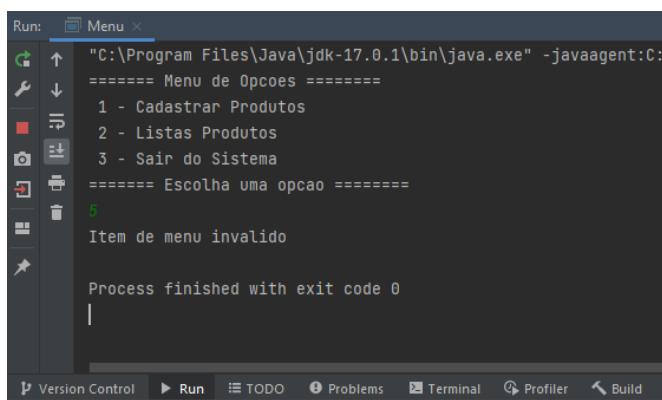
```
case 3:
    System.out.println("Voce escolheu o menu 3");
    System.out.println("Que eh a opcao sair do Sistema");
    break;
```



E quando o programa encontra a instrução **break** na **linha 27**, ele encerra o **switch e não testa as outras opções**.

Se o usuário digitar um valor diferente de **1, 2 ou 3**, o **switch irá executar a lista de comando do default**.

```
default:  
    System.out.println("Item de menu invalido");
```



A screenshot of the IntelliJ IDEA interface showing a terminal window titled "Run: Menu". The terminal displays the following output:

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:  
===== Menu de Opcoes =====  
1 - Cadastrar Produtos  
2 - Listas Produtos  
3 - Sair do Sistema  
===== Escolha uma opcao =====  
5  
Item de menu invalido  
  
Process finished with exit code 0
```

The terminal window is part of the IntelliJ IDEA interface, which includes tabs for Version Control, Run, TODO, Problems, Terminal, Profiler, and Build.



Vamos praticar!

Vamos fazer outro exemplo para o switch **verificar um char digitado pelo usuário**. Siga os passos para escrever o programa utilizando o **switch-case**:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_11_swich_case_II**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Calculadora** e insira o código abaixo:

```
1. import java.util.Scanner; // Import necessário para utilizar a classe Scanner  
2.  
3. public class Calculadora {  
4.     public static void main ( String [] args ) {  
5.  
6.         System.out.println("===== Calculadora =====");  
7.  
8.         Scanner entrada = new Scanner(System.in);  
9.         System.out.println("Digite um número:");  
10.        double num1 = entrada.nextDouble(); // Lê a opção digitada
```

```

11.     System.out.println("Digite outro número:");
12.     double num2 = entrada.nextDouble(); // Lê a opção digitada
13.     System.out.println("===== Operação Matemática =====");
14.     System.out.println(" + Adicionar os números ");
15.     System.out.println(" - Subtrair os números ");
16.     System.out.println(" * multiplicar os números ");
17.     System.out.println("===== Escolha uma opção =====");
18.     String oper = entrada.next(); // Lê a opção digitada
19.
20.     switch (oper) {
21.         case "+":
22.             double soma = num1 + num2;
23.             System.out.println("Você escolheu adição");
24.             System.out.println("O número " + num1 + " + " + num2 + " = " + s
oma);
25.             break;
26.         case "-":
27.             double sub = num1 - num2;
28.             System.out.println("Você escolheu subtração");
29.             System.out.println("O número " + num1 + " - " + num2 + " = " + s
ub);
30.             break;
31.         case "*":
32.             double mult = num1 * num2;
33.             System.out.println("Você escolheu multiplicação");
34.             System.out.println("O número " + num1 + " * " + num2 + " = " + m
ult);
35.             break;
36.         default:
37.             System.out.println("Operação não disponível ou inválida!");
38.     }
39.     entrada.close();
40. } // fim do método main
41. } // fim da classe Calculadora

```

Código utilizando o switch-case.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

```
Run: Calculadora >
    "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C
    ===== Calculadora =====
    Digite um número:
    5
    Digite outro número:
    2
    ===== Operação Matemática =====
    + Adicionar os números
    - Subtrair os números
    * multiplicar os números
    ===== Escolha uma opção =====
    +
    Voce escolheu adição
    0 número 5.0 + 2.0 = 7.0

    Process finished with exit code 0
```

```
Run: [ ] Calculadora x
C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...
===== Calculadora =====
Digite um número:
25
Digite outro número:
0
===== Operação Matemática =====
+ Adicionar os números
- Subtrair os números
* multiplicar os números
===== Escolha uma opção =====
-
Voce escolheu subtração
0 número 25.0 - 0.0 = 19.0

Process finished with exit code 0
|
```

```
Run: [ ] Calculadora x
      ↑
      "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:
      ===== Calculadora =====
      Digite um número:
      4
      Digite outro número:
      7
      ===== Operação Matemática =====
      + Adicionar os números
      - Subtrair os números
      * multiplicar os números
      ===== Escolha uma opção =====
      *
      Voce escolheu subtração
      0 número 4.0 * 7.0 = 28.0

      Process finished with exit code 0
      |
```



Estruturas de Repetição while e do-while

Os objetivos desta aula são:

- Compreender as estruturas de repetição;
- Conhecer o comando while e do-while e sua sintaxe.

Bons estudos!

Estruturas de repetição

Estruturas de repetição são necessárias para quando queremos **repetir a execução de uma ou mais instruções múltiplas vezes**. Desde modo, evitamos o trabalho manual de **reescrever o trecho de código repetidas vezes**. Por exemplo, para criar um programa para ler **10 notas** de alunos sem estrutura de repetição seria assim:

```
1. import java.util.Scanner;
2.
3. public class Main {
4.     public static void main(String[] args) {
5.         Scanner entrada = new Scanner(System.in);
6.
7.         int n1, n2, n3, n4, n5, n6, n7, n8, n9, n10;
8.
9.         System.out.println(" Informe a nota do aluno 1");
10.        n1 = entrada.nextInt();
11.
12.        System.out.println(" Informe a nota do aluno 2");
13.        n2 = entrada.nextInt();
14.
15.        System.out.println(" Informe a nota do aluno 3");
16.        n3 = entrada.nextInt();
17.
18.        System.out.println(" Informe a nota do aluno 4");
19.        n4 = entrada.nextInt();
20.
21.        System.out.println(" Informe a nota do aluno 5");
22.        n5 = entrada.nextInt();
23.
24.        System.out.println(" Informe a nota do aluno 6");
25.        n6 = entrada.nextInt();
26.
27.        System.out.println(" Informe a nota do aluno 7");
28.        n7 = entrada.nextInt();
29.
30.        System.out.println(" Informe a nota do aluno 8");
31.        n8 = entrada.nextInt();
32.
33.        System.out.println(" Informe a nota do aluno 9");
34.        n9 = entrada.nextInt();
35.
36.        System.out.println(" Informe a nota do aluno 10");
37.        n10 = entrada.nextInt();
38.    }
39. }
```

Essa abordagem **não é eficiente**, pois, além de ser **muito trabalhosa, torna-se difícil a manutenibilidade do código**. Por exemplo, se ao invés **10 alunos** tivéssemos **100** ou **1000**?

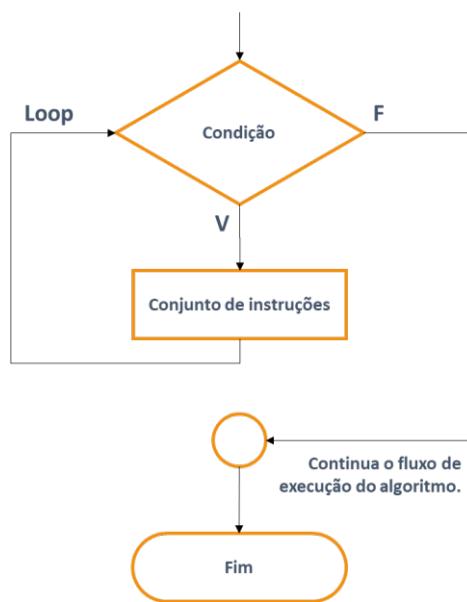
Por isso, existem formas **mais eficientes** de fazer isso, **como o uso do laço de repetição while**, por exemplo. As vantagens de usar **estruturas de repetição** é que o algoritmo passa a ter um tamanho (**número de linhas**) **menor** e se torna **mais legível** e é possível **aumentar a escalabilidade do projeto sem fazer grandes alterações no código**.

Em programação, podemos fazer **dois tipos de repetições**:

- Aquela em que **sabemos o número de vezes que as instruções serão executadas**.
- Aquela em que **não sabemos o número vezes que devemos repetir as instruções**, pois de alguma condição específica, por exemplo, o usuário digitar uma determinada tecla.

Comando while

O **while** (que significa **enquanto**) é uma repetição com **teste no início do comando**. A condição é **testada a cada iteração**, ou seja, **antes de executar novamente as instruções o while verificar se a condição ainda continua verdadeira**. Desse modo, o bloco de instruções será repetidamente executado **enquanto a condição for verdadeira** e, quando a condição for **falsa**, a execução do bloco de comandos será **interrompida**.



A sintaxe do comando **while** é:

```

while (condicao) {
    //Bloco de instruções
    instrucao1;
    instrucao2;
}
proxima_instrucao
  
```

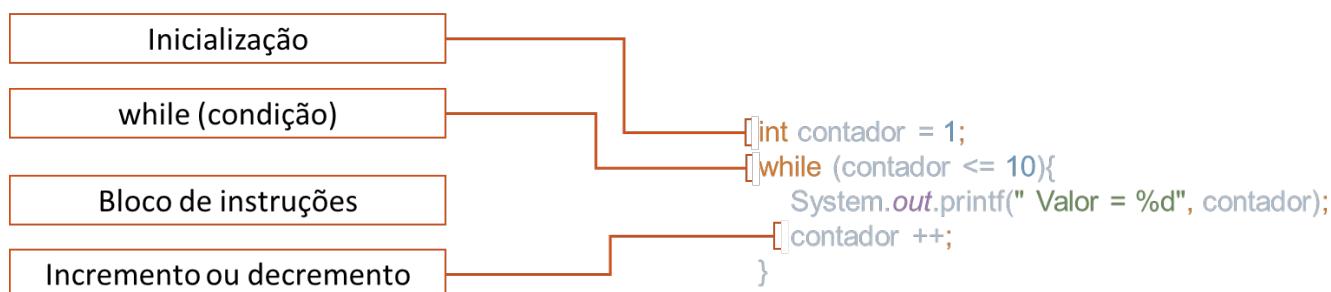
Annotations for the code:

- Enquanto a condição for verdadeira**: Points to the condition part of the while statement.
- Execute o bloco de instruções**: Points to the block of statements inside the while loop.

Contador

Quando queremos **repetir a execução de um programa** por um número **finito de vezes**, precisamos **criar um contador**. O contador nada mais é que uma **variável inteira**, que é usada para **contar a quantidade de vezes que um bloco de instruções é executado repetidamente**.

Antes de utilizar o contador, você deve garantir um **valor inicial** (**fazer a inicialização do contador**) **fora do bloco** de instruções do **while**. Você também **deve garantir** que a **condição** do **while** será **falsa** em algum momento, para isso, **dentro do bloco de instruções do while**, o contador deverá ser **incrementado** (**seu valor irá aumentar a cada execução do loop**) ou **decrementado** (**seu valor irá diminuir a cada execução do loop**) de acordo com a lógica implementada.



Vamos praticar!

Vamos fazer um programa que imprima a **sequência dos números inteiros de 1 a 100**. Siga os passos para criar o projeto:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_12_Imprimir_Sequencia**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Contador** e insira o código abaixo.

```

1. public class Contador {
2.     public static void main(String[] args) {
3.         int contador; // declarando o contador
4.         contador = 1; // inicializando o contador
5.
6.         while (contador <= 100) {
7.             System.out.print(contador + " ");
8.             if(contador%30 == 0) System.out.println(""); // Quebra de linha para ver todos os números
9.             contador++; // mesma coisa que utilizar contador = contador + 1
    
```

```

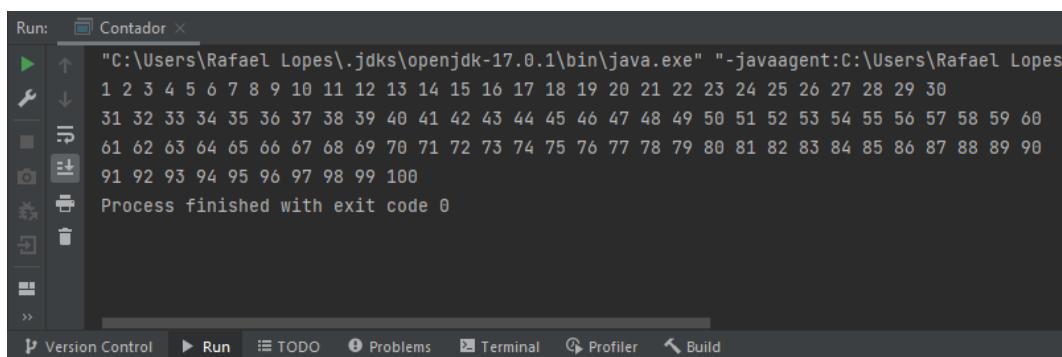
10.        }
11.    } // fim do método main
12. } // fim da classe Contador

```

Código do programa utilizando **while**.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o **programa**, ele irá **imprimir os números** de **1** a **100** no terminal.



