

Universidade Presbiteriana Mackenzie  
Faculdade de Computação e Informática  
Ciência da Computação  
Estrutura de Dados I – 3ª etapa  
Prof. André Kishimoto ([andre.kishimoto@mackenzie.br](mailto:andre.kishimoto@mackenzie.br))

## Configuração da máquina para desenvolvimento Java

### Sumário

Objetivos.....	1
Introdução.....	2
TL;DR.....	2
Ferramentas .....	3
JDK – Java Development Kit .....	3
Hello, World! em Java.....	14
Um pouco sobre o <i>prompt</i> .....	14
Criando um diretório/pasta .....	17
Criando o arquivo Hello.java.....	18
Compilando o arquivo Hello.java.....	21
Executando o programa Hello(.class).....	21
Entendendo o código do Hello.java.....	22
Eclipse IDE.....	25
Hello, World! no Eclipse .....	28
Criando um projeto Java.....	29
Criando o arquivo Hello.java.....	31
Compilando o projeto .....	34
Executando o projeto .....	35
Apêndice A: Histórico deste material.....	42

### Objetivos

Espera-se, após a leitura deste material, I) que a pessoa tenha uma máquina configurada corretamente para desenvolvimento Java com o JDK 17 21 e Eclipse 2022.12 2023-12; II) que tenha conseguido obter uma visão geral do que é Java; III) que tenha criado com sucesso um “Hello, World!” em Java, compilando e executando o exemplo por linha de comando e pelo Eclipse.

**Observação:** Consulte o Apêndice A para atualizações realizadas no material.

## Introdução

Este material descreve brevemente as ferramentas que serão usadas na disciplina **Estrutura de Dados I** do curso de **Ciência da Computação** da **Universidade Presbiteriana Mackenzie**.

Neste texto, veremos como obter e configurar as ferramentas, assim como compilar e executar um código escrito em Java.

Em caso de dúvidas, entre em contato com o professor.

### PARA PENSAR, PRATICAR E EXPLORAR...

As ferramentas indicadas neste material estão disponíveis para Windows, Linux e macOS, sendo que as principais diferenças estão relacionadas com a configuração do ambiente (por exemplo, variáveis de ambiente) e os comandos do terminal.

Este material foi escrito usando o sistema operacional Windows 10 64 bits. Se você não usa Windows, fica como exercício a adaptação do conteúdo apresentado no texto para o sistema operacional de sua escolha.

## TL;DR

### ATENÇÃO!

Optei por remover a seção TL;DR (*Too Long; Didn't Read*) do material.

Minha recomendação é você criar o seu próprio resumo TL;DR deste material, listando os pontos que você acha importante para configurar a máquina para desenvolvimento Java.

Peço, por favor, que compartilhe o seu TL;DR comigo, enviando o seu resumo por e-mail ([andre.kishimoto@mackenzie.br](mailto:andre.kishimoto@mackenzie.br)), para que eu consiga analisar se algum tópico não ficou claro, se algo não foi explicado ou se cometi algum erro, além de avaliar a utilidade deste material.

Sugestões e críticas construtivas são bem-vindas!

Obrigado,  
André Kishimoto

## Ferramentas

Para desenvolver programas com a linguagem Java, precisamos ter pelo menos duas ferramentas:

1. O Kit de Desenvolvimento Java, conhecido como **JDK**<sup>1</sup> (*Java Development Kit*), e
2. Um editor de texto simples/puro (*plain text*) para escrever nossos códigos Java.

No entanto, é muito comum as pessoas usarem um IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) no lugar de um editor de texto, uma vez que IDEs são específicos para desenvolvimento de software e oferecem diversas ferramentas e funcionalidades que auxiliam no trabalho de desenvolvimento.

Uma das funcionalidades oferecidas por IDEs é o editor de texto. Assim, podemos substituir o editor de texto por um IDE.

Nas nossas aulas usaremos o **Eclipse**, um IDE gratuito que não exige instalação e está disponível para Windows, Linux e macOS.

