Linguagem de Programação Java - Básico

Sumário

Sumário

Apresentação

Bem-vindo(a) ao curso **Linguagem de Programação Java - Básico!**

O objetivo deste curso é apresentar os principais conceitos sobre a linguagem de programação Java com uma breve introdução, percorrendo os fundamentos dessa linguagem, os fluxos de controle utilizados e, por fim, o uso de matrizes.

Desejamos a você um excelente aprendizado!

Vídeo

Confira o vídeo de apresentação do curso.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Seja bem-vindo(a) ao curso Linguagem de Programação Java Básico.

Neste curso você verá uma introdução à linguagem Java e conhecerá conceitos importantes para compreender o Java *Development Kit* (JDK).

Com esses conhecimentos será possível entender sua história, o que é portabilidade em Java, o que é *bytecode*, como instalar o JDK, do que se trata o ambiente de desenvolvimento integrado, como criar o primeiro programa em Java, qual é a estrutura dessa linguagem, entre outras particularidades iniciais.

Além disso, serão apresentadas as palavras reservadas na linguagem e seus operadores.

Você também conhecerá os conceitos de fluxo de controle e *arrays*, podendo colocar em prática esse conjunto indexado de informações.

Vamos começar esta jornada? Acompanhe.

Módulo 1

Introdução ao Java

fundação bradesco | escola virtual

Introdução ao Java

Neste primeiro módulo, você conhecerá um breve histórico referente à linguagem de programação Java, incluindo dados sobre o seu surgimento e alguns conceitos importantes para que você entenda o Java *Development Kit* (JDK).

Diante disso, você irá aprofundar-se um pouco mais no assunto e aprender como instalar e utilizar o JDK. Veja o que preparamos!

O que é Java?

A maioria das pessoas já se familiarizou com as tarefas rotineiras que os computadores realizam, porém, o que pouca gente sabe é que uma linguagem de programação como o Java nos permite escrever instruções que serão executadas nesses computadores. Tais instruções construídas em Java são chamadas de **softwares**, também conhecidos como "programas de computador". Sobre isso, um exemplo de programa de computador são os softwares de webconferência.

#PraCegoVer: na imagem, há uma empresária de costas, enquanto faz uma webconferência com treze colegas em seu notebook, o qual está em cima de uma mesa de escritório. Ao lado, existe uma pasta e um smartphone.

Aliás, os profissionais que constroem os programas de computador ou softwares

e utilizam uma linguagem de programação como o Java, são chamados de programadores ou desenvolvedores. No entanto, para que compreenda melhor o motivo por trás dessas nomenclaturas, conheça um pouco da história do Java, como ele foi criado e de que maneira foi desenvolvido!

Um pouco de história

Anteriormente, você viu os primeiros conceitos atrelados ao Java, o de software e desenvolvedores (ou programadores). Porém, neste momento, saberemos brevemente um pouco sobre a história dessa linguagem tão empregada atualmente no mundo da computação.

O Java foi criado pela empresa Sun Microsystems, no ano de 1991, quando um projeto de pesquisa corporativa resultou em uma linguagem de programação. No entanto, para entender melhor a história da linguagem Java, convidamos você a assistir o vídeo a seguir, que detalha sua evolução ao longo dos anos. Acompanhe!

Vídeo

Confira o **vídeo** a evolução da linguagem Java.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Antes de falar sobre a linguagem Java, é importante entendermos como ela se originou e evoluiu ao longo dos anos.

O criador, James Gosling, inicialmente, chamou a linguagem de OAK, em homenagem a uma árvore de carvalho que observava de sua janela na organização. Mais tarde, porém, descobriu que já existia uma linguagem de computador com esse nome.

Em 1991, a equipe da Sun, ao visitar uma cafeteria local no Vale do Silício, resolveu adotar o nome Java, que consiste no nome da Ilha de Java, adotado para designar uma linguagem de computador.

Um ano depois, em 1992, foi criado o mascote da linguagem Java, chamado Duke, baseado no emblema da série de televisão intitulado *Star Trek*, a qual James Gosling era um apreciador dos episódios.

Em 1995, a linguagem Java foi oficialmente lançada e, em 1996, ocorreu o lançamento do Java *Development Kit* (JDK). Este kit de desenvolvimento Java consiste no pacote que traz os elementos necessários para a construção dos programas em Java.

O lançamento do JDK foi realizado no Java *One*, a primeira conferência de desenvolvedores da linguagem de programação Java. E nesse momento foi divulgado o famoso logotipo, que logo após o lançamento da linguagem, contou com a identidade visual representada por uma xícara de café na cor azul, com um vapor de cor vermelha. Ele faz referência aos criadores da linguagem, uma vez que, na época de desenvolvimento, tomaram muito café!

Em 1998, a empresa Visa teve como base o Java em todos os seus cartões de crédito. Com essa adoção da linguagem pela Visa, houve a possibilidade de sua expansão para outras empresas.

Assim, no ano de 2002, 78% dos executivos já preferiam a linguagem

Java para a criação de *webservices*, que se constitui como uma tecnologia utilizada em ambientes corporativos de grandes empresas.

Em 2004, dois anos mais tarde, a NASA enviou para Marte um robô controlado por Java na missão *Spirit*, sendo que muitas pessoas provavelmente o assistiram pela internet ou em noticiários na televisão. E, em 2009, a empresa Oracle comprou a Sun Microsystems, garantindo a continuidade na linguagem Java até os dias de hoje.

Agora que você já conhece a origem da linguagem Java e sua evolução é possível perceber como ela impactou a programação, não é verdade? Confira na sequência mais detalhes sobre essa evolução.

O Java, enquanto ia avançando na programação, influenciado pela internet, igualmente a impactou. Além de simplificar a programação convencional na web, inovou por meio de um programa de rede chamado *applet*, o qual mudou a visão do conteúdo no mundo on-line. Isso porque o Java resolve alguns dos problemas complexos relacionados a internet, como portabilidade e segurança.

Dito isso, você pode estar se perguntando: por que a linguagem Java consegue resolver o problema da portabilidade e da segurança?

Basicamente, pode-se dizer que a **portabilidade** diz respeito a capacidade dessa linguagem usar o mesmo código em sistemas operacionais diferentes. Isso evita, por exemplo, que outros executáveis sejam baixados para que determinados programas possam funcionar, contribuindo, assim, para a **segurança** do sistema. Para saber mais sobre isso, acompanhe o tópico a seguir.

Portabilidade

No *podcast* a seguir, você ouvirá sobre o que é a portabilidade e como ela se aplica às linguagens de programação, como é o caso do Java. Confira!

Podcast

Confira o ***podcast*** sobre portabiliade.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Vamos falar sobre portabilidade e sua aplicação nas linguagens de programação?

A portabilidade consiste em um importante aspecto ligado à internet, visto que, nela, estão conectados diferentes tipos de computadores e sistemas operacionais.

Dentro desse contexto, podemos dizer que a portabilidade é quando um programa escrito para um tipo de sistema operacional pode ser compilado ou executado em diferentes arquiteturas.

Desse modo, para cada computador que for executar um programa Java, desde que esteja conectado à Internet, existe a possibilidade de o programa ser processado em um sistema diferente. Por exemplo, no caso de pequenos aplicativos, o programa deve ser capaz de baixá-los e executá-los por meio de várias CPUs, diversos sistemas operacionais e inúmeros navegadores conectados à internet.

Sendo assim, é desaconselhável utilizar versões diferentes de miniaplicativos para computadores diferentes, pois o mesmo código deverá ser válido em todos os computadores.

Portanto, alguns métodos de geração de código executável portátil são necessários. Dessa maneira, os mesmos mecanismos que ajudam a manter a segurança, também contribuem para a portabilidade!

Viu só como a portabilidade carrega uma importante função? Ela proporciona segurança aos softwares. Continue seus estudos e descubra como o *bytecode* é essencial para esse processo.

Aqui você pôde conhecer mais sobre a característica da portabilidade da linguagem Java, a qual, por sua vez, proporciona segurança aos softwares. Todavia, para que isso seja viabilizado, é necessário um outro elemento fundamental, o *bytecode*. Vamos conhecer melhor sobre ele? Siga adiante.

Bytecode

O segredo para permitir que o Java resolva os problemas de segurança e portabilidade que acabamos de aprender, decorrentes da saída do compilador Java, que não é um código executável, seria utilizar o *bytecode*. Ouça o *podcast* na sequência para saber mais a respeito do assunto!

Podcast

Confira o ***podcast*** sobre *bytecode*.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! Agora que você já sabe como a portabilidade é importante para a segurança dos softwares, confira como o *bytecode* é fundamental para que essa segurança seja viável.

O *bytecode* diz respeito a um conjunto de instruções altamente otimizadas, projetadas para serem executadas pelo sistema de tempo de execução Java, chamado Java *Virtual Machine* (JVM).

Originalmente o JVM foi projetado como um interpretador de *bytecode*. Parece algo novo, uma vez que muitas linguagens modernas são criadas para serem compiladas em código executável por motivos de desempenho. No entanto, o fato de os programas Java serem executados pelo JVM ajuda a resolver os principais problemas associados aos programas baseados na web.

A conversão de programas Java em *bytecode* facilita a execução de programas em vários ambientes, devido ao fato de que cada plataforma precisa apenas implementar o JVM. Assim, o pacote *runtime* aparece em determinado sistema, e qualquer programa Java pode ser executado.

Aqui, ainda vale mencionar que este pacote Java *Runtime Environment* (JRE) é de aplicativos, composto por bibliotecas (APIs) e pelo Java *Virtual Machine* (JVM). O kit contém as ferramentas necessárias para a execução de aplicativos e pode ser baixado diretamente pelo site da Oracle.

Embora os detalhes do JVM variem conforme as plataformas, todos entendem o mesmo *bytecode* Java. Portanto, se um programa Java for compilado para código nativo, precisam existir versões diferentes do mesmo programa para cada tipo de CPU conectado à internet.