```

Run: Contador x
C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" "-javaagent:C:\Users\Rafael Lopes"
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
Process finished with exit code 0

```

A instrução na **linha 4**:

```
contador = 1; // inicializando o contador
```

é onde inicializamos o nosso contador, ou seja, **definimos um valor inicial para ele** (no caso o valor inicial é **1**).

A condição do comando **while**

```
while (contador <= 100)
```

é **falsa** quando o contador assumir **valores acima de 100**.

Dentro do bloco de instruções do **while**, na **linha 9**, temos a instrução de **incremento do contador**, ou seja,

```
contador++; // mesma coisa que utilizar contador = contador + 1
```

Aumenta o valor da **variável contador** a cada vez que o **bloco de instruções** é executado. Com isso, **garantimos** que em **algum momento** a **condição do while** vai se tornar **falsa**.



Vamos praticar!

Vamos implementar agora um laço de repetição com um contador que tenha seu valor **decrementado**.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_13_While_Decremento**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Decremento** e insira o código abaixo:

```

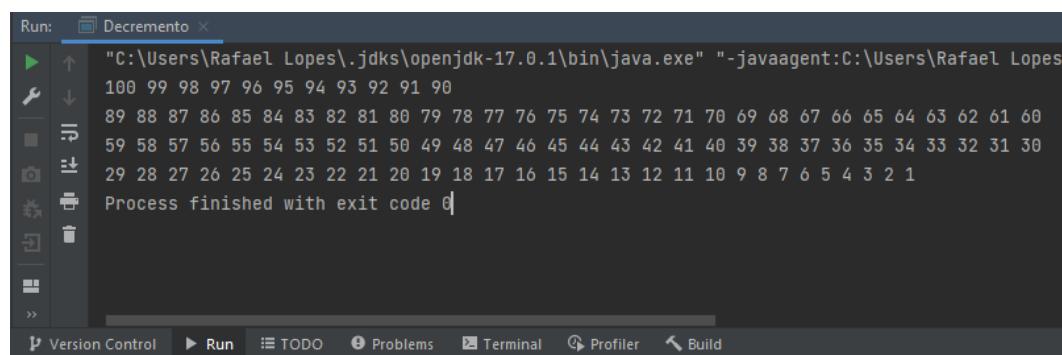
1. public class Decremento {
2.     public static void main(String[] args) {
3.         int contador = 100; // Declara e inicializa o contador
4.
5.         while (contador > 0) {
6.             System.out.print(contador + " ");
7.             if (contador % 30 == 0) System.out.println(""); // Quebra de linha para ve
r todos os números
8.             contador--; // mesma coisa que utilizar contador = contador - 1
9.         }
10.    } // fim do método main
11. } // fim da classe Decremento

```

Código do programa while com o contador sendo decrementado.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o **programa**, ele irá **imprimir os números de 100 a 1** no terminal.



The screenshot shows the IntelliJ IDEA interface with the 'Run' tool window open. The 'Decremento' run configuration is selected. The terminal pane displays the following output:

```

Run: Decremento ×
"C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" "-javaagent:C:\Users\Rafael Lopes\.
100 99 98 97 96 95 94 93 92 91 90
89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60
59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30
29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Process finished with exit code 0

```

The bottom navigation bar includes icons for Version Control, Run, TODO, Problems, Terminal, Profiler, and Build.

A instrução na **linha 4**:

```
int contador = 100; // Declara e inicializa o contador
```

é onde **declaramos e inicializamos** o nosso contador, ou seja, **definimos um valor inicial para ele** (no caso o valor inicial é **100**).

A condição do comando **while**

```
while (contador > 0)
```

é **falsa** quando o contador assumir **valores menores ou iguais a 0 (zero)**.

Dentro do bloco de instruções do **while**, na **linha 8**, temos a instrução de **decremento do contador**, ou seja,

```
Contador--; // mesma coisa que utilizar contador = contador - 1
```

Diminui o valor da **variável contador** a cada vez que o **bloco de instruções é executado**. Com isso, **garantimos** que em **algum momento a condição do while vai se tornar falsa**.

Acumulador

Podemos também usar **laços de repetição** para **acumular valores dentro de uma variável**. Nesse caso, a **variável** tem o **papel de acumuladora**, pois ela vai **armazenando os valores a cada repetir do loop**. Por exemplo, se quisermos calcular o **somatório** dos **números pares** de **1 a 100**.



Vamos praticar!

Vamos fazer um programa que **imprime** a **soma acumulada** dos números pares de **1 a 100**. Siga os passos para criar o projeto:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_14_Somatorio**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Somatorio** e insira o código abaixo:

```
1. public class Somatorio {
2.     public static void main(String[] args) {
3.         int contador = 1;
4.         int acumulador = 0;
5.
6.         while (contador <= 100) {
7.             if (contador % 2 == 0)
8.                 acumulador = acumulador + contador; // adiciona o contador par ao acumulador
9.             contador++; // incrementa o contador
```

```

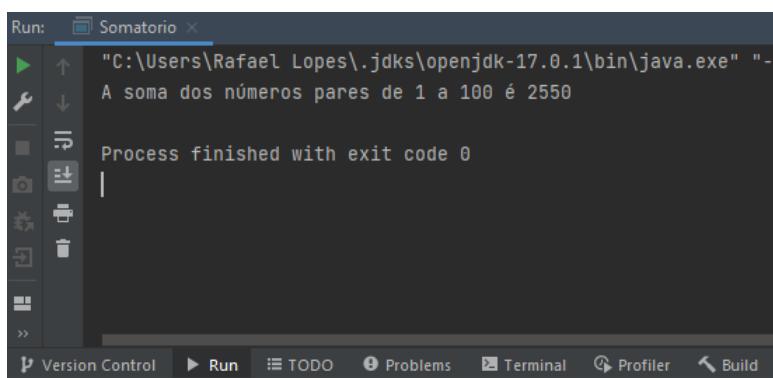
10.         }
11.         System.out.println("A soma dos números pares de 1 a 100 é " + acumulador);
12.     } // fim do método main
13. } // fim da classe Somatorio

```

Código do programa somatório.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, ele irá imprimir a soma dos números **pares** de **1 a 100** no terminal.



A instrução na **linha 4**:

```
int acumulador = 0; // Declara e inicializa o acumulador
```

é onde **declaramos** e **inicializamos** o nosso **acumulador**.



Importante!

Se nosso **acumulador** for usado para fazer um **somatório**, devemos **inicializá-lo** com o **valor neutro da operação de adição**, que é **0 (zero)**. Se nosso acumulador for usado para fazer um **produtório**, devemos inicializá-lo com o valor **neutro da operação de multiplicação**, que é **1 (um)**.

```
while (contador <= 100)
```

é **falsa** quando o contador assumir **valores acima de 100**.

Dentro do bloco de instruções do **while**, na **linha 7**, temos o teste lógico para verificar se o número é par, na **linha 8**, a instrução:

```
acumulador = acumulador + contador; // adiciona o contador par ao acumulador
```

realiza a **soma acumulativa**.

Variável de controle

Variável de controle é usado para **controlar as repetições**, quando não sabemos o número exato de **repetições do laço**. Nesse caso, o **loop é finalizado assim que a variável tem um determinado valor**.



Vamos praticar!

Vamos fazer um programa que **leia o nome e a idade de um grupo de pessoas** e mostre o **nome da pessoa novamente** se ela for **maior de idade**. Além disso, como não sabemos o total de pessoas **vamos perguntar se o usuário deseja continuar ou não a digitar as idades**. Assim, a cada iteração pergunte se o usuário se deseja continuar:

Mostre as opções “**s**” ou “**n**”

- caso digite “**s**” continue o loop;
- caso digite “**n**” encerre o loop.

Siga os passos para criar o projeto:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_15_Var_Control**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Controle** e insira o código abaixo:

```

1. import java.util.Scanner;
2.
3. public class Controle {
4.
5.     public static void main(String[] args) {
6.
7.         Scanner entrada = new Scanner(System.in);
8.
9.         char resp; // declara variavel de controle
10.        String nome;
11.        int idade;
12.        resp = 's'; // inicializa a variavel de controle
13.
14.        while (resp == 's') {
15.            System.out.println("Digite o seu nome:");
16.            nome = entrada.next();
17.            System.out.println("Digite sua idade:");
18.            idade = entrada.nextInt();

```

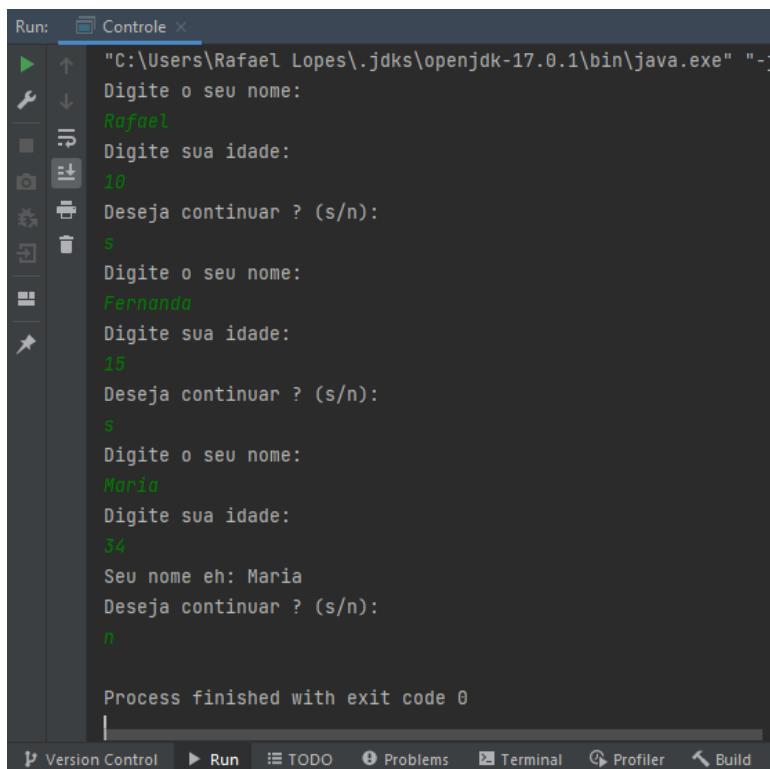
```

19.
20.         if (idade >= 18) {
21.             System.out.println("Seu nome eh: " + nome);
22.         }
23.
24.         System.out.println("Deseja continuar ? (s/n):");
25.         resp = entrada.next().charAt(0);
26.     }
27.     entrada.close();
28. } // fim do método main
29. } // fim da classe Controle

```

Código do programa Controle.

Ao executar o programa, você pode digitar quantos **nomes e idades** quiser até **selecionar n** para **parar o laço de repetição**.



```

Run: Controle x
C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" "-Dfile.encoding=UTF8" -jar C:\Users\Rafael Lopes\OneDrive\Área de Trabalho\Controle.jar
Digite o seu nome:
Rafael
Digite sua idade:
10
Deseja continuar ? (s/n):
s
Digite o seu nome:
Fernanda
Digite sua idade:
15
Deseja continuar ? (s/n):
s
Digite o seu nome:
Maria
Digite sua idade:
34
Seu nome eh: Maria
Deseja continuar ? (s/n):
n

Process finished with exit code 0

```

A instrução na **linha 7**:

```
char resp; // declara variavel de controle
```

é onde declaramos a **variável de controle**. E na **linha 10**, a instrução:

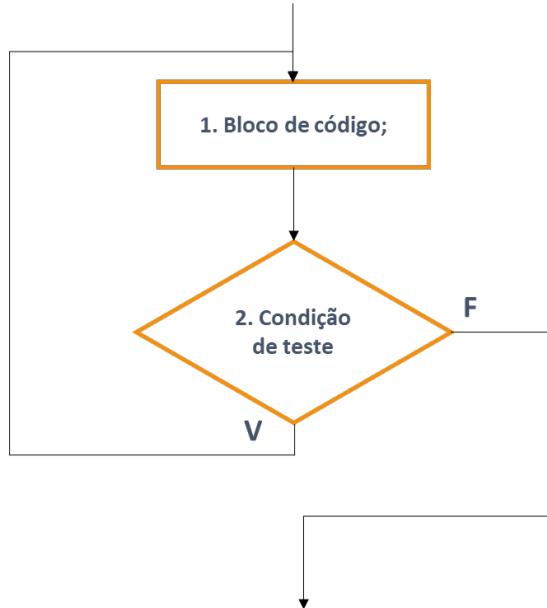
```
resp = 's'; // inicializa a variavel de controle
```

é onde inicializamos com o valor '**s**', que torna a condição do **while (linha 14)** **verdadeira**.

Na **linha 25**, lemos a **resposta do usuário** se ele deseja **continuar** ou **não**, enquanto ele digitar '**s**' o **loop continua repetindo as instruções** e quando o usuário digitar '**n**' o loop é **encerado**.

Comando do ... while

Esse tipo de **estrutura de repetição** é caracterizado por fazer o **teste de controle no final do bloco de comandos**. Os comandos repetidos são executados pelos menos uma vez antes da condição ser testada. A condição é testada no final e sempre após a execução do bloco do laço.



A sintaxe do comando do-while é:

```

do {
  instrucao1;
  instrucao2;
  instrucao3;
} while (condicao); ] Verifica a condição depois de executar
  proxima_instrucao; o bloco de instruções.
  
```



Vamos praticar!

Vamos criar um programa para **ler diversas notas e faça a soma acumulada das notas digitadas**. O loop deverá se repetir **enquanto a nota digitada for maior ou igual a zero e menor ou igual a dez**, caso **contrário** o loop deverá ser **encerrado e a soma das notas deverá ser exibida**.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

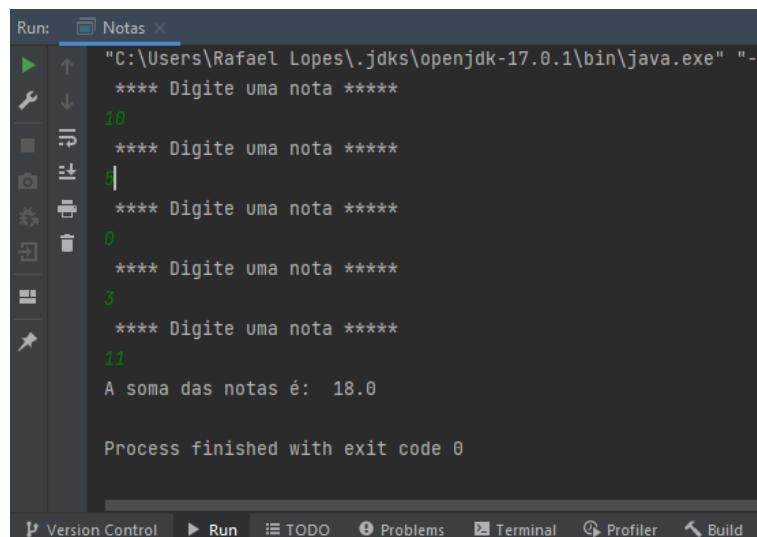
Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_16_Do_While**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Notas** e insira o código abaixo.

```
1. import java.util.Scanner;
2.
3. public class Notas {
4.     public static void main(String[] args) {
5.         Scanner entrada = new Scanner(System.in);
6.         double nota, soma = 0;
7.         do {
8.             System.out.println(" **** Digite uma nota ***** ");
9.             nota = entrada.nextDouble();
10.
11.             if (nota >= 0 && nota <= 10) soma += nota;
12.
13.         } while (nota >= 0 && nota <= 10);
14.         System.out.println("A soma das notas é: " + soma);
15.         entrada.close();
16.     } // fim do método main
17. } // fim da classe notas
```

Código utilizando o **do-while**.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.



```
Run: Notas
"C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" "-Dfile.encoding=UTF8" -cp .
**** Digite uma nota *****
10
**** Digite uma nota *****
0
**** Digite uma nota *****
3
**** Digite uma nota *****
11
A soma das notas é: 18.0

Process finished with exit code 0
```

Na **linha 14**, o **do-while** testa a condição para ver se repete o bloco de instruções ou se encerra o loop, ou seja, o **do-while** executa pelo menos uma vez o bloco de instruções antes de testar a condição de repetição.



Misturando o aprendizado

Vamos dar um **upgrade** no programa de **adivinhações com valor randômico**. Agora o programa só irá **parar de executar quando você acertar o valor sorteado**. E no final ele irá imprimir o **número total de tentativas**.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_17_Adivinha_Up**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **AdivinhaUP** e insira o código abaixo.