### PARA PENSAR, PRATICAR E EXPLORAR...

Se você pesquisar por “Java IDE” na internet, provavelmente encontrará diversas opções, como Eclipse, IntelliJ IDEA, NetBeans, JCreator, entre outros.

Embora o IDE que usaremos é o Eclipse, recomendo você explorar as opções de IDEs disponíveis para desenvolvimento Java.

Veja *screenshots*/vídeos, leia opiniões a respeito dos IDEs e explore de forma mais ativa: baixe, instale e use alguns IDEs por conta própria. De repente, você se adapta melhor com um outro IDE que não o Eclipse.

A seguir, veremos como obter e configurar o JDK.

## JDK – Java Development Kit

O JDK é composto por diversos componentes/ferramentas que permitem o desenvolvimento e execução de software escrito com a linguagem Java.

Dentre os diversos componentes, conhecidos como *Core JDK Tools*, encontram-se o interpretador Java (**java**), o compilador Java (**javac**), o *disassembler* (**javap**), o empacotador (**jar**), o gerador de documentação (**javadoc**) e o REPL/Java Shell (**jshell**).

---

<sup>1</sup> Às vezes também chamado de Java SDK (*Java Software Development Kit*).

### SAIBA MAIS

O termo REPL vem de *Read-Eval-Print Loop*:

O programa lê o comando que uma pessoa digita (*Read*), depois avalia a entrada fornecida pela pessoa (*Eval* ou *Evaluate*) e, de acordo com a especificação do programa, exibe algo em tela (*Print*). Esses passos ficam em um *Loop* até que o programa seja encerrado.

O JDK também fornece o **JRE** (*Java Runtime Environment* ou Ambiente de Execução Java), que possui os binários Java e outros arquivos necessários para execução de programas Java, além de implementar a **JVM** (*Java Virtual Machine* ou Máquina Virtual Java), responsável por converter o código Java compilado (*Java bytecode*) para código de máquina, gerenciar memória, entre outros.

Além das ferramentas de desenvolvimento e o JRE, o JDK possui as bibliotecas necessárias para escrever códigos Java (**Java API**).

A Figura 1 resume a relação entre JDK, JRE e JVM.

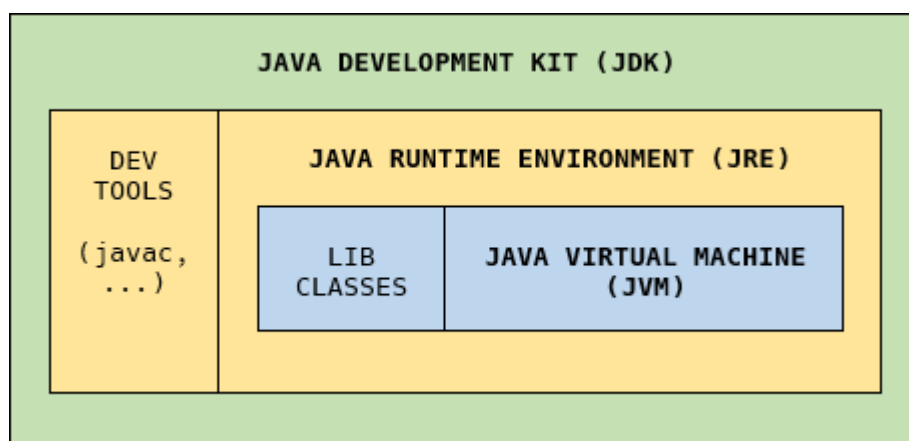


Figura 1: JDK vs. JRE vs. JVM.

### SAIBA MAIS

Caso você queira apenas executar programas Java, a máquina onde você rodará os programas pode ter apenas o JRE (que inclui a JVM) instalado. O JDK é obrigatório somente para desenvolvimento Java.

Agora que já temos uma ideia do que é o JDK, surge a pergunta:

→ *Qual JDK devo usar?*

Uma pesquisa por “JDK” na internet te apresentará diversas opções de JDK, o que pode ser confuso no começo.