Com isso, a execução do *bytecode* pela JVM é a maneira mais fácil de criar programas verdadeiramente portáveis. Além do fato de o JVM tornar os programas executados por ele mais seguros. Pois, por ser controlado, ele pode reter o programa e evitar que produza efeitos colaterais fora do sistema. Desse modo, certas restrições na linguagem Java também

aumentam a segurança.

Um programa, ao ser interpretado, geralmente será executado lentamente quando comparado a outro que está em execução, ou seja, um compilado em código executável. No entanto, em Java, a diferença entre os dois não é considerável. Como o *bytecode* é altamente otimizado, seu uso permite que o JVM execute programas mais rápidos que o esperado.

Viu só como a portabilidade, resultado da relação entre o *bytecode* e o JVM, promove a segurança dos softwares também? Fique ligado para descobrir mais sobre o Java!

Neste *podcast* você conheceu não só o conceito de *bytecode*, mas também a sua relação com o JVM (Java *Virtual Machine*), o qual contém o conjunto de instruções otimizáveis que propicia a portabilidade e, por consequência, a segurança aos softwares.

E por falar em portabilidade e segurança, como mencionado no *podcast*, para a construção dos programas em Java precisamos ter em mãos um kit instalado no computador. Por isso, no tópico a seguir, você saberá como baixá-lo e cadastrar o *login* desse importante recurso que permite a criação da linguagem Java.

Java *Development Kit* (JDK)

Para utilizarmos a plataforma Java, precisamos ter o Java *Development Kit* (JDK) instalado no computador. Ele pode ser baixado diretamente no site da Oracle. A versão utilizada é a Java SE8 (*Standard Edition*).

Será importante observar que o pacote de instalação JDK oferece duas principais instalações, conforme você conhecerá a seguir!

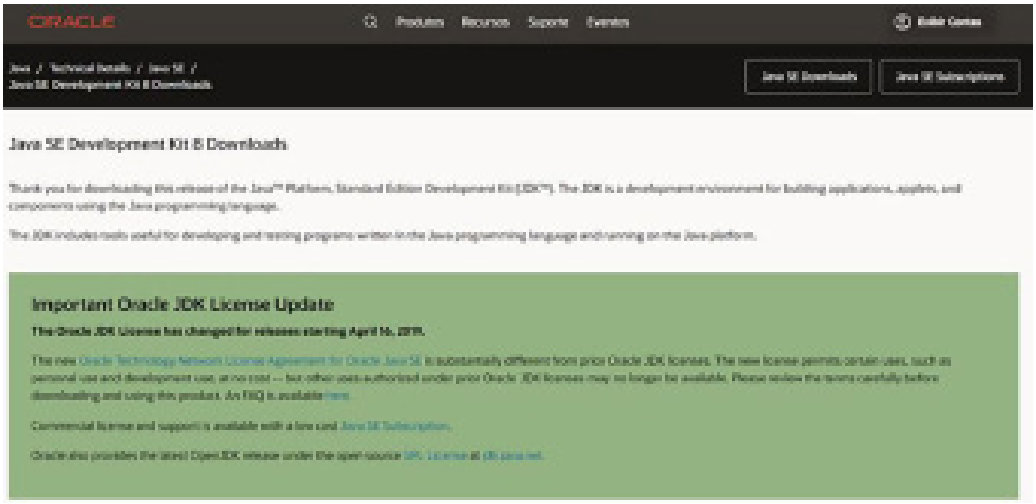
Instalação 1

Compilador Java (javac).

Instalação 2

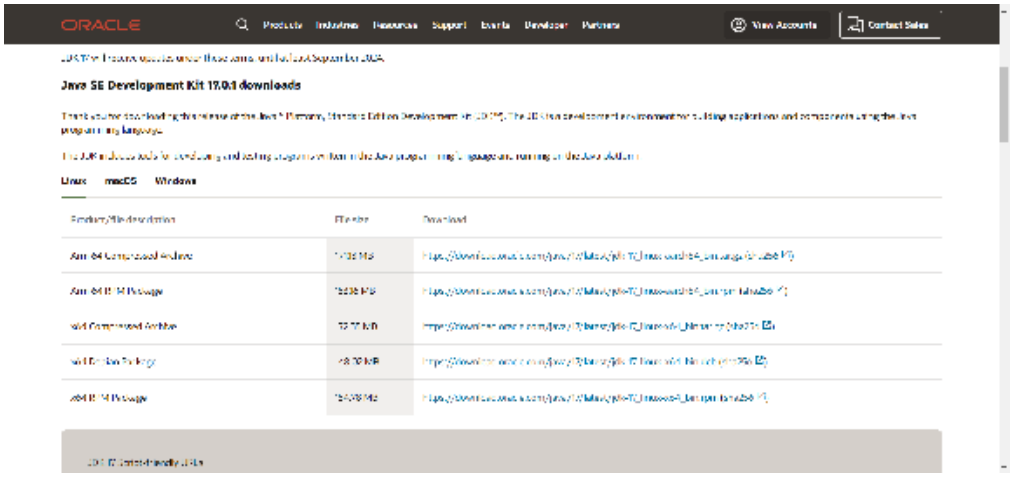
Interpretador padrão utilizado na inicialização dos aplicativos (Java).

O processo para a instalação é simples, sendo que, uma vez instalado, é possível compilar e executar os programas. Logo, no recurso adiante, você poderá acompanhar o passo a passo de como instalar o Java *Development Kit* (JDK) no seu computador.



#PraCegover: No *print* de tela, é possível ver a página principal do JDK, sendo que a barra de ferramenta com os seus botões principais está no topo e, abaixo dele, existem dois textos escritos contendo informações sobre a atualização do licenciamento do Oracle para o JDK.

Ao acessarmos a página de download do Java, é preciso observar a nota que informa sobre a atualização do licenciamento Oracle JDK, visto que ele sofreu alteração a partir de 16 de abril de 2019.



#PraCegoVer: no *print* de tela, é possível ver uma janela de aviso do Orcacle JDK, indicando a versão do sistema operacional em que o programa de criação de códigos do Java será instalado. Atrás da janela, existe a interface do programa e, no topo da tela, existe a barra de ferramentas com os seus botões principais.

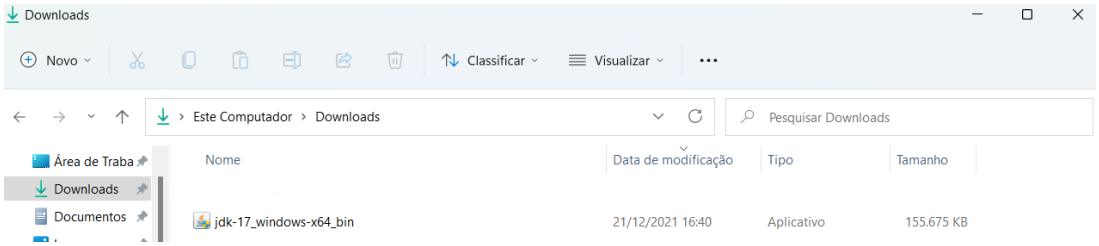
A versão que utilizaremos no curso será a “Windows X64 - jdk-8u281-windows-x64.exe”. Na imagem, é possível observar essa etapa!

Ao selecionar download, será solicitado o *login* e a senha de acesso. Caso não tenha, é preciso criar uma conta (gratuitamente) no site da Oracle. Para isso, basta selecionar “criar conta” e preencher o formulário. Veja a seguir!

#PraCegoVer: no *print* de tela, há a página de *login* da conta do Oracle, contendo a janela para digitar o *login* e senha. Ao fundo, existe o papel de parede do site.

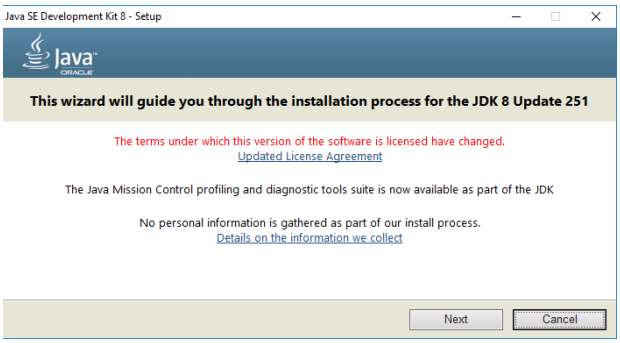
#PraCegoVer: no *print* de tela, há a página de criação da conta no site do Oracle, contendo a janela para digitar os dados de cadastro. Ao fundo, existe o papel de parede do site.

Em seguida, utilize novamente a URL para download do Java e insira as informações de acesso, a fim de iniciar o processo do pacote de instalação.

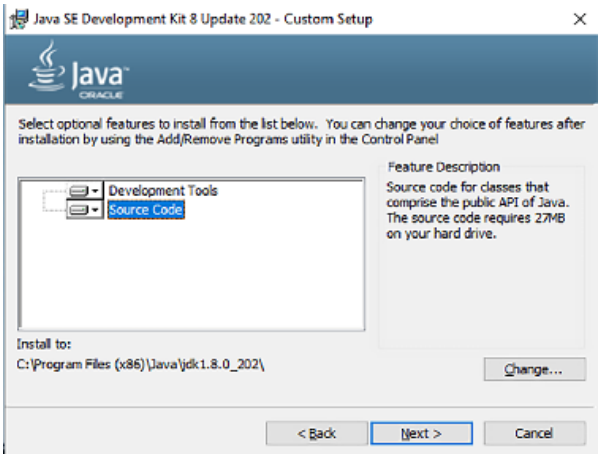


#PraCegoVer: na imagem, temos um *print* de repositório do computador demonstrando o caminho de instalação do pacote de Java.