```

1. import java.util.Scanner; // Import necessário para utilizar a classe Scanner
2. import java.util.Random; // Import necessário para utilizar a classe Random
3.
4. public class AdivinhaUP {
5.     public static void main(String[] args) {
6.         // Instância (buffer) utilizando a classe Scanner
7.         Scanner entrada = new Scanner(System.in);
8.         // Instância utilizando a classe Random
9.         Random rand = new Random();
10.        int palpate, num = 0;
11.        num = rand.nextInt(10) + 1;
12.
13.        do {
14.            System.out.println("Digite seu palpate:");
15.            palpate = entrada.nextInt();
16.            if (palpate == num) {
17.                System.out.println("Você acertou!!!");
18.            } else {
19.                if (palpate < num) {
20.                    System.out.println("Seu palpate está abaixo.");
21.                } else {
22.                    System.out.println("Seu palpate está acima.");
23.                }
24.            }
25.        } while (palpate != num);
26.
27.        entrada.close();
28.    }
29. } // fim do método main

```

Código do upgrade do programa adivinhação.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, você pode dar um **palpite** até **acertar o número sorteado**.

```
Run:  AdivinhaUP ×
▶  "C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" "-Djava.awt.headless=true" AdivinhaUP
Digite seu palpite:
5
Seu palpito está acima.
Digite seu palpite:
4
Seu palpito está acima.
Digite seu palpite:
3
Seu palpito está acima.
Digite seu palpite:
2
Você acertou!!!
Process finished with exit code 0
```



Estrutura de Repetição for

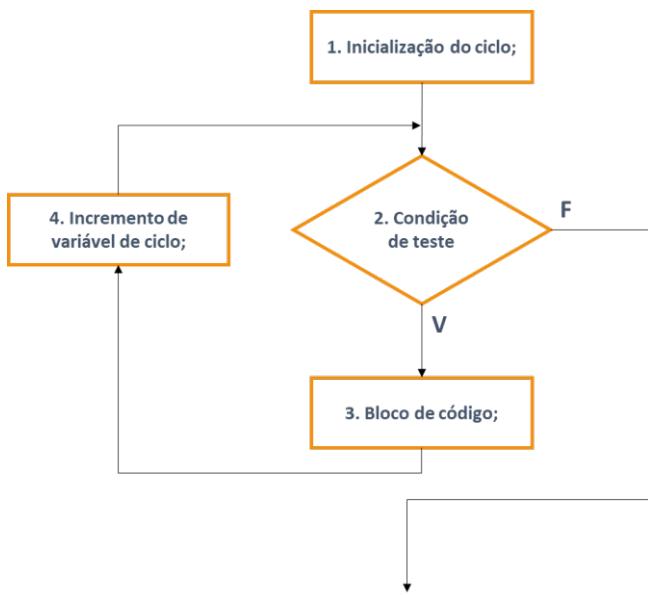
Os objetivos desta aula são:

- Compreender as estruturas de repetição;
- Conhecer o comando for;
- Diferenciar os comando for e while.

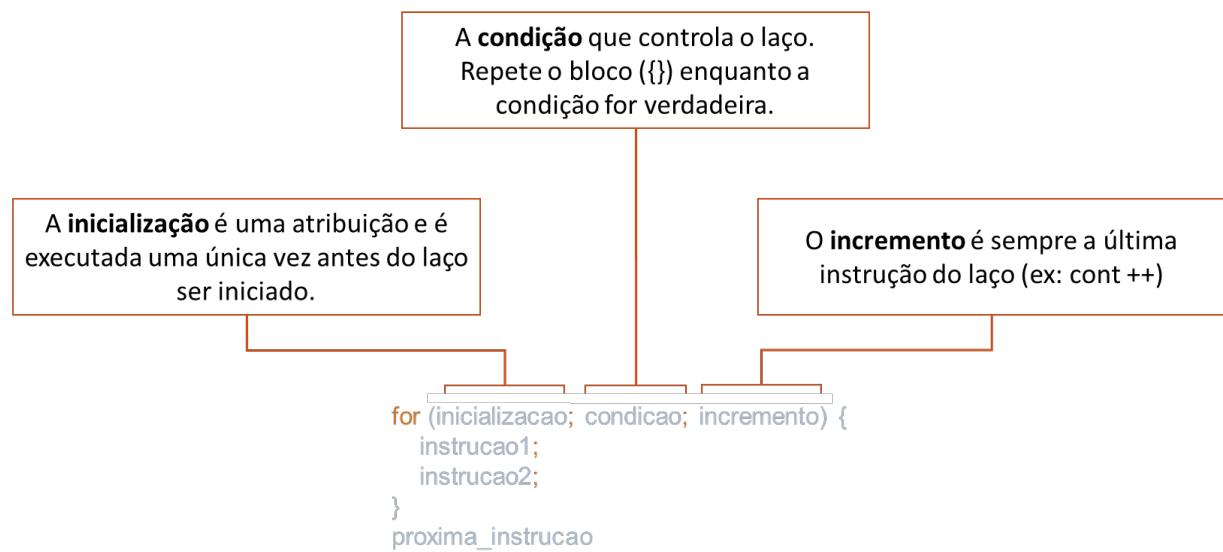
Bons estudos!

Estruturas de repetição for

O **for** (que significa para) também realiza o **teste lógico no início do laço**. Esse comando tem uma **estrutura um pouco diferente do while**, mas funciona **da mesma maneira**. Ou seja, o programa **não executará nenhuma repetição** (ações programadas) sem antes **testar a condição**.



A estrutura de repetição **for** é usada quando **queremos repetir um conjunto de instruções múltiplas vezes** e muito útil quando **sabemos o número de vezes que as instruções são repetidas**. No comando **for não temos um contador interno** e a sua sintaxe é:



Se o **laço de repetição** tiver **apenas uma instrução**, não precisamos de usar a **abertura e fechamento das chaves**.

```
for (initializacao; condicao; incremento) {
    instrucao1;
```

As operações de **incremento** e **decremento** podem ser realizadas:

- Completas (Instrução comum)
- Com os operadores de **incremento** e **decremento**.
- Ou usando **atribuição composta, contração** da **operação**

Instrução comum	Operador de incremento/decremento	Atribuição composta
<code>cont = cont + 1;</code>	<code>cont++;</code>	<code>cont +=1;</code>
<code>cont = cont - 1;</code>	<code>cont--;</code>	<code>cont-=1;</code>
<code>cont = cont + 5;</code>		<code>cont+=5;</code>
<code>cont = cont - 2;</code>		<code>cont-=2;</code>
<code>mult = mult * 3;</code>		<code>mult*=3;</code>



Vamos praticar!

Vamos fazer um programa que **leia a idade de 5 pessoas**. Siga os passos para criar o projeto: Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_18_For**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Exemplolidade** e insira o código abaixo:

```
1. import java.util.Scanner;
2.
3. public class ExemploIdade {
4.     public static void main(String[] args) {
5.         Scanner entrada = new Scanner(System.in);
6.
7.         int idade, acumuladorIdades = 0; // acumulador
8.         int contador; // declara o contador
9.
10.        for (contador = 0; contador < 5; contador++) {
```

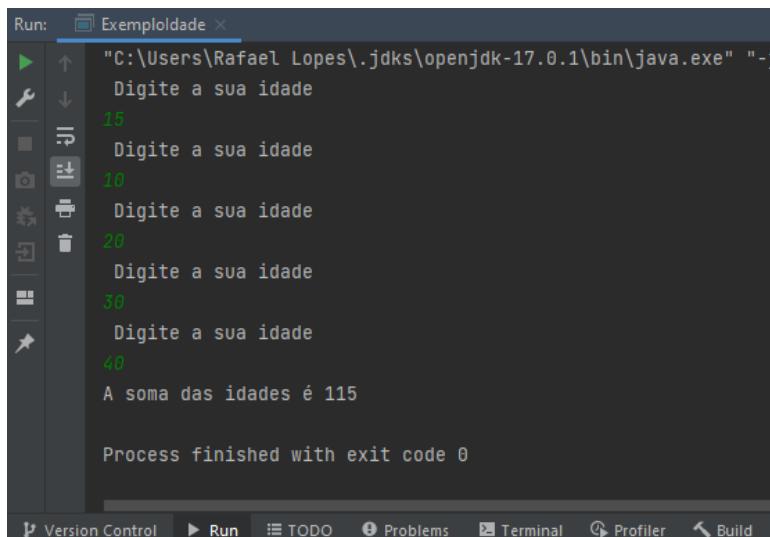
```

11.         System.out.println(" Digite a sua idade ");
12.         idade = entrada.nextInt();
13.         acumuladorIdades += idade; // acumula as idades
14.     }
15.
16.     System.out.println("A soma das idades é " + acumuladorIdades);
17.     entrada.close();
18. }
19. }
```

Código do programa utilizando **for**.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, ele irá **solicitar 5 idades** e **imprimir a soma dessas idades** no terminal.



```

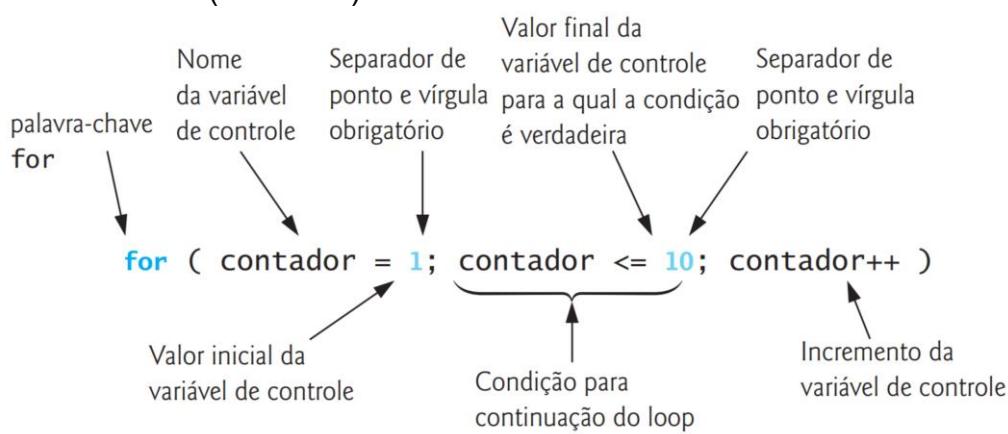
Run: Exemplodeidade
C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" "-Dfile.encoding=UTF8" -jar C:\Users\Rafael Lopes\IdeaProjects\Exemplodeidade\out\artifacts\Exemplodeidade_jar\Exemplodeidade.jar
Digite a sua idade
15
Digite a sua idade
10
Digite a sua idade
10
Digite a sua idade
20
Digite a sua idade
30
Digite a sua idade
40
A soma das idades é 115

Process finished with exit code 0
```

A instrução na **linha 10**:

```
for (contador = 0; contador < 5; contador++)
```

esse é o nosso comando **for** com a **inicialização** (`contador = 0;`), **condição testada** (`contador < 5;`) e **incremento** (`contador++`).



Sendo assim o **bloco de comandos** dentro do **for** é repetido por **5 vezes** e assim solicitando o usuário entrar com **5 valores de idades**.



Vamos praticar!

Vamos fazer um programa para exibir os **múltiplos** de **2 menores que 1000**. Siga os passos para criar o projeto:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_19_Multiplos**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Multiplos2** e insira o código abaixo:

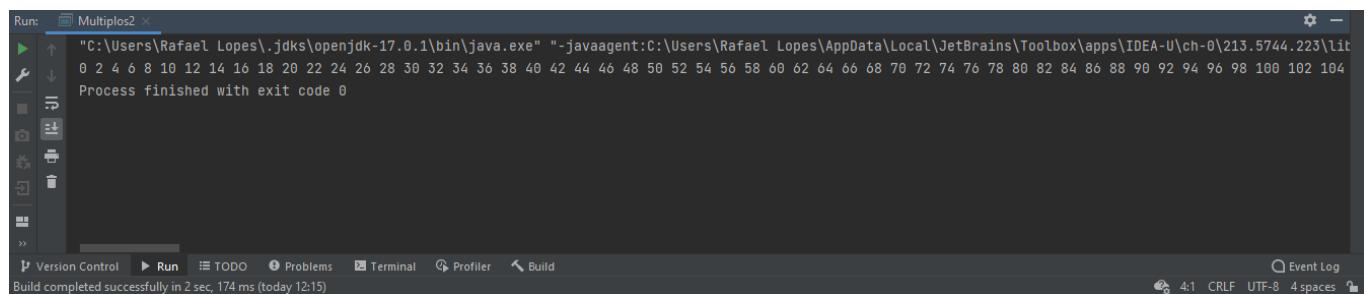
```

1. public class Multiplos2 {
2.     public static void main(String[] args) {
3.
4.         int numero;
5.
6.         for (numero = 0; numero < 1000; numero += 2) {
7.             System.out.print(numero + " ");
8.         }
9.     }
10. }
```

Código do programa múltiplo de 2.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, ele irá **imprimir a soma dos múltiplos** de **2 menores** do que **1000**.



The screenshot shows the IntelliJ IDEA interface with the 'Run' tab selected. The 'Run' dropdown menu is open, and 'Multiplos2' is chosen. The output window displays the following text:

```
"C:\Users\Rafael Lopes\jdks\openjdk-17.0.1\bin\java.exe" "-javaagent:C:\Users\Rafael Lopes\AppData\Local\JetBrains\Toolbox\apps\IDEA-U\ch-0\213.5744.223\lib
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100 102 104
Process finished with exit code 0
```

The bottom status bar indicates: 'Build completed successfully in 2 sec, 174 ms (today 12:15)' and 'Event Log'.

For x while

O comando **for** e o comando **while** tem **muitas semelhanças**. Quando usamos esses **comandos** para executar as instruções um número conhecido de vezes, ambos têm a **inicialização da variável de controle, uma condição testada e o incremento ou decremento** da variável de controle para garantir o término do laço de repetição.