### → *Como assim, “diversas opções de JDK”?*

Pois é. Agora que Java pertence à Oracle (após aquisição da Sun Microsystems em 2009-2010), existem diversas implementações do JDK. A própria Oracle possui dois JDK: *Oracle JDK* (gratuito para desenvolvimento e testes, mas pago se usado em ambiente de produção) e *Oracle OpenJDK* (gratuito para qualquer ambiente).

Além das versões da Oracle, existem JDK implementadas pela Amazon (*Corretto*), Alibaba (*Dragonwell*), Eclipse (*Temurin*), Microsoft (*OpenJDK*), e outros. Veja uma lista de JDKs em [1].

### → *OK... Mas, voltando à pergunta inicial: Qual JDK devo usar?*

Você pode usar qualquer implementação do JDK atualmente disponível (algumas são comerciais, mas a maioria é gratuita). No entanto, o JDK que será usado neste material (e nas aulas da disciplina) é especificamente a versão ~~JDK 17~~ **JDK 21**<sup>2</sup>, disponível no site <https://jdk.java.net/21> [2].

### → *Ih, tem versões diferentes do JDK?*

Sim! A linguagem Java, assim como outras linguagens de programação, está em constante atualização. Novas funcionalidades são adicionadas à linguagem, outras são marcadas como obsoletas e removidas, há melhorias de performance, e assim por diante.

### → *E por que, especificamente, JDK 17 21? Vi no site da Oracle que já existe o JDK 20 (e não sei o que aconteceu com o JDK 18 e 19...).*

A Oracle trabalha com dois modelos de atualização da linguagem Java [3]:

1. A cada seis meses, uma nova versão do Java é lançada. Essas versões focam em [novas] funcionalidades da linguagem e substituem a versão anterior. Por exemplo, o JDK 18 foi lançado em março de 2022. O JDK 19 foi lançado em setembro de 2022, seis meses depois do JDK 18, e substituiu o JDK 18. Por isso que o JDK 18 “sumiu” da página principal de downloads da Oracle. Agora que o JDK 20 foi lançado em março de 2023, o mesmo aconteceu com o JDK 19.
2. A cada dois ou três anos, é definida uma versão LTS (*Long-Term Support*) do Java. E a cada trimestre, a versão LTS da linguagem recebe atualizações que contém apenas melhorias de performance, segurança e estabilidade. Ou seja, não há novas funcionalidades na linguagem.

---

<sup>2</sup> **Atualização 07/02/2024:** A versão do Java 21 mais recente em fevereiro/2024 é a 21.0.2 e é a versão que usarei na minha máquina pessoal durante o curso.

**Atualização 03/08/2023:** A versão do Java 17 mais recente em agosto/2023 é a 17.0.8 e é a versão que usarei na minha máquina pessoal durante o curso.

Neste material ainda há textos e imagens fazendo referência ao JDK 17.0.6; apesar disso, o conteúdo ainda é válido para a versão 21.

## SAIBA MAIS

Diversas empresas de tecnologia oferecem versões LTS de seus produtos. LTS é uma maneira de gerenciar o ciclo de vida de um produto. No caso de software, uma versão LTS é considerada estável e recebe atualizações (manutenção) por um período geralmente maior do que as versões comuns.

Quando um software (ou tecnologia, como a linguagem Java) possui uma versão indicada como LTS, a pessoa que usa o software/tecnologia entende que aquela versão é estável e atualizações são apenas correções pontuais – não há nenhuma alteração considerável no software/tecnologia.

Dessa maneira, a pessoa não precisa ficar preocupada se o seu projeto vai “quebrar” ao atualizar a versão LTS do software/tecnologia (pelo menos isso que é o esperado de uma versão LTS).

Neste momento (~~agosto de 2023~~ fevereiro de 2024), a versão **LTS** mais recente do JDK é a ~~17~~ 21 (também conhecido como **Java 17** **Java 21**). ~~Já a versão mais recente do JDK é a 20 (Java 20), que será substituído pelo JDK 21 em setembro de 2023.~~ Portanto: versão LTS – esse é o motivo pela escolha do Java 17.