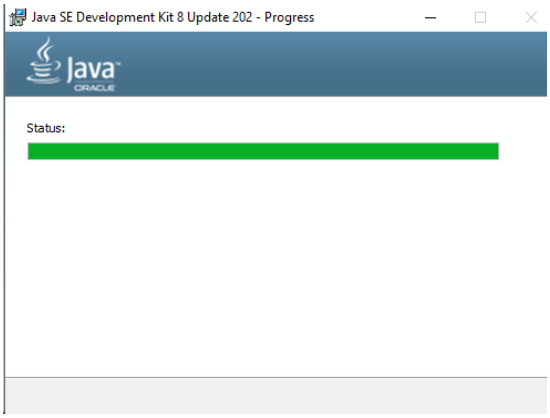
Após realizar o download, o processo seguinte é bem simples, podendo ser realizado conforme apresentado na imagem de passo a passo abaixo. As etapas consistem no famoso "*next*", "*next*" e "*finish*".



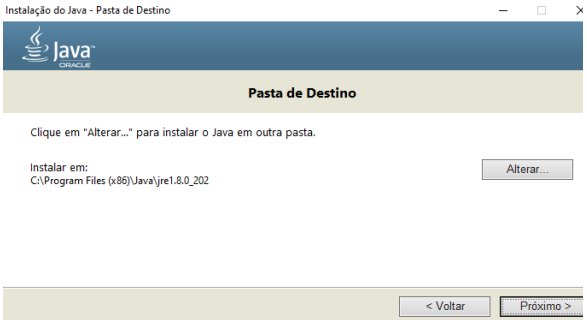
#PraCegoVer: na imagem, há uma captura de tela referente à janela inicial de instalação do Java. Nela, há o termo de aceite para a instalação.



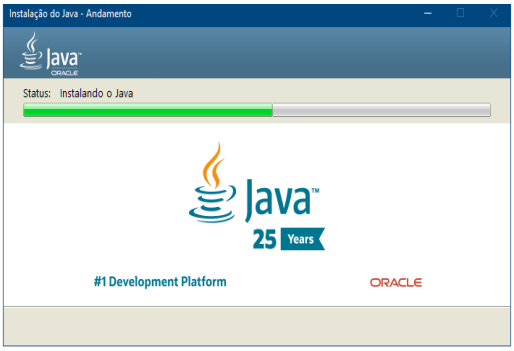
#PraCegoVer: na imagem, há uma captura de tela referente à janela inicial de instalação do Java. Nela, há a opção de onde instalar o programa no disco rígido.



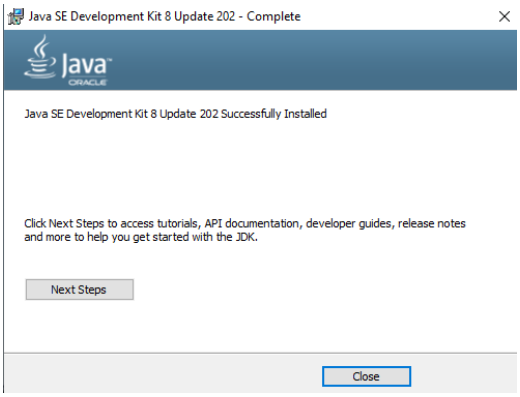
#PraCegoVer: na imagem, há uma captura de tela referente à janela inicial de instalação do Java. Nela, há uma barra de progresso da instalação.



#PraCegoVer: na imagem, há uma captura de tela referente à janela inicial de instalação do Java. Nela, há um aviso de onde ficará a pasta de destino ligada ao programa.



#PraCegoVer: na imagem, há uma captura de tela referente à janela inicial de instalação do Java. Nela, há uma barra de progresso da instalação com a logo do Java no centro.



#PraCegoVer: na imagem, há uma captura de tela referente à janela inicial de instalação do Java. Nela, há um aviso dizendo que a instalação foi finalizada

Uma vez finalizada a instalação, abra o *prompt* de comando e digite "*java-version*" para certificar-se de que o Java foi instalado.

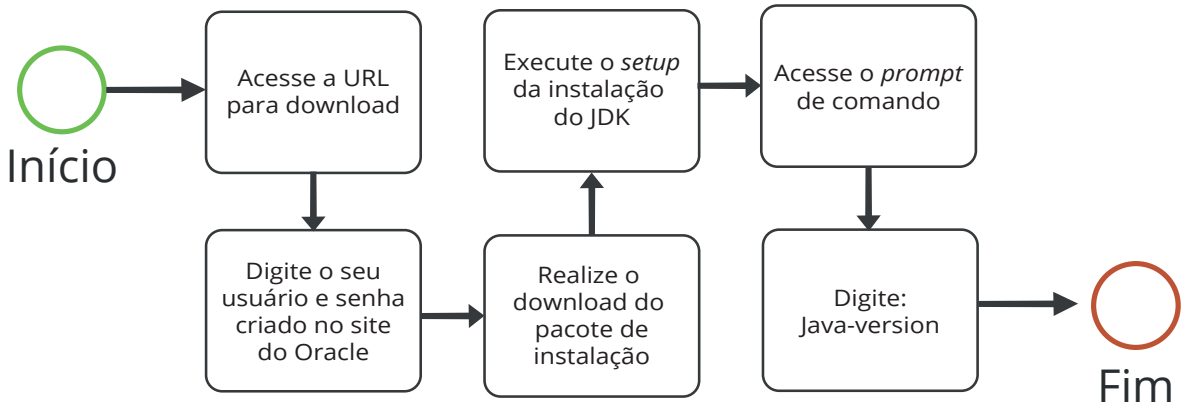
```
Cs. Prompt de Comando

C:\exercicio>java -version
java version "1.8.0_281"
Java(TM) SE Runtime Environment (build 1.8.0_281-b09),
Java HotSpot(TM) 64-Bit Server VM (build 25.281-b09, mixed
mode)

C:\exercicio>
```

Em resumo, na linha de ação abaixo, você pode conferir o processo completo de

instalação do Java:



#PraCegoVer: na imagem, temos um fluxograma do processo para download do pacote de instalação Java. Nele, existem oito etapas, sendo que cada uma é representada por uma figura geométrica. A primeira é um círculo contendo a palavra "início" escrita, depois, há um quadrado contendo a informação escrita "acesse a URL para download", seguido de outro contendo a informação escrita "digite o seu usuário e senha criado no site da Oracle", seguido de outro contendo a informação escrita "realize o download do pacote de instalação", seguido de outro contendo a informação escrita " execute o *setup* da instalação do JDK", seguido de outro contendo a informação escrita " Acesse o *prompt* de comando", seguido de outro contendo a informação escrita " digite: Java-Version", seguido de outro contendo a informação escrita "fim".

Neste tópico, você aprendeu como fazer o download e o cadastro para o *login* no Oracle, passos relevantes para utilizar o *Kit* JDK. Assim, após compreender essa primeira parte, ainda será necessário conhecer o *Integrated Development Environment* (IDEs), isto é, o ambiente de desenvolvimento integrado que nada mais são que softwares utilizados pelos programadores. Acompanhe no tópico a seguir.

Ambiente de desenvolvimento integrado ou *Integrated Development Environment* (IDEs)

Anteriormente, você conheceu as etapas para realizar o download e o cadastro para o *login* no Oracle, de modo que o JDK seja utilizado. Todavia, antes que isso seja efetivado, é essencial o conhecimento de um recurso muito eficaz para a programação: os *Integrated Development Environment* (IDEs), ou também conhecidos em português como ambiente de desenvolvimento integrado.

No *podcast*, será possível conhecer em maiores detalhes o que eles são e quais funções podem exercer. Acompanhe com atenção!

Podcast

Confira o ***podcast*** sobre os IDEs.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! Agora que você já aprendeu a baixar o kit JDK, chegou o momento de conhecer os *Integrated Development Environment* ou IDEs.

Os IDEs são softwares utilizados pelos programadores no desenvolvimento das tarefas de forma prática, pois oferecem uma elevada quantidade de recursos, como as sugestões de códigos, *debugs*, entre outras possibilidades.

Portanto, eles são, basicamente, softwares, que podem ser utilizados como ferramentas de apoio na elaboração ao desenvolvimento dos programas, permitindo, assim, maior agilidade.

Alguns exemplos de IDEs que podem ser utilizados são: Eclipse, *NetBeans, IntelliJ IDEA, BlueJ e JCreator.*

Viu só como os IDEs podem auxiliar na hora de programar? Então, ao acionar o JDK não perca tempo e otimize seu trabalho.

Simples não é mesmo? Até aqui você entendeu que os IDEs, no momento em que o JDK é acionado, proporcionam uma série de ações, como identificar *bugs*, sugerir códigos etc., as quais ajudam (e muito) na hora da programação.

Parabéns! Você chegou ao final do módulo 1.

Agora que já conhece um pouco da história da linguagem Java, principalmente, sobre a sua portabilidade e segurança, bem como instalar e se cadastrar no JDK e acerca dos IDEs, no próximo módulo, o foco será nos fundamentos dessa linguagem e suas particularidades, os quais são muito relevantes, quando você estiver programando.

Preparado(a)? Então vamos lá!

Módulo 2

Fundamentos da linguagem Java

fundação bradesco | escola virtual

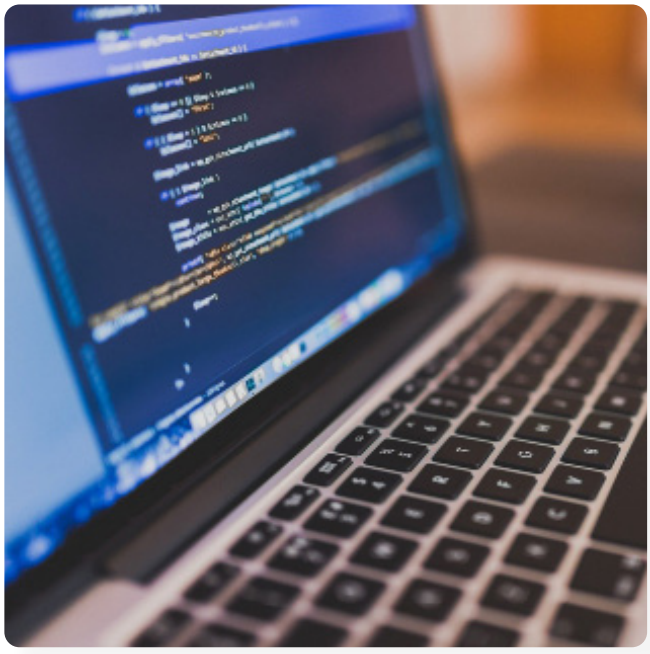
Fundamentos da linguagem Java

Neste módulo, você realizará, na prática, seu primeiro programa em Java. Para isso, verá sobre o funcionamento da estrutura da linguagem Java, como inserir comentários em uma ou mais linhas no código, como criar variáveis, as palavras reservadas em Java e os seus tipos, e como utilizar os operadores. Vamos começar!