```

for (inicializacao; condicao; incremento) {
    instrucao1;
    instrucao2;
}
while (condicao) {
    instrucao1;
    instrucao2;
}
incremento;
```



Vamos praticar!

Vamos fazer um programa que contenha um comando **for** e um comando **while** para executar as **mesmas operações**.

Assim, poderemos ver como é possível usar qualquer um dos dois comandos para executar ações um número de vezes limitada.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_20_For_While**

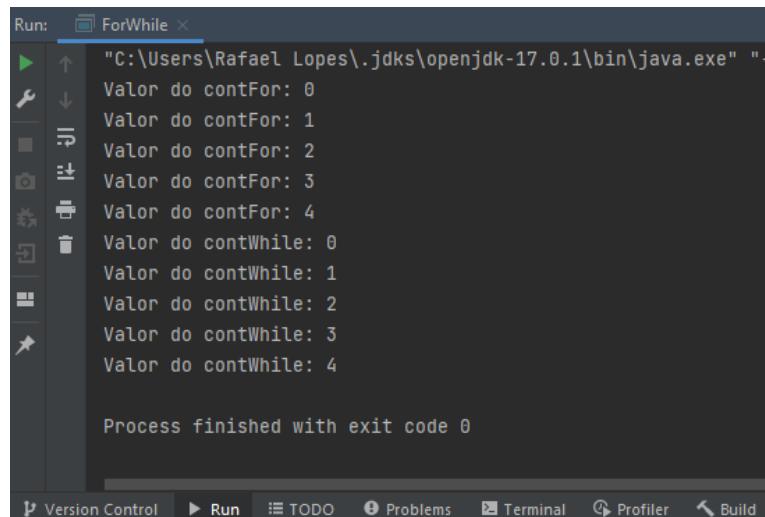
No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **ForWhile** e insira o código abaixo.

```

1. public class Forwhile {
2.     public static void main(String[] args) {
3.
4.         int contFor, contWhile;
5.
6.         //for(inicialização; condição; incremento)
7.         for(contFor = 0; contFor < 5; contFor++)
8.             System.out.println("Valor do contFor: " + contFor);
9.
10.        contWhile = 0; // Inicialização do while
11.
12.        //while(condição)
13.        while(contWhile < 5) {
14.            System.out.println("Valor do contWhile: " + contWhile);
15.            contWhile++; // Incremento do while
16.        }
17.    }
18.}
```

Código do programa ForWhile.

Ao executar o programa, ele irá imprimir os valores das variáveis de controle do **for** e do **while**.



```
Run: ForWhile x
C:\Users\Rafael Lopes\.jdks\openjdk-17.0.1\bin\java.exe" -
Valor do contFor: 0
Valor do contFor: 1
Valor do contFor: 2
Valor do contFor: 3
Valor do contFor: 4
Valor do contWhile: 0
Valor do contWhile: 1
Valor do contWhile: 2
Valor do contWhile: 3
Valor do contWhile: 4

Process finished with exit code 0
```

Version Control Run TODO Problems Terminal Profiler Build

A instrução na **linha 7**:

```
for(contFor = 0; contFor < 5; contFor)
```

mostra o comando **for** com a **inicialização da variável de controle**, a **condição** e o **incremento da variável de controle**. O mesmo pode ser visto no comando **while**, porém as instruções estão em **diversas linhas**, a **inicialização** na **linha 10**:

```
contWhile = 0; // Inicialização do while
```

a **condição** na **linha 13**:

```
while(contWhile < 5)
```

e o **incremento** na **linha 15**:

```
contWhile++; // Incremento do while
```



Vamos praticar!

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_21_For_While_2**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **ForWhile2** e insira o código abaixo.

```
1. public class ForWhile2 {  
2.     public static void main(String[] args) {  
3.  
4.         int contadorFor, contadorWhile;  
5.  
6.         //for(inicialização; condição; decremento)  
7.         for(contadorFor = 10; contadorFor > 0; contadorFor--)  
8.             System.out.print("*");  
9.  
10.        System.out.println("");  
11.        contadorWhile = 10; // Inicialização do while  
12.  
13.        //while(condição)  
14.        while(contadorWhile > 0) {  
15.            System.out.print("*");  
16.            contadorWhile--; // Decremento do while  
17.        }  
18.    }  
19.}
```

Código do programa ForWhile2.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

A instrução na **linha 7**:

```
for(contadorFor = 10; contadorFor > 0; contadorFor--)
```

mostra o comando **for** com a **inicialização da variável de controle**, a **condição** e o **decremento** da **variável de controle**. O mesmo pode ser visto no comando **while**, porém as instruções estão em diversas linhas, a **inicialização na linha 11**:

```
contadorWhile = 10; // Inicialização do while
```

a condição na **linha 14**:

```
while(contadorWhile > 0)
```

e o incremento na **linha 16**:

```
contadorWhile--; // Decremento do while
```

Algumas variáveis de controle usadas no comando **for**

Os exemplos a seguir mostram métodos para a **alteração da variável de controle** em uma estrutura **for**.

- `for (i=1; i<=100; i++)` | Alterne a variável de controle de 1 a 100 em incrementos de 1.
- `for (i=100; i>=1; i--)` | Alterne a variável de controle de 100 a 1 em decremento de 1.
- `for (i=7; i<=77; i+=7)` | Alterne a variável de controle de 7 a 77 em incrementos de 7.
- `for (i=20; i>=2; i -=2)` | Alterne a variável de controle de 20 a 2 em intervalos de -2.
- `for (j=2; j<=17; i +=3)` | Alterne a variável de controle na seguinte sequencia de valores: 2,5,8,11,14,17.
- `for (j=44; j>=0; i -=11)` | Alterne a variável de controle na seguinte sequencia de valores: 44,33,22,11,0.



Teste de Mesa

Os objetivos desta aula são:

- Compreender os testes de mesa;
- Conhecer a ferramenta de debug do IntelliJ;

Bons estudos!

Teste de mesa

O **teste de mesa** é uma análise para descobrir se **um programa funciona logicamente**. Basicamente, o **Teste de Mesa (Trace Table)** é um processo manual que é utilizado para validar a **lógica de um determinado algoritmo**. Esse processo era realizado com **papel e caneta** em cima de **uma mesa** e a **pessoa que programou** ia **seguindo o fluxo do algoritmo** e **atualizando os valores das variáveis** do programa **manualmente** a cada instrução do algoritmo que era analisada.

Esse teste é utilizado principalmente em algoritmos quando a linguagem utilizada não possui **nenhuma ferramenta automatizada de depuração**. Como as **linguagens de programação atuais costumam possuir tais ferramentas**, é mais comum utilizá-las a fazer o teste de mesa propriamente dito, embora para quem ainda é iniciante, recomendamos utilizá-lo, visto que provavelmente não terá domínio sobre a ferramenta de depuração.

Então essa parte da disciplina de programação em Java pode ser vista mais como um **desenvolvimento de habilidades de programação** para visualizar como as variáveis são **atualizadas a cada iteração de um loop**, por exemplo. Não vamos ter que fazer teste de mesa manualmente.



Vamos praticar!

Vamos fazer um programa que tenha **uma variável de controle**, **uma variável acumuladora** e a **condição testada de um laço de repetição**.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_22_Teste_de_Mesa**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Teste** e insira o código abaixo:

```

1. public class Teste {
2.     public static void main(String[] args) {
3.         int controle, acumulador = 0;
4.         boolean condicao;
5.         System.out.println("Número da iteração\tCondição\tControle\tAcumulador");
6.         for (controle = 1; controle <= 10; controle++) {
7.             acumulador += controle;
8.             condicao = controle <= 10;
9.             System.out.println("Iteração " + controle + "\t\t\t" + condicao + "\t\t\t"
+ controle + "\t\t\t" + acumulador);
}

```

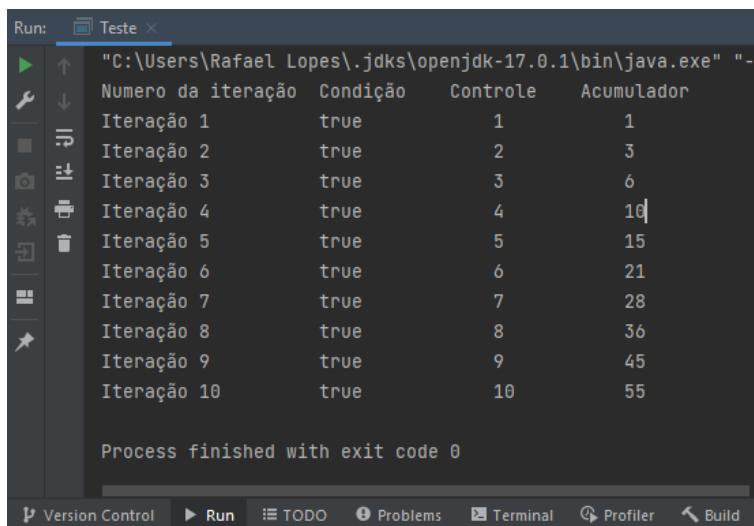
```

10.      }
11.  } // fim do método main
12. }
```

Código do programa teste de mesa.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, você consegue visualizar o valor de **cada variável** em cada **iteração do laço de repetição**.



Iteração	Condição	Controle	Acumulador
1	true	1	1
2	true	2	3
3	true	3	6
4	true	4	10
5	true	5	15
6	true	6	21
7	true	7	28
8	true	8	36
9	true	9	45
10	true	10	55

Process finished with exit code 0

Observe que quando a **variável de controle** assumiu o **valor 11**, a **condição testada** se tornou **falsa** e então as **instruções** dentro do **for** não foram **executadas mais**. Por isso, paramos de **imprimir na iteração 10** e mostrando que nesse momento a condição **ainda era verdadeira**.



Vamos praticar!

Vamos fazer um programa de teste de mesa para o comando while:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_23_Teste_de_Mesa**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Teste2** e insira o código abaixo:

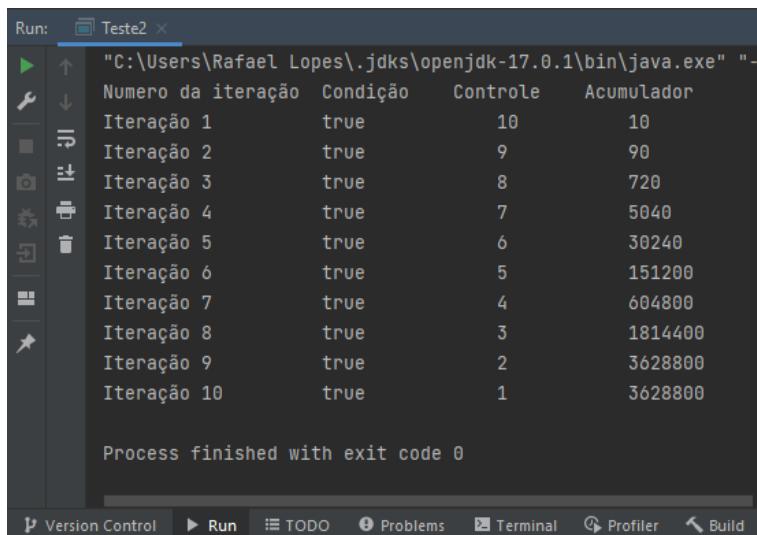
```

1. public class Teste2 {
2.     public static void main(String[] args) {
3.         int controle = 10;
4.         long acumulador = 1;
5.         boolean condicao;
6.         System.out.println("Número da iteração\tCondição\tControle\tAcumulador");
7.         while(controle > 0){
8.             int iteracao = 10 - controle + 1;
9.             acumulador *= controle;
10.            condicao = controle > 0;
11.            System.out.println("Iteração " + iteracao+ "\t\t\t" + condicao + "\t\t\t" +
    controle + "\t\t\t" + acumulador);
12.            controle--;
13.        }
14.    } // fim do método main
15. }
```

Código do programa do teste de mesa com o while.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, você consegue visualizar o **valor de cada variável** em cada **iteração** do **laço de repetição**.



Iteração	Condição	Controle	Acumulador
1	true	10	10
2	true	9	90
3	true	8	720
4	true	7	5040
5	true	6	30240
6	true	5	151200
7	true	4	604800
8	true	3	1814400
9	true	2	3628800
10	true	1	3628800

Process finished with exit code 0



Vamos praticar!

Vamos fazer um programa de teste de mesa para o comando do-while.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_24_Teste_de_Mesa**

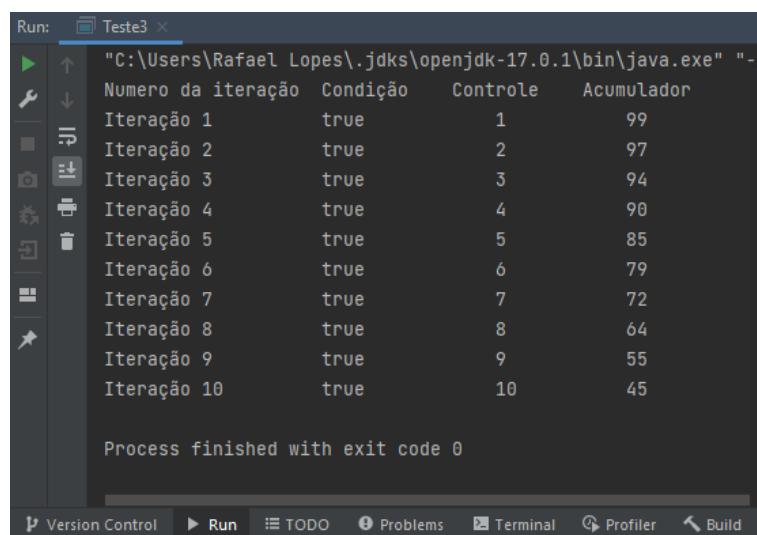
No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Teste3** e insira o código abaixo.