## ATENÇÃO!

Por conta do que foi explicado sobre LTS, ao iniciar um novo projeto, empresas e desenvolvedores dão preferência às versões LTS das ferramentas e tecnologias que vão usar para desenvolver o projeto.

Além disso, geralmente não atualizamos softwares e tecnologias para a sua última versão durante o desenvolvimento de um projeto (a menos que sejam atualizações da versão LTS).

Tenha em mente que sempre ficar atualizando um software/tecnologia para sua versão mais recente pode não ser uma boa decisão para um projeto, principalmente se ele já estiver em desenvolvimento, pois ao fazer isso corremos o risco de quebrar o projeto, sendo necessário restaurar a versão anterior do software/tecnologia (e reverter possíveis mudanças no projeto).

Em outras palavras: ***evite atualizar software/tecnologia no meio de um projeto!***

→ **Entendi... Acho.**

Ótimo! Mas lembre-se: se tiver dúvidas, avise o professor!

Para ajudar um pouco, a Figura 2 possui algumas partes destacadas que indicam o caminho para baixar o JDK 17 para Windows.

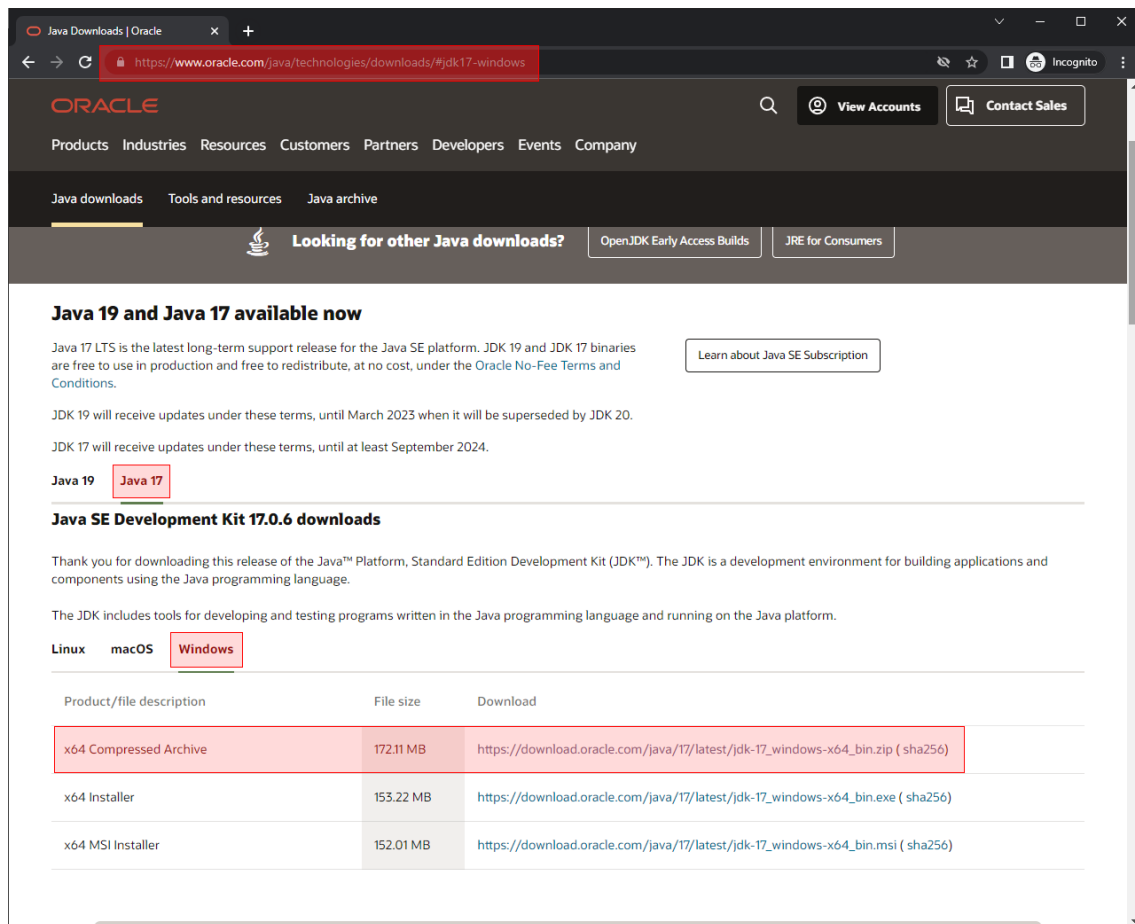


Figura 2: Download do JDK 17.

Observe que a versão **x64 Compressed Archive** está destacada, cujo link para download da versão mais recente é: [https://download.oracle.com/java/17/latest/jdk-17\\_windows-x64\\_bin.zip](https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip).

A versão no formato zip não possui instalador e foi escolhida para que não haja necessidade de usarmos uma conta de administrador para configurar o JDK na máquina.

→ **Baixe o arquivo `jdk-17_windows-x64_bin.zip`. E agora?**