Primeiro programa em Java

O primeiro programa em Java será construído da seguinte maneira: inicialmente, criaremos um arquivo e o salvaremos com a extensão ".java", nele serão escritos os comandos na linguagem Java. Depois, compilaremos esse arquivo e será criado outro com a extensão "*.class*", com o mesmo nome do programa que criamos. Esse arquivo "*.class*" é o programa Java. Os dados desse arquivo gerado são chamados de *bytecodes*.

Esse programa Java ainda não está no formato binário, ou seja, não tem conteúdo em linguagem de máquina. Os *bytecodes* podem executar em qualquer máquina que tenha o Java instalado.



Java *Virtual Machine*

O Java que executa um programa é conhecido como Java *Virtual Machine* ou JVM, como já conhecemos.

#PraCegoVer

Na imagem, há um enquadramento de um notebook aberto em cima de uma mesa de escritório. Na tela dele, existem várias linhas de código escritas.

O JVM simula o computador, o qual não é físico, sendo apenas construído em um software que realiza a interpretação do programa Java.

#PraCegoVer

Na fotografia, há um programador de costas e com um fone de ouvido enquanto está trabalhando.

Dessa forma, converte o programa em linguagem de máquina para ser executado pelo computador ou qualquer outro dispositivo que suporte o Java.

#PraCegoVer

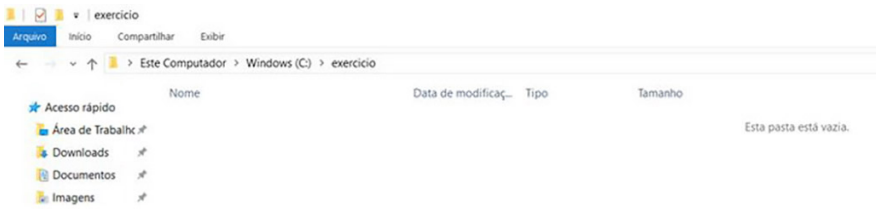
Na fotografia, uma tela de computador está enquadrada e contém várias linhas de código.

Vamos, então, criar o programa?

O primeiro código terá como saída, a exibição da expressão “Olá Mundo!”. Nesse primeiro momento, siga as etapas indicadas, não se preocupando em conhecer os detalhes.

Passo 1

O passo inicial é criar o primeiro **código em Java**, onde você criará um diretório chamado “exercício”: C:\exercicio.



#PraCegoVer: na imagem, temos um *print* o qual mostra a janela que contém a pasta em que o repositório C do computador está. Nela, é demonstrado a criação de diretório chamado “exercício”.

Passo 2

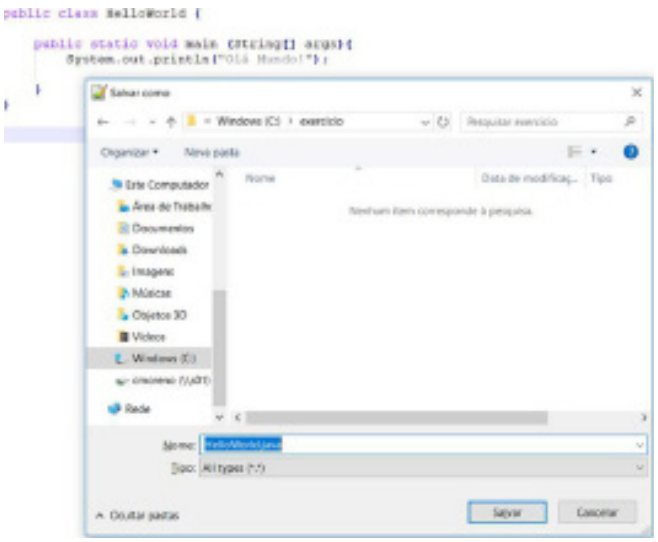
Na sequência, é necessário abrir um editor de texto, como o bloco de notas ou o NotePad++, que são os mais recomendados. Nele digite o código a seguir:

```
Public class HelloWorld {

public static void main (String[] args) {
System.out.println(“Olá Mundo!”);
}
}
```

Passo 3

Depois, salve o arquivo no diretório C:\exercicio, lembrando de utilizar o mesmo nome da classe "*HelloWorld.java*", respeitando, logicamente, a escrita das letras em maiúsculas e minúsculas.



#PraCegoVer: na imagem, temos o código Java ao fundo com um *pop-up* aberto para salvar. Trata-se de uma aba de repositório indicando um local para salvamento do código criado.

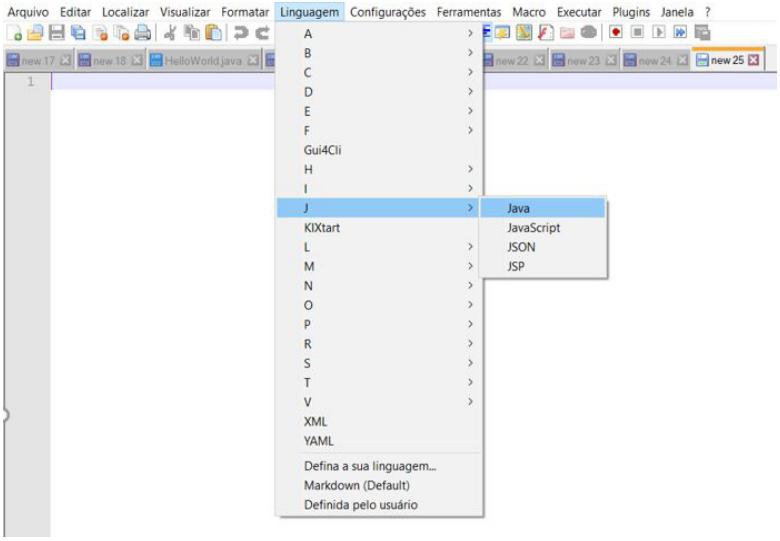
Passo 4

Em seguida, você já pode ver que o arquivo foi criado como "*HelloWorld.java*".

#PraCegoVer: na imagem, temos um *print* de repositório do computador demonstrando o diretório chamado "exercício" com um arquivo salvo: "*HelloWorld.java*".

Passo 5

Para utilizar a estrutura em Java, no menu superior, devemos buscar por “Linguagem”, “J” e selecionar a opção “Java”, como vemos a seguir.



#PraCegoVer: na imagem, temos um *print* indicando o caminho para a estrutura em Java. Em destaque, está a opção “Linguagem”, com “J” e “Java”.

Agora, vamos realizar um exercício para fixar os conhecimentos, iniciando a **compilação do arquivo** “*HelloWorld.java*”. Para isso, abra o *prompt* de comando e navegue até o diretório pelo *prompt* “CD exercício”. O comando “dir” exibirá o conteúdo do diretório que você selecionou.

```
C:\>cd exercicio

C:\exercicio>dir
O volume na unidade C é Windows
O Número de Série do Volume é 9A31-C65E

Pasta de C:\exercicio

09/02/2021  22:21    <DIR>          .
09/02/2021  22:21    <DIR>          ..
09/02/2021  22:21               123 HelloWorld.java
               1 arquivo(s)                123 bytes
               2 pasta(s)     213.918.973.952 bytes disponíveis
```

Na sequência, execute o comando "*javac HelloWorld.java*" e selecione "*Enter*". Desse modo, um arquivo com a extensão "*.class*" será criado, ou seja, um arquivo intitulado "*HelloWorld.class*".

#PraCegoVer: na imagem, temos um *print* de repositório do computador demonstrando o diretório chamado "exercício" com dois arquivos salvos: "*HelloWorld.java*" e "*HelloWorld.class*".

No terceiro momento, vamos executar o código no **Java *Virtual Machine***. Ainda no *prompt*, é preciso digitar o comando "*java HelloWorld*", lembrando que não é necessário inserir a extensão "*.class*", pois, por definição, o Java sempre executará esse arquivo. Ao selecionar "*Enter*", o resultado será exibido na tela: "Olá, Mundo!", como você observar na imagem a seguir:

```
C:\exercicio>javac HelloWorld.java

C:\exercicio>java HelloWorld

Ola Mundo!

C:\exercicio
```

Neste tópico, você acompanhou o passo a passo, de como criar e compilar um programa em Java, sendo que, para isso, foi utilizado o JDK em conjunto com o Java *Virtual Machine*, cuja função é simular e interpretar o código, a fim de que ele seja convertido para a linguagem de máquina e, assim, ser executado pelos computadores e outros dispositivos.

Com esse exemplo aplicado na prática, você já consegue identificar como é a estrutura da linguagem Java, não é mesmo? Contudo, para não restar dúvidas, vamos nos aprofundar em outros conceitos, principalmente, em relação às classes, aos métodos e aos comentários, os quais são elementos fundamentais para a estrutura da linguagem Java.

Estrutura da linguagem Java

O slogan da linguagem Java é "*write once run anywhere*", traduzindo de forma literal, temos algo como "escreva uma vez e execute em qualquer lugar". Porém, pensando a respeito disso, você saberia explicar o que temos dentro de um arquivo Java, ou seja, o que há inserido na fonte?