```

1. public class Teste3 {
2.     public static void main(String[] args) {
3.         int controle = 1;
4.         long acumulador = 100;
5.         boolean condicao;
6.         System.out.println("Número da iteração\tCondição\tControle\tAcumulador");
7.         do{
8.             acumulador -= controle;
9.             condicao = controle <=10;
10.            System.out.println("Iteração " + controle+ "\t\t\t" + condicao + "\t\t\t" +
    controle + "\t\t\t" + acumulador);
11.            controle++;
12.        }while(controle <=10);
13.    } // fim do método main
14. }
```

Código do programa teste de mesa do-while.

Ao executar o programa, ele irá **imprimir** os valores das **variáveis de controle** do **for** e do **while**.



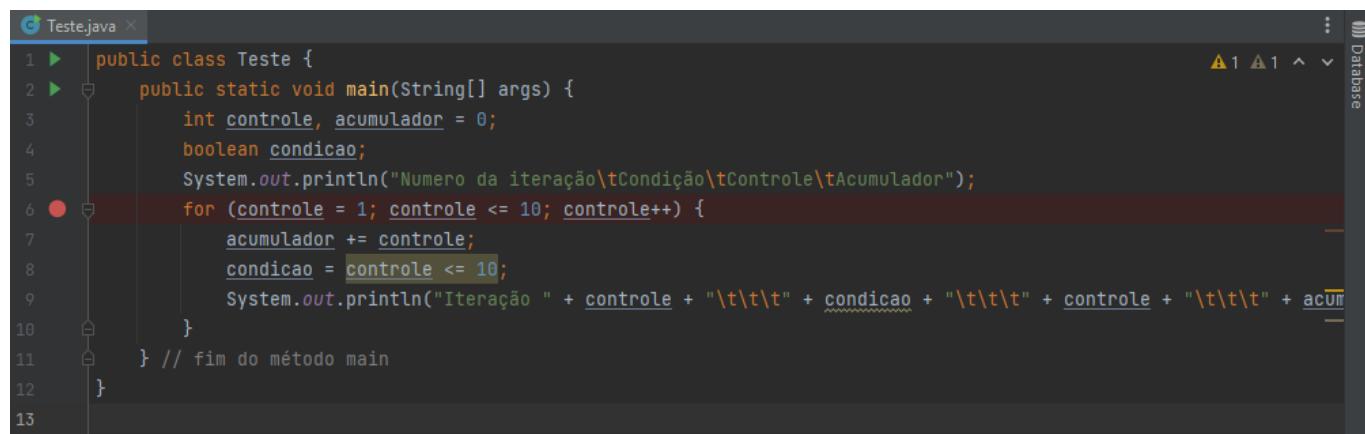
Iteração	Condição	Controle	Acumulador
1	true	1	99
2	true	2	97
3	true	3	94
4	true	4	90
5	true	5	85
6	true	6	79
7	true	7	72
8	true	8	64
9	true	9	55
10	true	10	45

Process finished with exit code 0

Ferramentas de debug do IntelliJ

O teste de mesa é **interessante quando você não tiver as ferramentas de debug em uma IDE**, mas se sua **IDE já possui esse tipo de ferramenta**, ele se torna apenas uma brincadeira para **testarmos instruções e formatar saídas**. Isso porque, é **muito mais útil você aprender a utilizar as ferramentas de depuração (debug)** disponíveis na **IDE** que você está utilizando, do que inserir instruções no código que não são necessárias para o funcionamento principal do programa.

Na **IDE IntelliJ**, você precisa marcar **um ou mais breakpoint (ponto de parrada)**. Você pode fazer isso clicando **duas vezes ao lado do número da linha desejada**. Por exemplo, no **Projeto_22_Teste_de_Mesa**. Vamos inserir um **breakpoint** na **linha 6**, onde está o comando **for**:

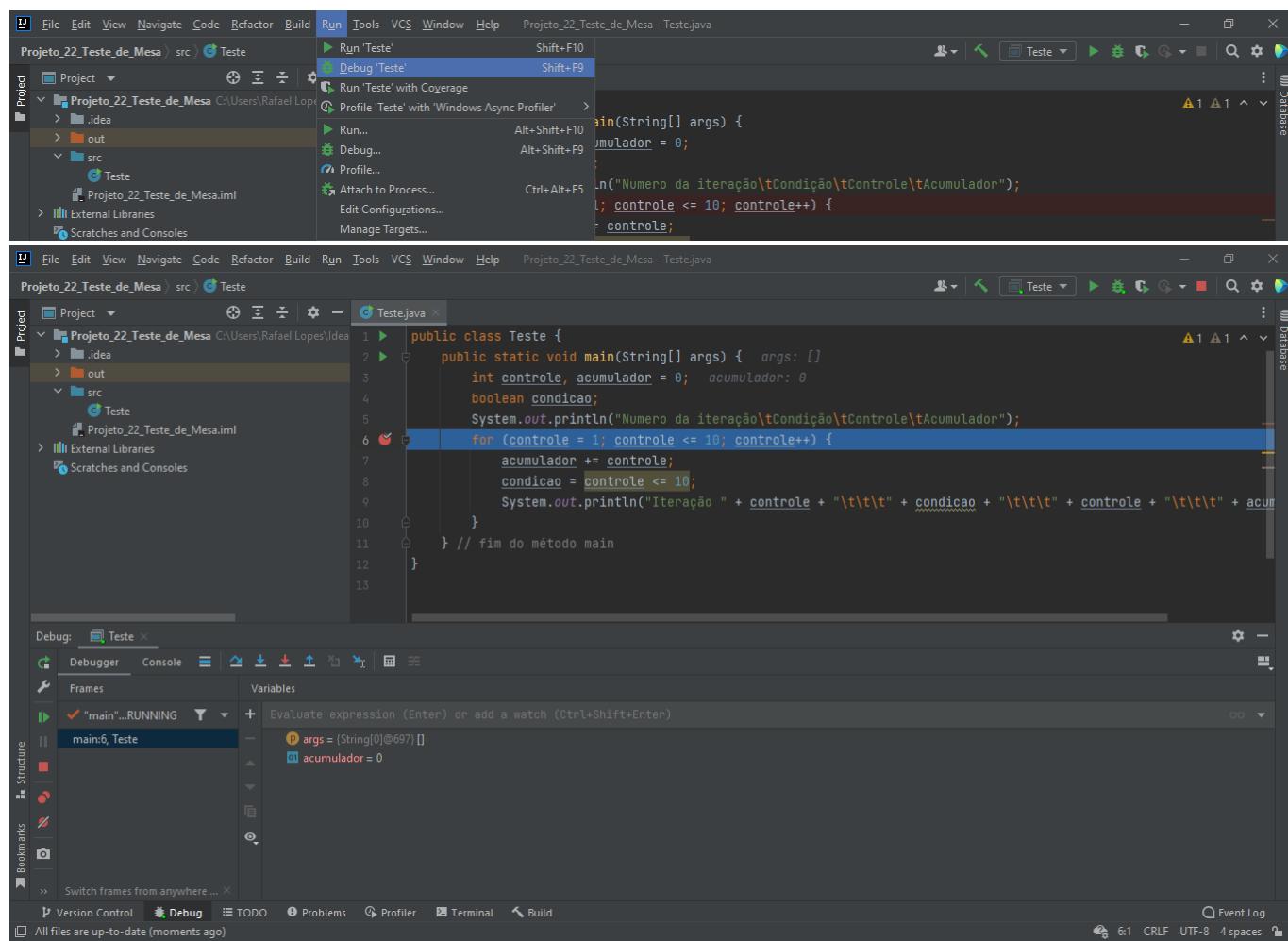


```

1 public class Teste {
2     public static void main(String[] args) {
3         int controle, acumulador = 0;
4         boolean condicao;
5         System.out.println("Número da iteração\tCondição\tControle\tAcumulador");
6         for (controle = 1; controle <= 10; controle++) {
7             acumulador += controle;
8             condicao = controle <= 10;
9             System.out.println("Iteração " + controle + "\t\t\t" + condicao + "\t\t\t" + controle + "\t\t\t" + acumulador);
10        }
11    } // fim do método main
12 }

```

Agora, você pode acessar a ferramenta de **debug** no Menu **Run → Debug**



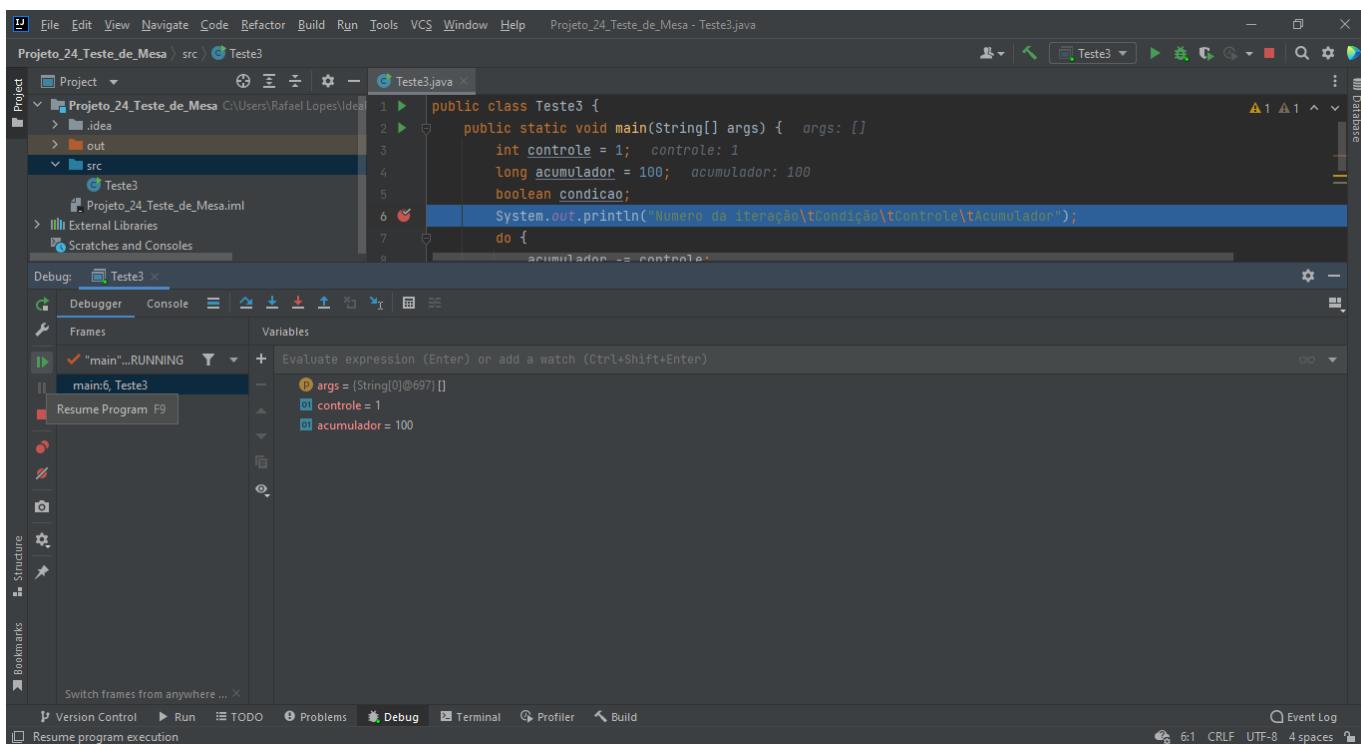
Run → **Debug 'Teste'**

```

1 public class Teste {
2     public static void main(String[] args) {
3         int controle, acumulador = 0;
4         boolean condicao;
5         System.out.println("Número da iteração\tCondição\tControle\tAcumulador");
6         for (controle = 1; controle <= 10; controle++) {
7             acumulador += controle;
8             condicao = controle <= 10;
9             System.out.println("Iteração " + controle + "\t\t\t" + condicao + "\t\t\t" + controle + "\t\t\t" + acumulador);
10        }
11    } // fim do método main
12 }

```

Observe que ao acessar o **Debug** do programa, as variáveis que já foram **inicializadas** com algum valor já aparecem na **janela do depurador**. Se você clicar **uma vez** no botão **Resume Program** (tecla de atalho F9), o programa irá executar uma **iteração do for** e parar na **linha novamente**.



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project:** Projeto_24_Teste_de_Mesa
- File:** Teste3.java
- Code:**

```
public class Teste3 {
    public static void main(String[] args) { args: []
        int controle = 1; controle: 1
        long acumulador = 100; acumulador: 100
        boolean condicao;
        System.out.println("Número da iteração\tCondição\tControle\tAcumulador");
        do {
            acumulador -= controle;
        }
    }
}
```
- Variables:** Shows args = [String@697], controle = 1, and acumulador = 100.
- Frames:** Shows "main" RUNNING and main:6, Teste3.
- Buttons:** Includes "Resume Program" (F9).
- Bottom:** Shows the status bar with "6:1 CRLF UTF-8 4 spaces".

Você pode clicar **várias vezes** até acabar as **iterações** do comando **for**:



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project:** Projeto_24_Teste_de_Mesa
- File:** Teste.java
- Code:**

```
public class Teste {
    public static void main(String[] args) { args: []
        int controle, acumulador = 0; acumulador: 0 controle: 1
        boolean condicao; condicao: true
        System.out.println("Número da iteração\tCondição\tControle\tAcumulador");
        for (controle = 1; controle <= 10; controle++) { controle: 1
            acumulador += controle;
            condicao = controle <= 10;
            System.out.println("Iteração " + controle + "\t\t\t" + condicao + "\t\t\t" + controle + "\t\t\t" + acumulador);
        }
    }
}
```
- Variables:** Shows args = [String@697], controle = 1, and acumulador = 0.
- Frames:** Shows "main" RUNNING and main:1, Teste.
- Buttons:** Includes "Resume Program" (F9).
- Bottom:** Shows the status bar with "6:1 CRLF UTF-8 4 spaces".

Se clicar mais uma vez, o debug chegará ao **final do programa**, terminando assim a sua **execução**.



Os objetivos desta aula são:

- Compreender as vantagens de usar vetores;
- Percorrer vetores com o uso do comando for;

Bons estudos!

Vetores

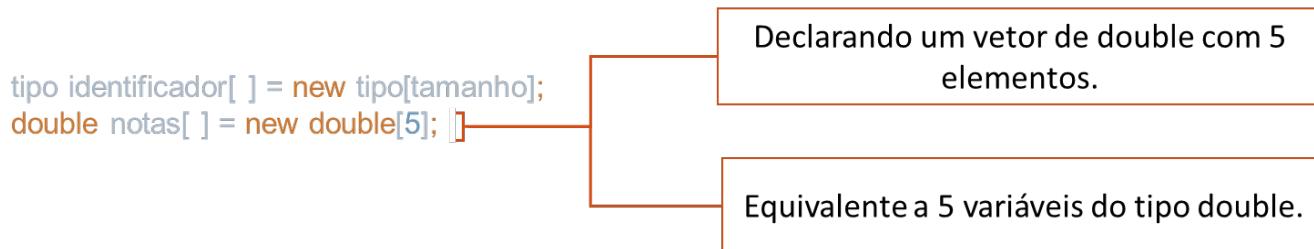
Nós usamos variáveis para **armazenar os dados inseridos pelo usuário** ou resultante do **processamento de um cálculo**, etc. Um exemplo interessante foi o usado na aula 03 quando lemos duas notas de um aluno e calculamos a média dessas notas. Nesse exemplo, nós criamos duas variáveis para armazenar as duas notas e tudo funcionou bem. Mas agora, suponha que queremos armazenar agora **dez notas** ou **100 notas**, não é prático criar **10** ou **100 variáveis**. Para isso, utilizamos **vetores**.

Vetor é um **tipo especial de variável homogênea**, que possui posições **contínuas na memória**, que são **acessadas pelo mesmo nome**. Eles armazenam “**dados**” do **mesmo tipo** (**int**, **char**, **double**, etc). Exemplo: um **vetor do tipo double** para **armazenar cinco notas** dos alunos.

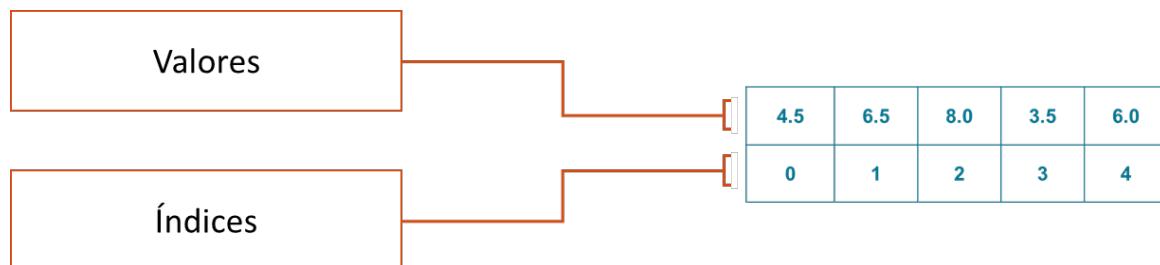
Notas

4.5	6.5	8.0	3.5	6.0
0	1	2	3	4

A sintaxe de um **vetor** é:



Para acessar um **elemento** de um **vetor** é necessário saber o **índice desse elemento**:



O índice mostra a **posição de um elemento dentro do vetor** e é necessário para manipular **cada dado do vetor**. O **primeiro elemento** do vetor **sempre tem índice zero** e o **último** é o tamanho **declarado no vetor menos um**. Por exemplo:

```
double notas[] = new double[200]
```



O tamanho do vetor é **200** posições, que **inicia no índice 0** e vai **até o índice 199**.

Podemos **declarar e inicializar** os vetores em Java passando uma **lista de valores**. A lista de valores deve estar entre **chaves** e **cada valor deve estar separados por vírgula**.

```
double notas[] = new double[ ]{1.5, 4.5, 1.2, 9.8, 9.9};  
int primos[] =new int[ ]{2, 3, 5, 7, 11, 13};  
char dias [] = new char[ ]{'d', 's', 't', 'q', 's', 's'};
```

Agora, vamos considerar o vetor:

```
double notas[] = {4.5, 6.5, 8.0, 3.5, 6.0};
```

Para **acessar um elemento do vetor** usamos o **nome do vetor** e colocamos **o índice que desejamos acessar entre colchetes**. Então para **alterar o elemento no índice zero do vetor de 4.5 para 9.0** devemos fazer a seguinte atribuição:

```
notas[0] = 9.0;
```

Se quisermos **imprimir a última nota** devemos fazer o seguinte:

```
System.out.println(notas[4]);
```

Vetores são muito úteis quando usamos **estruturas de repetição**, assim podemos **preencher o seu conteúdo** e, também, **percorrer** para **ler ou alterar os seus elementos/valores**. Por exemplo, para imprimir **múltiplos valores de um vetor**, podemos “**caminhar**” entre os **laços de um vetor** por meio de **laço de repetição**. Por exemplo, podemos usar um **for** para ler **cinco notas e armazenar em um vetor**:

```
for (int i = 0; i < 5; i++){  
    System.out.println("Digite uma nota: ");  
    nota [i] = entrada.nextDouble();  
}
```

Por exemplo, uma “**simulação**” dos valores que o usuário poderia digitar é:

> Digite uma nota:

5.5

> Digite uma nota:

6.5

> Digite uma nota:

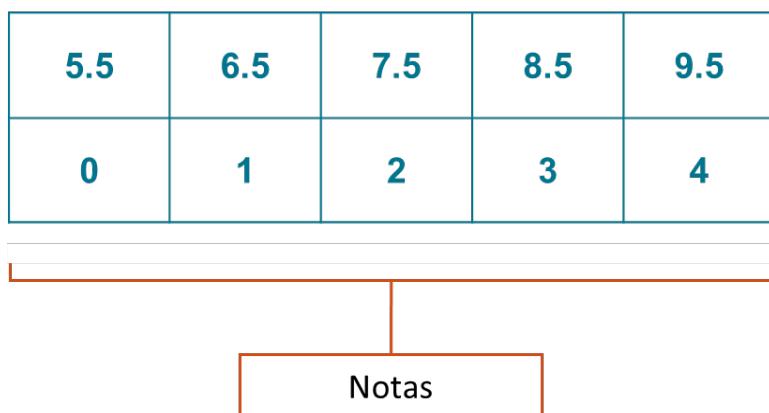
7.5

> Digite uma nota:

8.5

> Digite uma nota:

9.5



Observe que a **ordem que o usuário digita as notas** é a **ordem que elas são armazenadas no vetor**. Isso porque o nosso comando **for** percorre o vetor da posição 0 até a posição 4.

Também, podemos usar **laços de repetição** para imprimir os dados armazenados no vetor.

```
for (int i = 0; i < 5; i++) {
    System.out.println("Nota: ", nota[i]);
}
```

A saída impressa no terminal é:

> Nota: 5.5
> Nota: 6.5
> Nota: 7.5
> Nota: 8.5
> Nota: 9.5

Em Java, podemos usar a instrução **length** para descobrirmos o **comprimento de um vetor**, caso não saibamos o tamanho de um vetor. A instrução **length retorna o tamanho do vetor** e possui a seguinte sintaxe:

```
tamanho = nome_do_vetor.length;
```

O valor retornado deve ser **armazenado** em uma **variável**, que no exemplo se chama **tamanho**. Assim, podemos criar um **vetor com tamanho dinâmico** com o seguinte trecho de código:

```
Scanner entrada = new Scanner(System.in);
System.out.println("Digite o tamanho do vetor: ");
int tamanho = entrada.nextInt();

double notas[] = new double[tamanho];
int tamanhoVetor = notas.length; //Lendo o tamanho do vetor

System.out.println("O tamanho do vetor é: " + tamanhoVetor);
entrada.close();
```

A instrução **length** retorna o **tamanho do vetor** de acordo com o **valor definido pelo usuário**. Por exemplo, se o usuário digitar **30** para o **tamanho do vetor** temos a seguinte saída no terminal:

> 30

O tamanho do vetor é: 30



Vamos praticar!

Vamos fazer um programa que **leia 10 notas digitadas** pelo usuário. As notas devem ser **armazenadas em um vetor** e depois de armazenadas elas deverão **ser impressas na tela**.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_25_Vetores**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Exemplo1Vetor** e insira o código abaixo.

```
1. import java.util.Scanner;
2.
3. public class Exemplo1Vetor {
4.     public static void main(String[] args) {
5.
6.         double[] notas = new double[10];
7.
8.         Scanner entrada = new Scanner(System.in);
9.
10.        // comprimento do vetor
```

```
11.     int tamanho = notas.length;
12.
13.     // loop para leitura das 10 notas
14.     for (int i = 0; i < tamanho; i++) {
15.         System.out.println(" Digite a nota " + (i + 1));
16.         notas[i] = entrada.nextDouble();
17.     }
18.
19.     // loop para imprimir as 10 notas
20.     for (int i = 0; i < tamanho; i++) {
21.         System.out.println(" nota : " + (i + 1) + " = " + notas[i]);
22.     }
23.
24.     entrada.close();
25. }
26. }
```

Código do programa utilizando vetores.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o código, o programa irá solicitar **10 notas** e depois irá **imprimir todas as notas digitadas pelo usuário**.

```
Run: Exemplo1Vetor ×
▶ "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users\Thiago\IdeaProjects\Vetores\lib\jdwp-agent.jar=transport=dt_socket,address=8000,server=y,suspend=n
Digite a nota 1
10
Digite a nota 2
5
Digite a nota 3
4
Digite a nota 4
6
Digite a nota 5
1
Digite a nota 6
0
Digite a nota 7
7
Digite a nota 8
9
Digite a nota 9
8
Digite a nota 10
10
nota : 1 = 10.0
nota : 2 = 5.0
nota : 3 = 4.0
nota : 4 = 6.0
nota : 5 = 1.0
nota : 6 = 6.0
nota : 7 = 7.0
nota : 8 = 9.0
nota : 9 = 8.0
nota : 10 = 10.0

Version Control Run TODO Problems Build Profiler Terminal
```

Na **linha 14**, o comando **for** é usado para **percorrer o vetor** e **preenchê-lo** com o **valor** que o usuário **digitou no teclado**.

Na **linha 20**, o comando **for** é usado para **percorrer o vetor** e **ler os valores** de cada elemento para **imprimir no console**.



Vamos praticar!

Vamos fazer um programa para encontrar o maior valor em um vetor de 5 inteiros.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_26_Vetores**

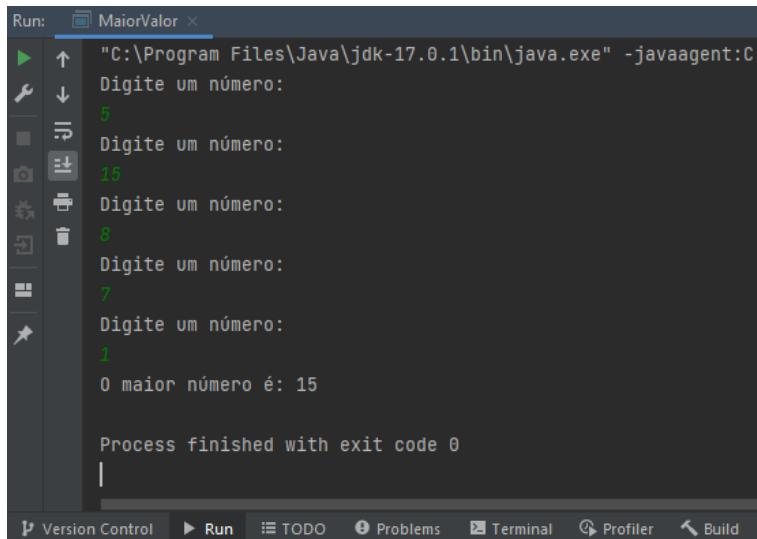
No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **MaiorValor** e insira o código abaixo.

```
1. import java.util.Scanner;
2.
3. public class MaiorValor {
4.     public static void main(String[] args) {
5.
6.         Scanner entrada = new Scanner(System.in);
7.
8.         int maior;
9.         int[] numeros = new int[5];
10.
11.        // comprimento do vetor
12.        int tamanho = numeros.length;
13.
14.        for (int i = 0; i < tamanho; i++) {
15.            System.out.println("Digite um número:");
16.            numeros[i] = entrada.nextInt();
17.        }
18.
19.        maior = numeros[0];
20.
21.        // encontra o maior número
22.        for (int i = 0; i < tamanho; i++) {
23.            if (numeros[i] > maior) {
24.                maior = numeros[i];
25.            }
26.        }
27.
28.        System.out.println("O maior número é: " + maior);
29.        entrada.close();
30.    }
31. }
```

Código do programa utilizando vetores.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, ele irá solicitar **cinco valores inteiros**, depois irá procurar o **maior valor digitado e imprimi-lo na tela**.



```
Run: MaiorValor x
↑ "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:
Digite um número:
5
Digite um número:
15
Digite um número:
8
Digite um número:
7
Digite um número:
1
O maior número é: 15

Process finished with exit code 0
```

Na **linha 22**, o comando **for** percorre o vetor e através da instrução **if** (**linha 23**) o programa **compara os valores do vetor**. Quando o **valor lido** é **maior do que o valor armazenado** na variável **maior** a condição do **if** é **verdade** e o programa **armazena esse valor na variável maior**.



Vamos praticar!

Vamos fazer um programa para **somar os valores** de um **vetor**.

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_27_Vetores**

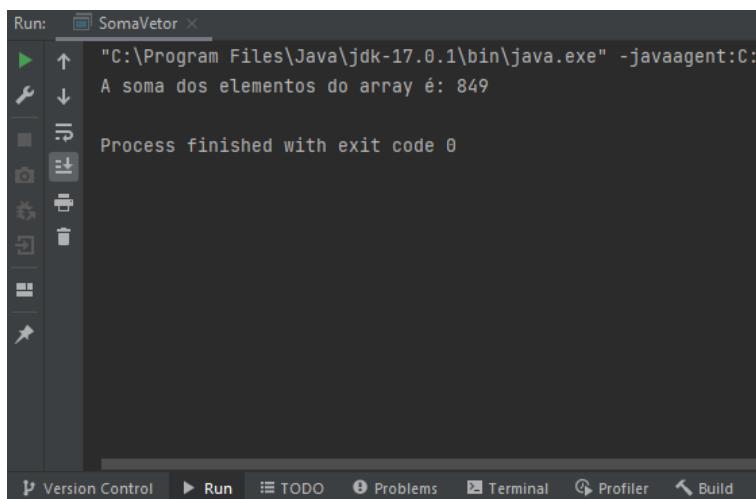
No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **SomaVetor** e insira o código abaixo.

```
1. public class SomaVetor {
2.
3.     public static void main(String[] args) {
4.         int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
5.         int total = 0;
6.
7.         // adiciona o valor de cada elemento ao total
8.         for (int counter = 0; counter < array.length; counter++)
9.             total += array[counter];
10.
11.         System.out.printf("A soma dos elementos do array é: %d%n", total);
12.     }
13. } // fim da classe SomaVetor
```

Código do programa soma elementos do vetor.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o programa, ele irá imprimir a soma de todos os elementos do vetor.



```
Run: SomaVetor ×
▶ "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:
A soma dos elementos do array é: 849
Process finished with exit code 0
```

Version Control Run TODO Problems Terminal Profiler Build

Observe na **linha 9**, temos a variável **acumuladora total**, que vai **armazenando o valor acumulativo da soma dos elementos do vetor**. Para isso, utilizamos um laço de repetição **for** e percorremos **cada elemento do vetor** para realizar a **soma acumulativa dos valores do vetor**.



Strings

Os objetivos desta aula são:

- Compreender o uso de strings;
- Conhecer os métodos utilizados para trabalhar com strings.

Bons estudos!

Strings

Strings é um **conjunto de caracteres** muito utilizado em diferentes linguagens de programação de diversas formas possíveis. Strings são **vetores homogêneos** que **armazenam dados do tipo char (caracteres)**, ou seja, **uma string pode conter letras, números e caracteres especiais** e, dessa forma, possibilita enviar **bloco de dados** e não somente **palavras** ou **frases**.

No contexto de aplicações reais, **strings são a forma de enviar informações de uma aplicação para outra**. Por exemplo:

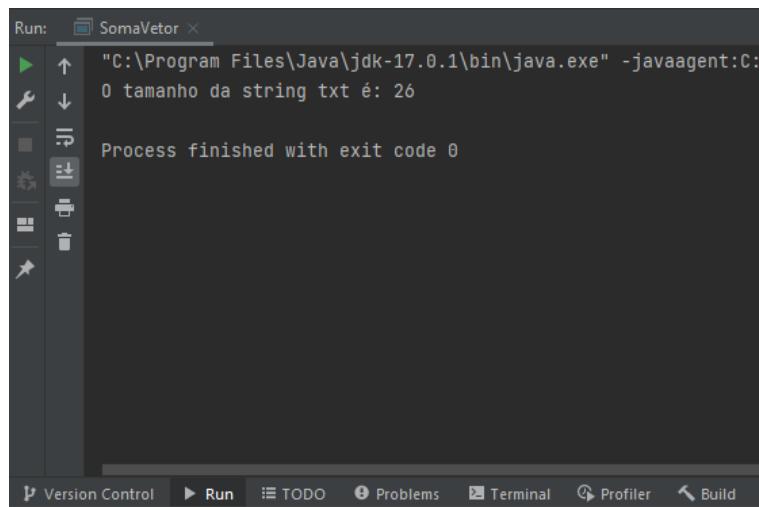
```
String greeting = "Hello";
```

Métodos para trabalhar com strings

A linguagem de programação **Java** possui **diversos métodos para manipular strings**. O primeiro deles é o método **length()**, que é usado para retornar o **tamanho de vetor ou string**.