Na fonte, existe a definição da classe do programa, sendo que, dentro dessa classe, há um conjunto de métodos. Destes, recebemos as instruções do aplicativo. Aliás, para construirmos uma classe, utilizamos a palavra "*class*", seguido do nome da classe que queremos utilizar. Tal nome deverá ser o mesmo do arquivo-fonte que criamos. Observe com atenção as informações na imagem!

```
public class HelloWorld {
```

O corpo da classe deve ser criado entre chaves (no bloco), nos quais poderemos encontrar os métodos, conforme apresentado no código a seguir.

```
public static void main (String[] args) {
System.out.print("Olá Mundo!");
}
}
```

É importante ressaltar que os métodos definem o tipo de retorno, o nome do método e os parâmetros que esse método recebe, os quais são informados entre parênteses, como pode observar na linha de código apresentada:

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

Nos métodos, há instruções, sendo que cada uma deve ser finalizada com o sinal de ponto e vírgula (;). Este símbolo indica o final da instrução e imprime o texto ou exibe o resultado na tela, como "Olá, Mundo!", no *prompt* de comandos.

```
System.out.print("Olá Mundo!");
```

Como pode observar, o texto "Olá, Mundo!" está entre aspas duplas. A este valor, chamamos de ***string*** ou **literal *string***.

Os **comentários** em Java servem para informar aos programadores a respeito da finalidade de cada parte do programa, facilitando o seu entendimento e o que está sendo realizado e/ou executado no momento.

Esses comentários podem ser de três tipos, vamos conhecê-los!

Tipo 1

Comentário de uma única linha.

Tipo 2

Comentário de múltiplas linhas.

Tipo 3

Comentário para documentação do código.

Observe abaixo, por exemplo, como o resultado de um comentário para documentação do código, é apresentado:

```
/**
*Programa que imprime em tela Olá Mundo!
 *@author Professor Carlos Moreno
*/

public class HelloWorld {
public static void main (String[] args) {
System.out.print (“Olá Mundo!”);
}
}
```

Em suma, podemos incrementar o código com as informações necessárias, de modo a torná-lo extremamente legível, como no exemplo apresentado.

Para imprimir mensagens em duas ou mais linhas, utilizamos o comando ***println***. Este adiciona uma linha ao final do comando.

```
/**
* Programa que imprime em tela Olá Mundo!
*seguido de outra linha
*@author Professor Carlos Moreno
*/

public class Helloworld {
/* Método principal da classe*/
public static void main (String[] args) {
System.out.println("Olá Mundo!");
System.out.println("Meu primeiro programa em Java!");
}//fim do método
}//fim da classe
```

Até este ponto, você viu sobre os elementos fundamentais para a estrutura da linguagem Java, ou seja, as classes, os métodos e os comentários, sendo que, paralelamente a partir das telas apresentadas, você notou a aplicação deles, para gerar o programa "Olá, Mundo!".

No entanto, ainda que a estrutura da linguagem Java esteja desvendada, outro aspecto relevante deve ser conhecido, que é a lógica de programação da linguagem Java. Ou seja, a construção de uma série de comandos destinados a executar uma função específica no software.

Assim, veja mais sobre isso adiante!

Lógica de programação e variáveis

Depois de compreender a aplicação dos elementos fundamentais para a estrutura da linguagem Java, agora, devemos entender o raciocínio lógico por trás dele, ou seja, a lógica de programação e variáveis.

A lógica de programação consiste naquilo que utilizamos para escrever ou desenvolver os programas, os quais serão interpretados pelos computadores, utilizando sequências ou etapas, de modo que possam executar uma ou, até mesmo, diversas funções. Essa sequência é chamada de **algoritmo**.

Ainda no contexto da lógica, também temos as **variáveis**, que fazem referência aos dados. Na escola, por exemplo, você aprendeu nas aulas de matemática o conceito de a = 1, b = 2 e c = a + b, certo? Diante disso, esses valores são chamados de variáveis.

> Essencialmente, as variáveis são espaços nos quais definimos em memória para o armazenamento de dados, que serão executados pelo processador.

Veja outro exemplo, onde criaremos um arquivo e o chamaremos de "variável". Ele deve ser salvo como "variavel.java" no diretório C:\exercicio\variavel.java, executado a partir do *prompt* de comandos, como aprendemos anteriormente. Desse modo, teremos "*javac variavel.java*" e "*java variavel*".

Observe a seguir um código de programa sem variável.

```
/**
*Conceitos de Variável
*@author Professor Carlos Moreno
*/

public class Variavel {
/ * Método principal da classe* /
public static void main (String[] args) {
System.out.println ("Texto Simples");
}//fim do método
}//fim da classe
```

A partir dessa etapa, vamos sofisticar a implementação do código. Utilizaremos o código de programa para imprimir o valor da variável "nome" na tela do *prompt* de comando. Após inserir os comandos, salve o código e, na sequência, execute-o para visualizar o resultado. Veja como fica neste exemplo abaixo:

```
/**
*Conceitos de Variável
*@author Professor Carlos Moreno
*/

public class Variavel {
/ * Método principal da classe*/
public static void main(String[] args) {
/* Declaração das nossas variáveis*/
String nome = "Carlos";
int idade = 40;
boolean = casado = true;
/ * Imprime em tela o resultado de nossa variável*/
System.out.println(nome);
}//fim do método }
//fim da classe
```

Neste tópico, com base nos exemplos apresentados, você conheceu a lógica de programação que a linguagem Java traz a partir dos conceitos de algoritmo e das variáveis.

Contudo, ainda é necessário compreender outro aspecto relevante associada à essa lógica de programação, ou seja, estamos falando das palavras reservadas em Java. Assim, no próximo tópico, vamos abordá-las com mais profundidade.

Palavras reservadas em Java

Após estudar sobre a lógica de programação, sobretudo, em relação aos algoritmos e às variáveis, chegou a hora de você conhecer as palavras reservadas. Elas são como identificadoras gramaticais exclusivas. Logo, seu uso é restrito, não podendo ser aplicadas para além do que foi definido a elas.

Na tabela a seguir, você pode compreender quais palavras são essas e que foram criadas para serem utilizadas na linguagem de programação Java.

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

#PraCegoVer: no quadro, temos as palavras reservadas em Java, incluindo: *abstract*, *assert*, *boolean*, *break*, *byte*, *case*, *catch*, *char*, *class*, *const*, *switch*, *transient*, *volatile*, *continue*, *default*, *do*, *double*, *else*, *enum*, *extends*, final, *finally*, *float*, *this*, *try*, *while*, *for*, *goto*, *if*, *implements*, *import*, *intanceof*, *int*, *interface*, *long*, *native*, *throw*, *void*, *new*, *package*, *private*, *protected*, *public*, *return*, *short*, *static*, *strictfp*, *super*, *throws* e *synchronized*.

As palavras reservadas são divididas em oito grupos, vamos conhecer cada um deles.

No **grupo de modificadores** de acesso, você encontra:

Private

Acesso apenas dentro da classe.

Protected

Acesso por classes no mesmo pacote e subclasses.

Public
Acesso de qualquer classe.

Já no **grupo de modifcadores de classes**, **variáveis ou métodos**, temos:

Abstract	Classe que não pode ser instanciada ou método que precisa ser implementado por uma subclasse não abstrata.
Class	Especifica uma classe.
Extends	Indica a superclasse que a subclasse está estendendo.
Final	Impossibilita que uma classe seja estendida, que um método seja sobrescrito ou que uma variável seja reinicializada.
Implements	Indica as interfaces que uma classe irá implementar.
Interface	Especifica uma interface.
Native	Indica que um método está escrito em uma linguagem dependente de plataforma, como o C.
New	Instancia um novo objeto, chamando seu construtor.
Static	Faz um método ou uma variável pertencer à classe em vez de pertencer às instâncias.
Strictfp	Usado na frente de um método ou uma classe para indicar que os números de ponto flutuante seguirão as regras de ponto flutuante em todas as expressões.
Synchronized	Indica que um método só possa ser acessado por um *thread* de cada vez.
Transient	Impede a serialização de campos.
Volatile	Indica que uma variável pode ser alterada durante o uso de *threads*.

Temos, ainda, o grupo de palavras reservadas para **controle de fluxo dentro de um bloco de código**. Neste grupo, você encontrará:

Break	Sai do bloco de código em que se encontra.
Case	Executa um bloco de código dependendo do teste do *switch.*
Continue	Pula a execução do código que viria após a linha e vai para a próxima passagem do *loop*.
Default	Executa o bloco de código caso nenhum dos testes de *switch-case* seja verdadeiro.
Do	Executa o bloco de código uma vez e realiza um teste em conjunto com o *while* para determinar se o bloco deverá ser executado novamente.
Else	Executa um bloco de código alternativo caso o teste *if* seja falso.
For	Utilizado para realizar um *loop* condicional de um bloco de código.
If	Usado para realizar um teste lógico de verdadeiro ou falso.
Instanceof	Determina se um objeto é uma instância de determinada classe, superclasse ou interface.
Return	Retorna de um método sem executar nenhum código que venha depois da linha e pode retornar uma variável. Isto é, finaliza imediatamente a execução do método e transfere o fluxo de execução ao módulo chamador. Utilizamos o *return* para definir o que será retornado quando determinada função for executada.
Switch	Indica a variável a ser comparada nas expressões de *case*.
While	Executa um bloco de código repetidamente enquanto a condição for verdadeira.

Por sua vez, no caso do **grupo de tratamento de erros**, consideramos as seguintes palavras:

Assert
Testa uma expressão condicional para verificar uma suposição do programador.

Catch

Declara o bloco de código usado para tratar uma exceção.

Finally

Bloco de código, após um *try-catch*, executado independentemente do fluxo de programa seguido ao lidar com uma exceção.

Throw

Usado para passar uma exceção para o método que o chamou.

Throws

Indica que um método pode passar uma exceção para o método que o chamou.

Try

Bloco de código que tentará ser executado, mas que pode causar uma exceção.

Para o **grupo de controle de pacotes**, outras palavras reservadas são aplicadas:

Import

Importa pacotes ou classes para dentro do código.

Package

Especifica em qual pacote todas as classes de um arquivo estão inseridas.

Pensando nos **grupos tipos primitivos**, temos as seguintes palavras reservadas:

Boolean	Valor indicando verdadeiro ou falso.
Byte	Inteiro de 8 *bits* (*signed*).
Char	Caractere *unicode* (16 *bit unsigned*).
Double	Número de ponto flutuante de 64 *bits* (*signed*).
Float	Número de ponto flutuante de 32 *bits* (*signed*).
Int	Inteiro de 32 *bits* (*signed*).
Long	Inteiro de 64 *bits* (*signed*).
Short	Inteiro de 32 *bits* (*signed*).

Ainda quanto aos tipos primitivos, é importante mencionar que, como notamos, a linguagem de programação Java oferece oito tipos para utilização, como lógico, caractere, números inteiros e ponto flutuante.

Na tabela a seguir, você entenderá melhor a respeito desses tipos primitivos e suas particularidades. Observe com atenção!

Valores possíveis					
Tipos	**Primitivo**	**Menor**	**Maior**	**Valor Padrão**	**Tamanho**
Inteiro	*byte*	**-128**	**127**	**0**	**8** *bits*
	short	**-32768**	**32767**	**0**	**16** *bits*
	init	**-2.147.483.648**	**2.147.483.647**	**0**	**32** *bits*
	long	**-9.223.372.036.854.770.000**	**9.223.372.036.854.770.000**	**0**	**64** *bits*
Ponto flutuante	*float*	**-1.4024E-37**	**3.4028234+38**	**0**	**32** *bits*
	double	**-4.94E-307**	**1.79769313486231570E+308**	**0**	**64** *bits*
Caractere	*char*	**0**	**65535**	**0**	**16** *bits*
Booleano	*boolean*	**false**	**0**	**0**	**1** *bit*

Também precisamos mencionar **o grupo das variáveis de referência**, conforme apresentado na sequência.

Super
Refere-se à superclasse imediata.

This

Refere-se à instância atual do objeto.

No entanto, temos o **grupo com algumas palavras reservadas que não são utilizadas** na linguagem Java por motivos plausíveis. Veja quais são:

Const

Não se utiliza para declarar constantes, trocando por *public static final*.

Goto

Em Java, é feito pelo *break* ou *continue* com rótulo. É útil para interromper de uma vez só a execução de um laço múltiplo ou pular para determinado ponto do programa.

Neste tópico, você conheceu um conjunto de palavras essenciais para a lógica de programação da linguagem Java: as palavras reservadas. Elas desempenham funções singulares, sendo divididas em oito grupos (modificadores de acesso, modificadores de classes, variáveis ou métodos, controle de fluxo dentro de um bloco de código, tratamento de erros, controle de pacotes, tipos primitivos e variáveis de referência).

Entretanto, além das palavras reservadas, existem outros símbolos que as acompanham e desempenham uma atribuição operacional na aplicação das linhas de programação da linguagem Java, são os chamados operadores. Continue seu estudo para conhecê-los!

Operadores

Anteriormente, você estudou sobre os algoritmos, as variáveis e as palavras reservadas, sendo que cada um é essencial para a linguagem Java. Porém, a fim de que esses itens sejam articulados e, assim, servirem de base para a criação das linhas de programação, é preciso que os operadores entrem em cena.

Nesse sentido, os operadores são símbolos utilizados na execução de operações, assim como nas operações lógicas e matemáticas, estão divididos em três grupos. Sobre eles, observe adiante:

Operadores matemáticos

Aplicam cálculos matemáticos nas linhas de código.

Operadores de incremento

Acréscimo de uma determinada unidade ao valor da variável.

Operadores lógicos

Encadeiam expressões a partir de um raciocínio lógico, sobretudo, atribuindo a noção de verdadeiro e falso.

No quadro a seguir, você encontra os tipos de operadores mais utilizados na linguagem Java.

Operador	Função	Operador	Função
+	adição	~	complemento
-	subtração	<<	deslocamento à esquerda
*	multiplicação	>>	deslocamento à direita
/	divisão	>>>	deslocamento à direita com zeros
%	resto	=	atribuição
++	incremento	+=	atribuição com adição
--	decremento	-=	atribuição com subtração
>	maior que	*=	atribuição com multiplicação
>=	maior ou igual	/=	atribuição com divisão
<	menor que	%=	atribuição com resto
<=	menor ou igual	&=	atribuição com *and*
==	igual	|=	atribuição com or
!=	não igual	^=	atribuição com xor
!	não lógico	<<=	atribuição com deslocamento à esquerda
&&	e lógico	>>=	atribuição com deslocamento à direita
||	ou lógico	>>>=	atribuição com deslocamento à direita com zeros
&	*and*	?:	operação ternário
^	xor	(TIPO)	conversão de tipos (*cast*)
|	or	*INSTANCEOF*	comparação de tipos

#PraCegoVer: no quadro, temos os operadores da linguagem Java com suas funções, incluindo + (adição), - (subtração), * (multiplicação), / (divisão), % (resto), ++ (incremento), -- (decremento), > (maior que), >= (maior ou igual), < (menor que), <= (menor ou igual), == (igual), != (não igual), ! (não lógico), && (e lógico), || (ou lógico), & (*and*), ^ (*xor*) e | (*or*). Há, ainda, ~ (complemento), << (deslocamento à esquerda), >> (deslocamento à direita), >>> (deslocamento à direita com zeros), = (atribuição), += (atribuição com adição), -= (atribuição com subtração), *= (atribuição com multiplicação), /= (atribuição com divisão), %= (atribuição com resto), &= (atribuição com *and*), |= (atribuição com *or*), ^= (atribuição com *xor*), <<= (atribuição com deslocamento esquerdo), >>= (atribuição com deslocamento direito), >>>= (atribuição com deslocamento à direita com zeros), ?: (operação ternário), (tipo) (conversão de tipos (*cast*)) e *instanceof* (comparação de tipos).

Para que fique mais claro os conceitos apresentados até aqui, vamos conhecer mais detalhadamente os grupos dos operadores, iniciando pelos operadores **matemáticos**. Também chamados de binários, esses operadores têm a função de evidenciar o tipo de cálculo a ser empregado na linha de programação.

Considerando isso, observe a tabela e veja os principais tipos de operadores matemáticos, bem como os seus respectivos exemplos.

Operador	Função	Java
+	Adição	2 + 2
-	Subtração	10 - 5
*	Multiplicação	3 * 9
/	Divisão	35 / 7
%	Resto	20 % 10

#PraCegoVer: no quadro, temos os operadores matemáticos principais com suas funções, incluindo + (adição), - (substração), * (multiplicação), / (divisão) e % (resto).

Agora que você já conheceu mais sobre os operadores principais e suas funções, na imagem posterior convidamos você a observar o operador matemático de soma em que uma variável X do tipo *double* recebe a soma de dois números. Tenha atenção aos detalhes!

```
/**
*Operadores
*Operadores Matemáticos - Soma
*@author Professor Carlos Moreno
*/

public class Operador {
/* Método principal da classe */
public static void main (String[] args) {
double x = 7 + 3
System.out.println("x");
}//fim do método
}//fim da classe
```

Para realizar o mesmo procedimento com os demais operadores matemáticos, incluindo outros valores, você pode criar as linhas de código a partir dos seguintes princípios:

```
double x = 2 + 2;
double x = 2 * 2;
double x = 2 – 1;
double x = 10 % 2.
```

Há situações em que você precisará indicar se um número é positivo ou negativo. Para tanto, é possível informar que seu número é positivo ou negativo simplesmente adicionando os respectivos sinais de mais (+) ou menos (-), seguidos do número. Observe como realizar essa operação:

```
/**
*Operadores
*Operadores Matemáticos
*Indica que o número é Positivo
*@author Professor Carlos Moreno
*/
public class Positivo {
/*Método principal da classe*/
public static void main (String[] args) {
double x = +3
System.out.println("x");
}//fim do método
}//fim da classe
```

No código, é exibido o valor de X positivo. Criamos a variável X do tipo *double* e informamos que ela receberá o número 3 positivo.

```
/**
*Operadores
*Operadores Matemáticos
*Indica que o número é Negativo
*@author Professor Carlos Moreno
*/

public class Negativo {
/*Método principal da classe */
public static void main (String[] args) {
double x = -3
System.out.println("x");
}//fim do método
}//fim da classe
```

Já neste código, é exibido o valor de X negativo. Criamos a variável X do tipo *double* e informamos que ela receberá o número 3 negativo.

Já os **operadores de incremento** são responsáveis pelo acréscimo de uma unidade ao valor da variável. Assim, sendo unário, será um operador que não necessitará do uso de outra variável na execução do seu processo.

Para que você entenda melhor, observe como aplicar esse operador no código, apresentado na sequência!

```
/**
*Operadores
*Operadores Matemáticos
*Incremento
*@author Professor Carlos Moreno
*/

public class Incremento {
/*Método principal da classe */
public static void main (String[] args) {

int x = 8;
int y = ++x;//pré-incremento

System.out.println("x=" + x);
System.out.println("y=" + y);

}//fim do método
}//fim da classe
```

Incialmente, temos, um código **pré-incremento**, ou seja, quando o operador de incremento ainda não foi aplicado.

```
/**
*Operadores
*Operadores Matemáticos
*Incremento
*@author Professor Carlos Moreno
*/

public class Incremento {
/*Método principal da classe */
public static void main (String[] args) {

int x = 8;
int y = x++;//pós-incremento

System.out.println("x=" + x);
System.out.println("y=" + y);

}//fim do método
}//fim da classe
```

Agora, vemos como fica o código anterior com a inserção do operador de incremento (**pós-incremento**).

Vale mencionar que, para executar os códigos, você deve considerar duas opções:

Opção 1

Pré-incremento = --x.