```
String txt = "ABCDEFGHIJKLMNPQRSTUVWXYZ";
System.out.println("O tamanho da string txt é: " + txt.length());
```

A saída esperada para esse trecho de código é:



The screenshot shows a Java IDE's run console. The output window displays the following text:
Run: SomaVetor ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:
0 tamanho da string txt é: 26
Process finished with exit code 0

Temos também, os métodos **toUpperCase()** e **toLowerCase()**, que são usados para **transformar as letras da string para maiúsculas** ou para **minúsculas**, respectivamente.

Também, temos o método **indexOf()**, que retorna o **índice da primeira ocorrência de um texto específico na string (incluindo espaço em branco)**. Se o método **não encontrar o texto**, ele retornará **-1**.



Vamos praticar!

Vamos fazer um programa para trabalhar com os métodos mencionados anteriormente:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_28_Strings**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Strings** e insira o código abaixo.

```

1. public class Strings {
2.     public static void main(String[] args) {
3.         String txt = "Ola Pessoal do IOS";
4.         // Imprime o tamanho da string
5.         System.out.println("O tamanho da string txt é: " + txt.length());
6.         // Imprime os caracteres da string em maiúsculo
7.         System.out.println(txt.toUpperCase());
8.         // Imprime os caracteres da string em minúsculo
9.         System.out.println(txt.toLowerCase());
10.        // Imprime a posição da primeira ocorrência de um valor na string
11.        System.out.println(txt.indexOf("s"));
12.        System.out.println(txt.indexOf("IO"));
13.        // Não encontra o texto na string
14.        System.out.println(txt.indexOf("io")); // Case Sensitive
15.    }
16. }
```

Código do programa utilizando métodos para manipular strings.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

```

Run: Strings ×
> "C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:
0 tamanho da string txt é: 18
OLA PESSOAL DO IOS
ola pessoal do ios
6
15
-1

Process finished with exit code 0
```

Na **linha 5**, o método **length()** retorna o **tamanho da string**, ou seja a **quantidade de caracteres** da string.

Na **linha 7**, o método **toUpperCase()** retorna os caracteres **maiúsculos da string**.

Na **linha 9**, o método **toLowerCase()** retorna os caracteres **minúsculos da string**.

Nas **linhas 11 e 12**, o método **indexOf()** retorna a **posição da primeira ocorrência de determinado texto na string**. Na **linha 14**, o método retorna **-1**, quando **não encontra o padrão na string**. Observe que esse método **diferencia maiúsculas de minúsculas (Case sensitive)**.

Mais métodos para trabalhar com strings

O método **charAt()** retorna o caractere em um índice específico da string.

O método **compareTo()** compara **duas strings** e retorna:

- 0 se as duas strings forem iguais
- < 0 se a primeira string é menor que a segunda string.
- > 0 se a primeira string é maior que a segunda string

Sintaxe:

```
Primeira_string.compareTo(segunda_string);
```

Os métodos **startsWith()** e **endsWith()** verifica se a string **começa** ou **termina com um valor textual**, respectivamente. Esses métodos retornam **true** ou **false**.



Vamos praticar!

Vamos fazer um programa para trabalhar com os métodos mencionados anteriormente:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_29_Strings**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Strings** e insira o código abaixo.

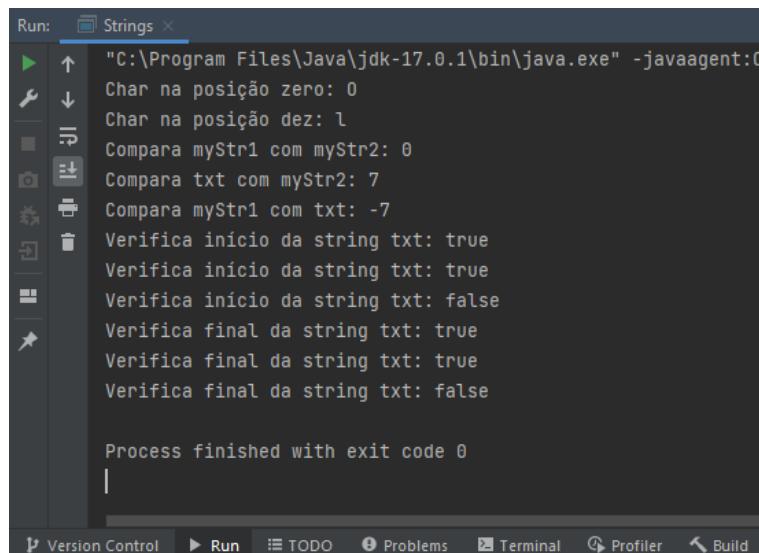
```
1. public class Strings {  
2.     public static void main(String[] args) {  
3.         String txt = "Olá Pessoal do IOS";  
4.         String myStr1 = "Hello";
```

```

5.         String myStr2 = "Hello";
6.
7.         char result = txt.charAt(0); // Caractere na posição zero
8.         System.out.println("Char na posição zero: " + result);
9.         result = txt.charAt(10); // Caractere na posição dez
10.        System.out.println("Char na posição dez: " + result);
11.
12.        // Compara strings
13.        System.out.println("Compara myStr1 com myStr2: " + myStr1.compareTo(myStr2)); // Retorna 0
14.        System.out.println("Compara txt com myStr2: " + txt.compareTo(myStr2)); // Retorna um valor > 0
15.        System.out.println("Compara myStr1 com txt: " + myStr1.compareTo(txt)); // Retorna um valor < 0
16.
17.        // Verifica o ínicio e o final de um string
18.        System.out.println("Verifica início da string txt: " + txt.startsWith("0la")); // Retorna true
19.        System.out.println("Verifica início da string txt: " + txt.startsWith("0 ")); // Retorna true
20.        System.out.println("Verifica início da string txt: " + txt.startsWith("o ")); // Retorna false
21.
22.        System.out.println("Verifica final da string txt: " + txt.endsWith("IOS")); // Retorna true
23.        System.out.println("Verifica final da string txt: " + txt.endsWith("S")); // Retorna true
24.        System.out.println("Verifica final da string txt: " + txt.endsWith("s")); // Retorna false
25.    }
26. }
```

Código do programa utilizando métodos com strings.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.



```

Run: Strings ×
" C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\Users\...\.m2\repository\org\jboss\tools\jdt\jdt-dt\1.0.0.Final\jdt-dt-1.0.0.Final.jar
Char na posição zero: 0
Char na posição dez: l
Compara myStr1 com myStr2: 0
Compara txt com myStr2: 7
Compara myStr1 com txt: -7
Verifica inicio da string txt: true
Verifica inicio da string txt: true
Verifica inicio da string txt: false
Verifica final da string txt: true
Verifica final da string txt: true
Verifica final da string txt: false

Process finished with exit code 0
|
```

Na **linha 14**, a instrução **retorna um valor maior que zero arbitrário e aleatório**. A única coisa que podemos afirmar com certeza é que é maior que zero.

Na **linha 15**, a instrução **retorna um valor menor que zero arbitrário e aleatório**. A única coisa que podemos afirmar com certeza é que **é menor que zero**.

Mais alguns métodos para trabalhar com strings

O método **isEmpty()** verifica se a **string está vazia** e retorna **true** se a string **estiver vazia** e **false** caso **contrário**.

O método **lastIndexOf()**, que retorna o **índice da última ocorrência de um texto específico na string (incluindo espaço em branco)**. Se o método **não encontrar o texto**, ele retornará **-1**.

O método **replace()** retorna uma **nova string substituindo um valor desejado pelo outro**.

O método **trim()** **retira os espaços em branco no início e final** de uma **string**.



Vamos praticar!

Vamos fazer um programa para trabalhar com os métodos mencionados anteriormente:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

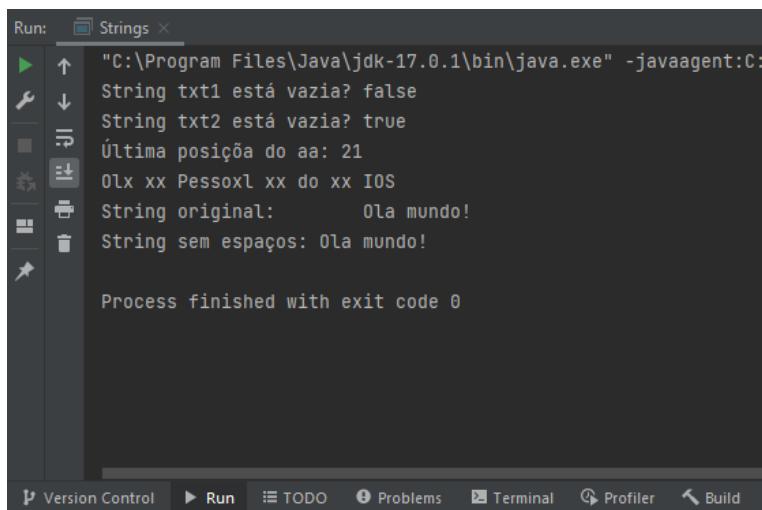
Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_30_Strings**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Strings** e insira o código abaixo:

```
1. public class Strings {  
2.     public static void main(String[] args) {  
3.         String txt1 = "Ola aa Pessoal aa do aa IOS";  
4.         String txt2 = "";  
5.         String txt3 = "      Ola mundo!      ";  
6.  
7.         // Verifica se a string está vazia  
8.         System.out.println("String txt1 está vazia? " + txt1.isEmpty());  
9.         System.out.println("String txt2 está vazia? " + txt2.isEmpty());  
10.  
11.        System.out.println("Última posição do aa: " + txt1.lastIndexOf("aa"));  
12.  
13.        // Substituir algo na string  
14.        String txt4 = txt1.replace('a', 'x');  
15.        System.out.println(txt4);  
16.  
17.        // Retirar espaços no início e no final  
18.        System.out.println("String original: " + txt3);  
19.        System.out.println("String sem espaços: " + txt3.trim());  
20.    }
```

Código do programa utilizando método para manipular strings.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.



The screenshot shows a Java IDE's run console. The title bar says "Run: Strings". The output window displays the following text:

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...\StringManipulation.jar
String txt1 está vazia? false
String txt2 está vazia? true
Última posição do aa: 21
Olá xx Pessoal xx do xx IOS
String original: Olá mundo!
String sem espaços: Olá mundo!

Process finished with exit code 0
```

Below the output window, the IDE's navigation bar is visible with tabs for Version Control, Run, TODO, Problems, Terminal, Profiler, and Build.



Matriz

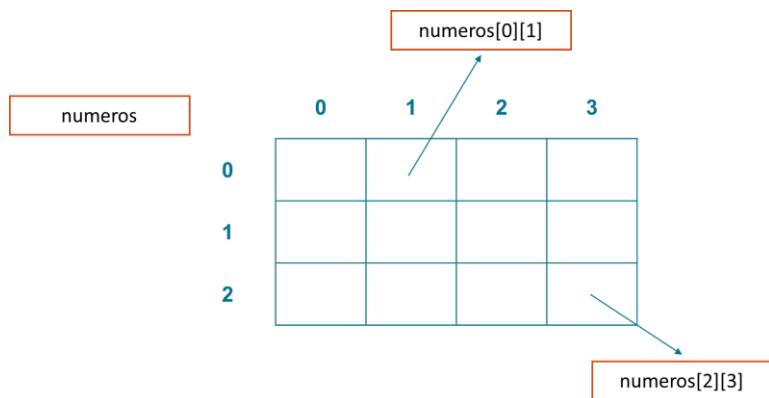
Os objetivos desta aula são:

- Compreender o uso de matrizes;
- Conhecer os métodos utilizados para trabalhar com matrizes.

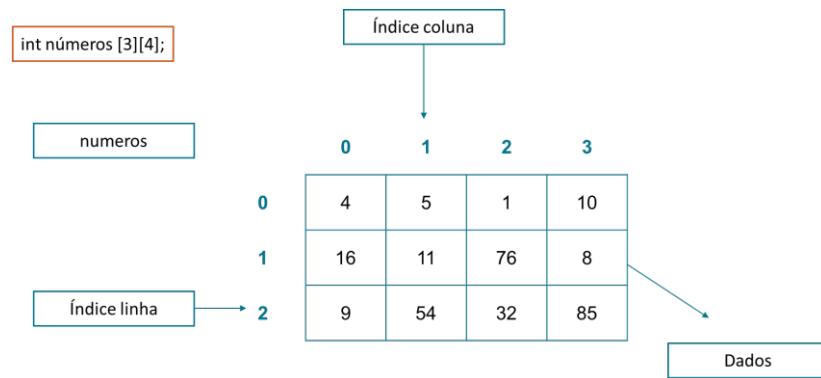
Bons estudos!

Matrizes

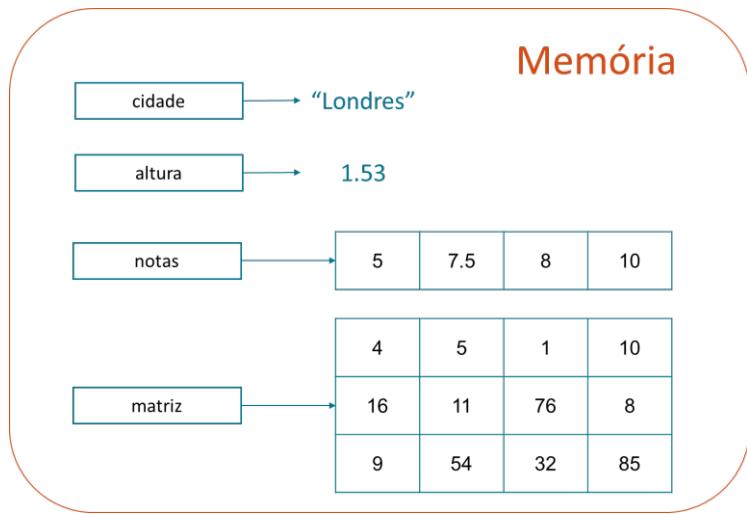
Matriz é uma **variável homogênea multidimensional** e basicamente é um **conjunto de variáveis do mesmo tipo**, que possuem o **mesmo identificador (nome)** e são alocadas **sequencialmente na memória**. Uma matriz precisa de um **índice para cada uma de suas dimensões**.