Opção 2

Pós-incremento = x++.

Quanto ao uso dos **operadores lógicos**, é preciso que você observe que eles somente aceitam operadores do tipo *boolean*. Acompanhe no quadro a seguir.

Operador	Função	Representa
!	**Não lógico**	Not
&&	**E lógico**	And
||	**Ou lógico**	Or

#PraCegoVer: no quadro, temos os operadores lógicos e suas funções, incluindo ! (não lógico), && (e lógico) e || (ou lógico).

O operador **&&** é chamado de "operador E". Ele retornará *true* se os dois operadores lógicos forem verdadeiros, logo, <operador1> && <operador2>. Contudo, essa não é a única combinação possível. No quadro a seguir, você pode notar que existem outras opções de utilização do operado.

Operando 1	Operando 2	Operando 3
true	true	true
true	false	false
false	true	false
false	false	false

#PraCegoVer: no quadro, temos o quadro da verdade de &&, com operando 1 (*true*, *true*, *false* e *false*), operado 2 (*true*, *false*, *true* e *false*) e operando 3 (*true*, *false*, *false* e *false*).

No código a seguir, você pode ver na prática a aplicação do operador &&.

```
/**
* Classe utilizada para demonstrar o uso do operador logico
E ( && )
*/

public class OperadorLogicoE {
public static void main (String[] args) {
boolean a = true;
boolean b = false;
boolean c = true;
System.out.println(a && b);
System.out.println(a && c);
}
}
```

O **operador ||**, por outro lado, é chamado de "operador OU". Ele retornará *true*

se ao menos um dos dois operadores lógicos forem verdadeiros, logo, temos <operador1> || <operador2>. Sobre isso, veja no quadro seguinte outros tipos de utilização dos operadores.

Operando 1	Operando 2	Operando 3
true	true	true
true	false	true
false	true	true
false	false	false

#PraCegoVer: no quadro, temos o quadro da verdade de ||, com coluna operando 1 (*true*, *true*, *false* e *false*), coluna operando 2 (*true*, *false*, *true* e *false*) e coluna operando 3 (*true*, *true*, *true* e *false*).

No código a seguir, observe como é feita na prática a aplicação do operador|| .

```
/**
* classe utilizada para demonstrar o uso do operador logico
ou (11)
*/
public class OperadorLogicoou {
public static void main (String[] args) {
boolean a = true;
boolean b = false;
boolean c = false;
System.out.println(a || b);
System.out.println(b || c);
}
}
```

O **operador !** é chamado de “operador de negação”. Ele retornará true se o operador lógico for falso, assim como retornará false se o operador lógico for verdadeiro.

Operando 1	Operando 2
true	false
false	true

#PraCegoVer: #praCegover: tabela contendo suas operações, operando 1 com true e false e operando 2 com as operações false e true.

No código a seguir, observe como é realizada a aplicação do operador !.

```
/**
* classe utilizada para demonstrar o uso do operador logico
negação
*/
public class OperadorLogicoNegacao {
public static void main(String[] args) {
boolean a = true;
boolean b = false;
boolean c = false;

System.out.println(!a);
System.out.println(!(b || c));
}
}
```

Assim, fica mais fácil identificar os operadores e como eles trabalham dentro de um código de programação, não é mesmo?

Parabéns! Você chegou ao final do módulo 2.

Desta forma, você pode acompanhar neste módulo uma abordagem mais prática em relação ao funcionamento da linguagem Java. Nesse sentido, foi possível compreender não só os elementos fundamentais dessa linguagem (classes, métodos e comentários), como também entender a aplicação dos algoritmos, das variáveis, das palavras reservadas e dos operadores.

Mas existem outros dois conceitos na linguagem Java, os quais são muito importantes, principalmente quando você for ordenar os dados e avaliar os blocos de programa presentes no código. Tais conceitos são o de fluxo de controle e *arrays*. Vamos lá?!

Módulo 3

Fluxo de controle e *arrays*

fundação bradesco | escola virtual

Fluxo de controle e *arrays*

Ainda continuando uma abordagem mais prática do nosso curso de linguagem Java, iniciaremos este módulo descrevendo os conceitos de fluxo de controle e *arrays*, os quais são muito importantes para que você possa programar adequadamente os softwares a partir da linguagem Java. Sabendo disso, vamos lá.

O fluxo condicional nos permite realizar determinada avaliação de um bloco do programa. De acordo com essa avaliação, podemos executar ou não um trecho de código. Por outro lado, temos a coleção de dados, que é um *array*.

Com base nesses dois assuntos, durante este último módulo, você aprenderá como criar, acessar os elementos, realizar pesquisas dentro de uma coleção e como ordená-la. Verá também, sobre os principais fluxos de controle que são utilizados atualmente. Vamos em frente!

Fluxos condicional e de repetição

No vídeo a seguir, você conhecerá os fluxos condicional e de repetição, incluindo ***if***, ***else***, ***else if***, ***while***, ***do...while*** e ***for***. Acompanhe atentamente para compreender do que se trata cada um desses fluxos e como eles são aplicados!

Vídeo

Confira o vídeo sobre fluxos condicional e de repetição.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Vamos conhecer alguns fluxos de condicional e de repetição?

Iniciando pelo fluxo condicional, temos o *if*, seu uso indica para o programa que desejamos iniciar um teste. Assim, ele avaliará determinada condição entre parênteses (bloco). Caso essa condição retorne um valor verdadeiro *(true)*, o programa executará as instruções definidas no código. Neste exemplo, a variável idade é definida com o tipo *int* (número inteiro) e valor 8.

A condição é: se a idade informada for menor que 10, imprimirá em tela (exibirá) a mensagem “Ainda é uma criança”.

```
*FLUXO CONDICIONAL
*Utilização do IF
*@author Professor Carlos Moreno
*/
public class ControleFLuxo {
/*Método principal da classe*/
public static void main (string [] args) {

int idade = 8; // A variável idade é definida como 8
(8 anos)
if  (idade < 10) { // Se a idade foz menor que 10
(anos)
System.out.println (“Ainda é uma criança”); //
Imprima em tela a mensagem

}//fim do método
}//fim da classe
```

O *else*, por sua vez, significa que, caso o bloco seja falso, executará o comando *else*. Isto é, se o valor da variável, assim como o resto da divisão, for igual a zero, então o número (definido na variável) será par. Do contrário, será exibida a mensagem “este número é ímpar”.

Observe, neste exemplo, que a variável “número” é definida com o tipo *int* e valor 8.

A condição é: se o valor da variável “número” (definida como 8), dividindo por 2 (sinal de divisão é %), com o resultado da divisão (resto da divisão) igual a zero, então o número informado é par.

```
*FLUXO CONDICIONAL
*utilização do IF, ELSE
*@author Professor Carlos Moreno
*/

public class ControlefLuxo2 {
/*Mátodo principal da classe*/
public static void main (Stringl [] args) {

int numero = 8; //define o valor da variável em 8
if ( (numero % 2) == 0) { //verifica se o resto da
divisão é igual a 0
System.out .println (“Este é um número par”); //
Imprima em tela a mensagem
else {
Systen.out.println (“Esse é um número impar”) ; //
Imprima em tela a mensagem

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

Mas, caso seja necessário avaliar outras condições, podemos usar o *else if*, até que alguma das condições esteja satisfeita no código.

A variável “idade” é definida com o tipo *int* e recebe o valor 7.

A condição é: o valor da variável “idade” (definida em 7) é menor ou igual a 7?

Se a condição for verdadeira, o código exibirá em tela a mensagem “Ainda é uma criança”.

Por outro lado, se o valor da variável for maior que 7 e menor ou igual a 18, então o código exibirá em tela a mensagem “É um adolescente”.

Já se o valor da variável for maior que 18 e menor ou igual a 60, o código exibirá em tela a mensagem ”É um adulto”.

Se alguma das condições anteriores não for satisfeita, será exibida a mensagem “Está na melhor idade”, ou seja, o valor da variável será maior que 60.

```
*FLUXO CONDICIONAL
*Utilização do IF, ELSE IF, ELSE
*@author Professor Carlos Moreno
*/
public class ControleFluxo3 {
/*Método principal da classe*/
public static void main (String [ ] args) {

int idade = 7;
if (idade <= 7){ //a idade é menor ou igual a 7?
Systen.out .println(“Ainda é uma criança”); // se
verdadeiro, imprima

} else if (idade > 7 && idade <= 18) //a idade é
maior que 7 e menor ou igual a 18?
System.out .println(“É um adolescente”); // se
verdadeiro, imprima

} else if (idade > 18 && idade <= 60)  //a idade é
maior que 18 e menor ou igual a 60?
System.out .println(“É um adulto”); // se
verdadeiro, imprima

} else {
System.out .println(“Está na melhor idade”); //
senao, imprima

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

Agora falando sobre os laços ou *loops*, ou estruturas de repetição, estas são utilizadas na execução, de modo repetido, de determinado bloco ou certa instrução, enquanto uma condição estiver sendo satisfeita.

Dentro disso, a estrutura *while* é utilizada quando desejamos executar esses laços em determinado bloco, enquanto a condição for verdadeira.

O código traz uma variável "i" do tipo *int* (inteiro), recebendo o valor zero.

A estrutura de repetição *while* informa que, enquanto o valor da variável "i" for menor que 2, o código exibirá o resultado em tela, mostrando os valores que foram incrementados em "i++", ou seja, serão exibidos em tela os valores 0, 1 e 2.

```
*FLUXO DE REPETIÇÃO
*Utilização do WHILE (ENQUANTO)
*@author Professor Carlos Moreno
*/
import java.util.arraylist;
//Importamos a classe ArrayList para poder fazer
uso do vetor e armazenar elementos.

public class While {
/*Método principal da classe*/
public static void main (String [ ] args) {
//Fluxo de repetição usando While
int i = 0; //a variável i recebe o valor zero (0)
```

```
while (i < 2) { // enquanto i for menor que 2
System.out.println(i); //imprima valor de i
i++; //incremento da variável i.
}//fim do método
}//fim da classe
```

A estrutura *do...while*, por sua vez, inicia a execução da instrução, imprimindo e incrementando o valor da variável "i", enquanto este for verdadeiro.

O código traz uma variável "i", com o valor 3.

A estrutura de repetição *do...while* informa que, enquanto o valor da variável "i" for menor que 5, o código será executado enquanto a condição não ultrapassar do valor 4.

Isto é, a estrutura de repetição executará o programa enquanto o valor inserido for menor que 5. Ou seja, na saída estarão os números 3 e 4.

```
*FLUXO DE REPETIÇÃO
*Utilização DO...WHILE (FAÇA...ENQUANTO)
*@author Professor Carlos Moreno
*/
import java.util.arraylist;
//Importamos a classe ArrayList para poder fazer
uso do vetor e armazenar elementos.

public class Dowhile {
/*Método principal da classe*/
public static void main (String[ ] args) {
//Fluxo de repetição usando While
int i = 3; //a variável i recebe o valor três (3)
```

```
do {
System.out .println(i); //imprima valor de i
i++; //incremento da variável i.
}
while (i < 5) { // enquanto i for menor que 5
}//fim do método
}//fim da classe
```

Para finalizar, a estrutura *for* é semelhante à *do...while*.

O código traz uma variável "i", iniciando o valor em 0 e informando que o valor da variável é menor que 5, incrementando-a. Isto é, será exibido em tela o resultado da variável "i" de zero a cinco, tal como 0, 1, 2, 3 e 4.

O número 5 não será exibido, pois a variável está definida como menor que 5 (< 5). Caso estivesse como <=5, ele seria exibido.

```
/**
*FLUXO DE REPETIÇÃO
*Utilização FOR
*@author Professor Carlos Moreno
*/

class FOR {
public static void main(String args []) {
for (int i = 0; i < 5; i++ ) { //ordem crescente, de
0 à 5
System.out.println(i);
}
}
}
```

Agora é com você! Pratique estes fluxos modificando as condições iniciais mostradas aqui para se familiarizar cada vez mais com a linguagem.

No vídeo, você pôde entender na prática como o fluxo de controle é criado, de modo a ser aplicado em pesquisas, no ordenamento das linhas de código etc.

Além disso, você entendeu os dois tipos de fluxo de controle condicional e de repetição, sobretudo, no que diz respeito ao emprego dos comandos *if*, *else*, *else if*, *while*, *do...while* e *for*, observando a sua estrutura a partir de exemplos extraídos de *prints*.

Agora, que tal ir ao estudo dos *arrays*? Na sequência, você poderá entender como iniciar esse elemento e ter contato com os *arrays* multidimensionais. Confira o que preparamos!

Iniciando *arrays*

Após entender o fluxo de controle, seus tipos (condicional e de repetição) e como eles são construídos, você estudará acerca dos *arrays*. Eles dizem respeito a um conjunto indexado de informação, ou seja, ele armazena informações da mesma natureza. Assim, para iniciar um *array*, primeiro, precisamos definir o tipo do conjunto e, depois, indexar a chave em determinado valor.

No exemplo apresentado a seguir, temos uma lista dos países que compõem o BRICS (acrônimo formado pelas iniciais dos países Brasil, Rússia, Índia, China e África do Sul, sendo um agrupamento de países de mercado emergente em relação ao seu desenvolvimento econômico).

Observe que a primeira posição na lista inicia em zero, logo, o primeiro componente corresponde ao Brasil, o segundo à Rússia, e assim por diante.

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

String[] paises = {“BRASIL”, “RÚSSIA”, “INDIA”, “CHINA”);

System.out.println(paises[0]);

}
}
```

Podemos utilizar a classe “*java.util.Arrays*”, mas que não faz parte do pacote “*lang*” do Java. Desse modo, devemos importá-la utilizando o comando “*import java.util.Arrays*”, para que seja possível aplicar os recursos do exemplo.

```
import java.util.Arrays;

public class Arraysimples {
public static void main (String [] args) {
String[] paises = {“BRASIL”, “RUSSIA”, “INDIA”, “CHINA”};

System.out.println(paises [0]);
System.out.println(Arrays.toString (paises))
}
}
```

Nesta etapa dos seus estudos, você já pode imaginar que precisa realizar uma pesquisa no *array*. Esta pesquisa deverá retornar um país que compõe o *array*. Para tanto, utilize um objeto chamado “*arrays*”, com o método “*binarySearch*” para buscar o elemento. Em nosso caso, vamos buscar pelo país Brasil, inserido na lista países. Será, então, retornada a posição do elemento, conforme definido no método “*binarySearch*”, na coleção.

```
import java.util.Arrays;

public class ArraySimples {
public static void main(String[] args) {
String[] paises = {“BRASIL”, “RÚSSIA”, “INDIA”, “CHINA”);

System.out.println(paises[0]);
System.out.println(Arrays.toString (paises))

int posicao = Arrays.binarySearch (paises, “Brasil”)
System.out.println(posicao)
}
}
```

Igualmente, será possível ordenar o *array* utilizando a classe “*Arrays.sort()*”. Observe a próxima imagem para entender o processo.

```
import java.util.Arrays;

public class ArraySimples {
public static void main (String[] args) {
String[] paises = {“BRASIL”, “RÚSSIA”, “INDIA”, “CHINA”}

System.out.println(paises[0]);
System.out.println(Arrays.toString (paises));

int posicao = Arrays.binarySearch (paises, “Brasil”);
System.out.println(posicao);

Arrays. sort (paises, 0, paises.length); //ordena o nosso
array iniciando em zero
System.out.println (Arrays.toString (paises)); //imprime o
resultado em tela
}
}
```

É importante mencionar que você ainda pode utilizar *arrays* com duas ou mais dimensões, informando no código qual elemento deseja obter o acesso. Veja no exemplo abaixo:

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

String[] [] duasdim = {{“Carlos”, “M”, “SP”}, {“Lidiana”, “F”,
“SP”}};

System.out.println (duasdim [0][0]);
}
}
```

Observe que adicionamos colchetes para cada elemento que desejamos incluir. Desse modo, o primeiro colchete representa o primeiro conjunto de elementos do array {“Carlos”, “M”, “SP”}, enquanto o segundo colchete representa qual elemento do array retornará.

Os valores presentes nos colchetes são lidos da seguinte maneira: 0 correspondente à primeira casa do array e 1 corresponde à segunda casa do array.

Desse modo, no exemplo, serão exibidas as informações: “Carlos” e “Lidiana”, pois nos dois arrays o valor 0 corresponde às casas aos quais pertencem. Caso os valores dos colchetes fossem [1] [1], teríamos como resultado a exibição dos valores “M” e “F”.

Você compreendeu, neste tópico, as principais características ligadas ao conceito de *array* e a maneira pela qual é possível criá-lo e indexar um valor nele. Para isso, utilizamos exemplos práticos que ajudaram a fixar os conceitos apresentados.

Fechamento

Parabéns, você finalizou o curso de Java Básico. Você aprendeu as principais características da linguagem Java. No primeiro módulo apresentamos uma breve trajetória histórica dela desde a sua criação nos anos 90 do século passado até os dias de hoje, ressaltando dois aspectos singulares: a portabilidade e a segurança. Em seguida, você aprendeu como baixar, instalar e fazer o cadastro do *login* do Java *Development Kit* (JDK), kit de criação de linhas de códigos baseado em Java.

Em seguida, foi a vez de aprender de uma forma mais prática, o funcionamento da estrutura da linguagem Java a partir dos seus elementos fundamentais, ou seja, o algoritmo, as variáveis, as palavras reservadas e os operadores. Por fim, no último módulo, você compreendeu o conceito de fluxos de controle (condicionais e de repetição) e de *arrays*, além de visualizar o uso deles por meio de exemplos práticos.

É válido frisar, que tudo o que foi abordado é fundamental, não só porque vai lhe ajudar a compreender aspectos mais complexos da linguagem Java, mas também será de grande relevância em sua futura prática profissional.

Lembre-se, é importante é sempre praticar e se manter atualizado a respeito dos assuntos apresentados aqui, uma vez que os avanços tecnológicos e as alterações na área são constantes!

Boa sorte em sua jornada.

Referências

BURD, B. **Começando a programar em Java para leigos**. Rio de Janeiro: Alta Books, 2014.

DEITEL, P. J.; DEITEL, H. **Java**: how to program. 11. ed. São Paulo: Pearson, 2017.

DE OLIVEIRA, Alexandre Dutra; DA SILVA, Luis Fernando. **Desenvolvimento de software aplicativo comercial com C# e camadas**. Clube de Autores (managed), São Paulo, 2017.

ELCHINATOR. PIXABAY. Disponível em: https://pixabay.com/pt/photos/origem-c%C3%B3digo-software-computador-4280758/. Acesso em: 16 maio 2021.

FREEPIK. **Graphic resources for everyone**. Freepik, 2021. Disponível em: www.freepik.com/br.

JAVA Logo. **Marcas Logos**, [*s. l.*], [*s. d.*]. Disponível em: https://marcas-logos.net/java-logo/. Acesso em: 11 maio 2021.

JAVA SE Development Kit 8 downloads. **Oracle**, [*s. l.*], [*s. d.*]. Disponível em: https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html. Acesso em: 11 maio 2021.

SCHILDT, H. **Java para iniciantes**. 6. ed. Porto Alegre: Bookman, 2015.

SIERRA, K.; BATES, B. **Use a cabeça!** JAVA. 2. ed. Rio de Janeiro: Alta Books, 2007.

YOU ALWAYS wondered: Duke, the Java mascot #Java #Duke @Oracle. **Adafruit**, [*s. l.*], 7 mar. 2019. Disponível em: https://blog.adafruit.com/2019/03/07/you-always-wondered-duke-the-java-mascot-java-duke-oracle/. Acesso em: 9 fev. 2021.