Por exemplo, a **matriz** **numeros** tem **3 linhas** e **4 colunas**.



Por exemplo, na **memória do computador** podemos ter **diversas variáveis** como mostrado abaixo:



A variável **matriz** possui **3 linhas e 5 colunas**, ela pode armazenar **15 elementos**, sendo que o primeiro elemento está na posição **(0, 0)**. Lembre-se, o **índice de uma matriz deve começar no valor zero**.

Toda matriz deve ser representada por um **tipo de dado**, que será **armazenado na matriz**, um **indicador**, que o **nome da matriz**, e o **tamanho**, que a **dimensão (linha e coluna)** da matriz.

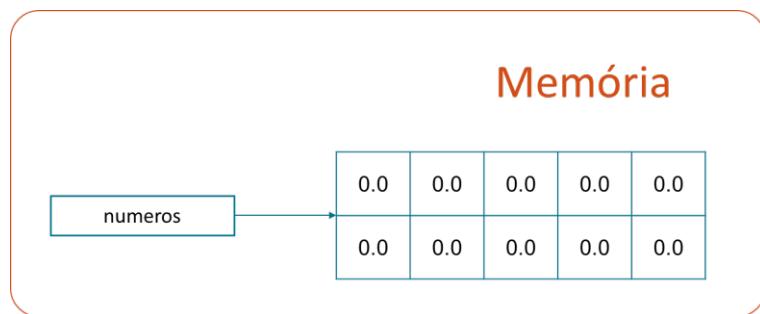
A sintaxe de uma matriz é:

```
tipo identificador [] [] = new tipo[tamLin][tamCol];
```

Por exemplo:

```
double numeros [] [] = new double[2][5];
```

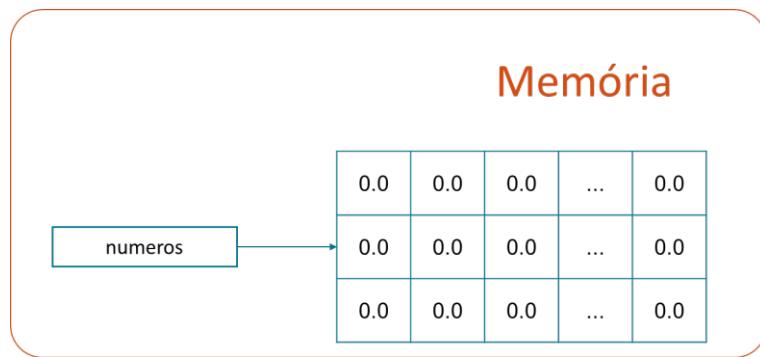
A variável **numeros** é do tipo **double** e possui **duas linhas e cinco colunas** e, com essa declaração o computador separa **dez posições de memória sequenciais para armazenar os seus elementos**:



Vejamos outro exemplo:

```
int numeros [] [] = new int[3][100];
```

Nesse caso, o computador separa 300 posições de memória para armazenar valores inteiros dentro da matriz **numeros**.

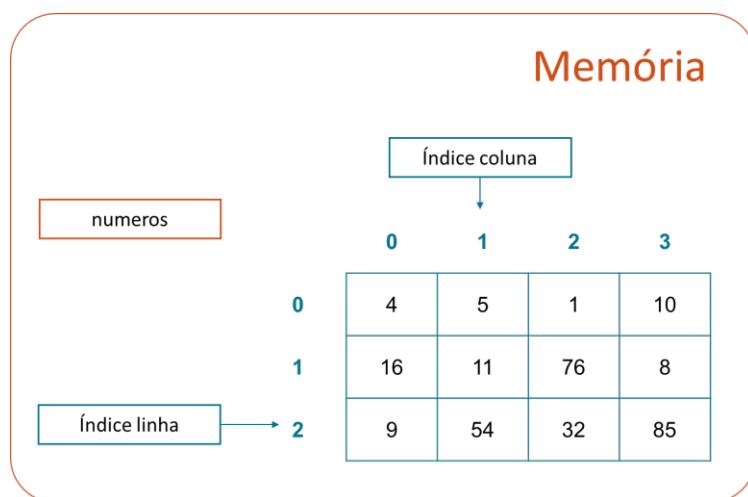


Nós podemos também criar **matrizes** a partir de uma **lista de valores entre chaves { }** e **separados por vírgula.**

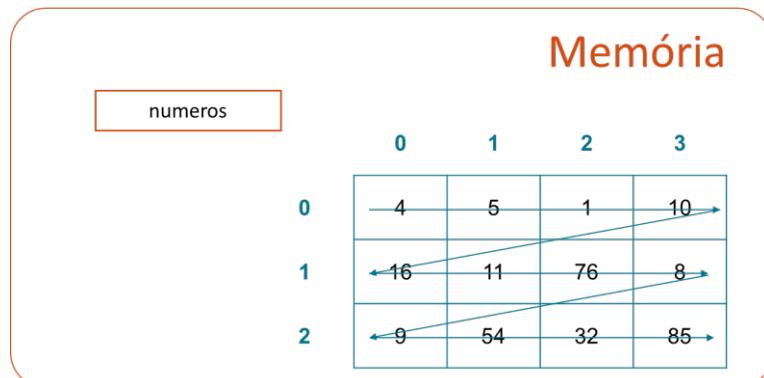
```
int[][] numeros = {{4, 5, 1, 10}, {16, 11, 76, 8}, {9, 54, 32, 86}};  
int[][] n2 = {{2, 2}, {3, 4}};  
int n3[][] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
```

Nesse caso, a vírgula separa os **elementos** em **colunas** e cada **grupo de chaves { }** é uma **nova linha**. Então, a variável **numeros** tem **3 linhas e 4 colunas**, a variável **n2** tem **2 linhas e 2 colunas** e a variável **n3** tem **3 linhas e 3 colunas**.

Nós acessamos os elementos da **matriz** pelo **índice**, que **indica a posição do dado nela**. Sempre quando estamos trabalhando com **matriz bidimensional** usaremos a **linha** e a **coluna** para identificar **cada posição de um elemento**.



A ideia básica, para **percorrer os elementos de uma matriz**, é **fixar a primeira linha** e **percorre todas as colunas**, depois **fixar na segunda linha** e **percorre todas as colunas**. Dessa forma, conseguimos **percorrer todos os elementos da matriz**.



Geralmente, para percorrer uma **matriz** utilizamos **dois laços de repetição**, o **primeiro laço percorre a linha** e o **segundo laço percorre a coluna**.

```
for (i = 0; <=2; i++) {//Percorre linhas
    for (j = 0; j<=3; j++) {//Percorre colunas
        System.out.println("Digite um número: ");
        numeros [i][j] = entrada.nextInt();
    }
}
```

No **primeiro laço for**, a variável **i** inicia em **0 (que é o índice da primeira linha)**, então a **próxima linha que será executada** é o segundo **laço for**. Nesse segundo laço, a variável **j** inicia em **zero (que a primeira coluna)** e vai até **3 (que é a última coluna)**. Então, a variável **i** é **incrementada para 1**, o **segundo laço reinicia novamente** e a variável **j** inicia em **zero (que a primeira coluna)** e vai até **3 (que é a última coluna)**.

Por fim, a variável **i assume** o valor **2 (que é a última linha)** e a variável **j** inicia em **zero (que a primeira coluna)** e vai até **3 (que é a última coluna)**. Assim, percorrendo **todos os elementos da matriz numeros**.

Precisamos **percorrer** os elementos da **matriz** para **preenchê-la** e para **ler** os elementos dela caso quisermos **imprimi-los**, por exemplo.

```
for (i = 0; <=3; i++) {//Percorre linhas
    for (j = 0; j<4; j++) {//Percorre colunas
        System.out.println(numero[i][j] + "\t");
    }
    System.out.println(" ");
}
```

Propriedade length usada em uma matriz

Vamos ver o seguinte exemplo:

```
int numeros[][] = {{4, 5, 1, 10}, {16, 11, 76, 8}, {9, 54, 32, 89}};
int i, j;
for (i = 0; i < numeros.length; i++) {
    for (j = 0; j < numeros[i].length; j++) {
        System.out.println(numeros[i][j] + "\t");
    }
    System.out.println(" ");
}
```

Nesse exemplo, a instrução **numeros.length** retorna a **quantidade de linhas**, que são **4** e a instrução **numeros[i].length** retorna a **quantidade de colunas para cada linha**.



Vamos praticar!

Siga os passos para escrever o programa utilizando operadores relacionais:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_31_Matriz**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Exemplo1Matriz** e insira o código abaixo.

```

1. public class Exemplo1Matriz {
2.     public static void main(String[] args) {
3.         // Declarar e inicializar a matriz
4.         int numeros[][] = {{4, 5, 1, 10},{16, 11, 76, 8},{9, 54, 32, 89}};
5.         int i, j;
6.         for(i = 0; i < numeros.length; i++){
7.             for(j = 0; j < numeros.length; j++){
8.                 System.out.print(numeros[i][j] + "\t");
9.             }
10.            System.out.println("");
11.        }
12.    }
13. }
```

Código do programa utilizando matriz.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

Ao executar o código, o programa imprimir os **valores da matriz** em suas respectivas **linhas e colunas**.

```

Run: Exemplo1Matriz x
C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:
4 5 1
16 11 76
9 54 32
Process finished with exit code 0
```

Na **linha 6**, o comando **for** é usado para **percorrer as linhas da matriz** e na **linha 7** o segundo **for** é usado para **percorrer as colunas da matriz**.

Observe, que o comando **System.out.print** na **linha 8** possui um **código de sequência de escape \t**, que salta um **espaço de tabulação separando os elementos da matriz**. E na linha 10, o comando **System.out.println** tem uma **string vazia**, pois esse comando provoca uma quebra de linha para **separar as linhas da matriz**.



Vamos praticar!

Siga os passos para escrever o programa utilizando operadores relacionais:

Crie um novo projeto no **IntelliJ IDEA**. Para isso, acesse o Menu **File**, a opção **New** e depois clique na opção **Project**.

Não se esqueça de dar um nome representativo para o projeto, por exemplo: **Projeto_32_Matriz**

No diretório **src**, crie um **arquivo Java Class**. Não se esqueça de dar um nome **representativo para o projeto**, por exemplo: **Exemplo2Matriz** e insira o código abaixo.

O resultado do código pode ser visto ao **compilar e executar o código**, clicando com o **botão direito na classe** e selecionando a opção **Run**.

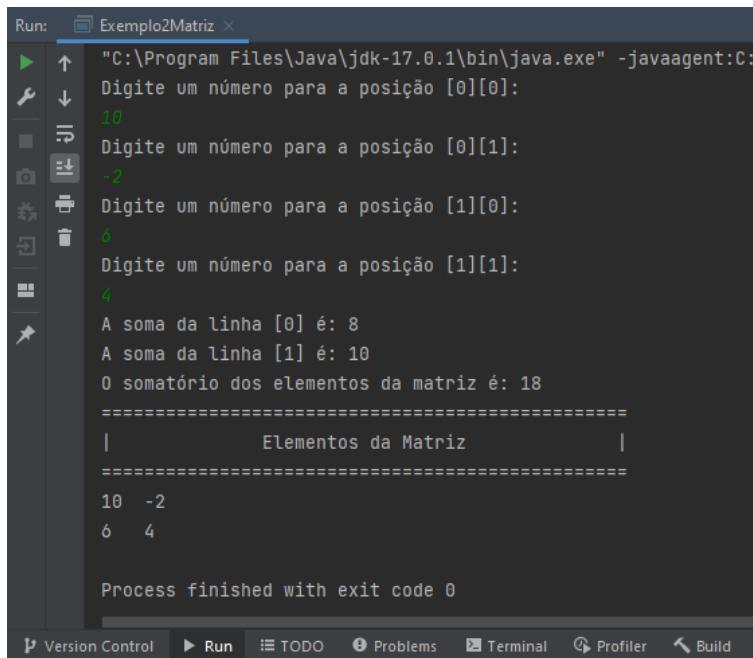
```
1. import java.util.Scanner;
2.
3. public class Exemplo2Matriz {
4.     public static void main(String[] args) {
5.
6.         Scanner teclado = new Scanner(System.in);
7.
8.         // Declaração da matriz
9.         int[][] numeros = new int[2][2];
10.        int somaLinhas = 0, total = 0;
11.
12.        // Armazenar os dados
13.        for (int i = 0; i < numeros.length; i++){
14.            for (int j = 0; j < numeros[i].length; j++){
15.                System.out.println("Digite um número para a posição [" + i + "]["
16.                    + j + "]: ");
17.                numeros[i][j] = teclado.nextInt();
18.            }
19.
20.            // Precorrer os dados para calcular o somatório
21.            for (int i = 0; i < numeros.length; i++){
22.                somaLinhas = 0; // zera o acumulador de linhas toda vez que inicia u
ma nova linha
23.                for (int j = 0; j < numeros[i].length; j++){
```

```

24.                 somaLinhas += numeros[i][j]; // acumula os somatório dos elementos de cada linha
25.             }
26.             System.out.println("A soma da linha [" + i + "] é: " + somaLinhas);
27.             total += somaLinhas; // acumula o somatório total dos elementos da matriz
28.         }
29.
30.         System.out.println("O somatório dos elementos da matriz é: " + total);
31.
32.         // Precorrer a matrix para imprimir os dados
33.         System.out.println("=====");
34.         System.out.println(" | Elementos da Matriz | ");
35.         System.out.println("=====");
36.         for (int i = 0; i < numeros.length; i++){
37.             for (int j = 0; j < numeros[i].length; j++){
38.                 System.out.print(numeros[i][j] + "\t");
39.             }
40.             System.out.println("");
41.         }
42.         teclado.close();
43.     }
44. }
```

Código do programa utilizando matriz.

Ao executar o programa, ele irá solicitar **cinco valores inteiros**, depois irá procurar o maior valor digitado e imprimi-lo na tela.



The screenshot shows a Java IDE interface with a terminal window titled "Exemplo2Matriz". The terminal displays the following interaction:

```

Run: Exemplo2Matriz ×
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" -javaagent:C:\...
Digite um número para a posição [0][0]:
10
Digite um número para a posição [0][1]:
-2
Digite um número para a posição [1][0]:
6
Digite um número para a posição [1][1]:
4
A soma da linha [0] é: 8
A soma da linha [1] é: 10
O somatório dos elementos da matriz é: 18
=====
| Elementos da Matriz |
=====
10  -2
6   4

Process finished with exit code 0
```

The bottom navigation bar includes icons for Version Control, Run, TODO, Problems, Terminal, Profiler, and Build.

Nesse exemplo, utilizamos **três laços de repetição**: um para **preencher a matriz com os valores digitados pelo usuário**, outro para **realizar o somatório das linhas e de todos os elementos da matriz** e o terceiro para **imprimir os valores do vetor**.