Primeiro, descompacte o arquivo em um local que você se lembre depois (por exemplo, `C:\java`). Dependendo de como você descompactou o arquivo, todos os arquivos contidos no zip devem estar dentro de uma pasta `jdk-17.x.y`, onde `x` e `y` podem variar de acordo com a versão mais recente do Java 17 que você baixou (por exemplo, se é a versão Java 17.0.6, então a pasta que existe dentro do arquivo zip é `jdk-17.0.6`).

Veja a Figura 3 – nesse exemplo, o arquivo foi descompactado na unidade/drive `D:` da máquina, dentro da pasta `\dev\tools`.

Em seguida, recomendo renomear a pasta `jdk-17.x.y` para simplesmente `jdk-17`. Assim, caso você atualize o JDK 17, basta substituir o conteúdo dentro dessa pasta `jdk-17` pelos arquivos da versão mais recente.

Agora, vamos configurar a variável de ambiente `JAVA_HOME`.

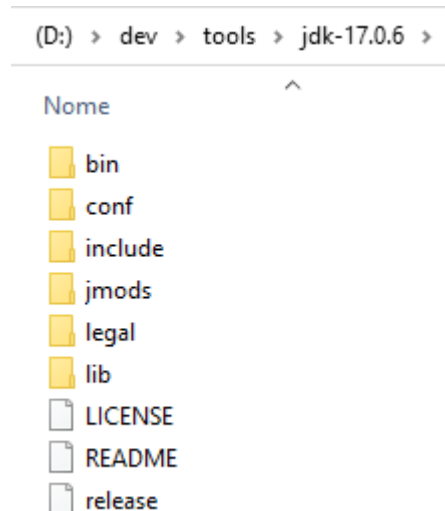


Figura 3: Conteúdo do arquivo zip do JDK 17.

### ATENÇÃO!

Hoje em dia, é muito comum trabalharmos com diretórios (pastas) e arquivos que possuem espaços em branco e caracteres acentuados/especiais, por exemplo: `C:\Users\Meu Usuário\Documents\Trabalho de Programação - EDI.java`.

Porém, quando estiver programando, recomendo acostumar-se a trabalhar com pastas e arquivos *sem* espaços em branco e *sem* letras acentuadas e símbolos especiais, pois ainda há ferramentas que não funcionam muito bem quando encontram esses casos nos nomes das pastas e arquivos.

Uma sugestão é eliminar os espaços. Por exemplo, `Trabalho de Programação - EDI.java` é renomeado para `TrabalhoDeProgramacao-EDI.java` e salvo em `C:\Users\MeuUsuario\Documents\` (observe que não há espaço em branco entre `Meu` e `Usuario`, que perdeu o acento no a).

Outra sugestão é usar o símbolo *underscore* no lugar do espaço em branco. Dessa outra forma, `Trabalho de Programação - EDI.java` é renomeado para `Trabalho_de_Programacao-EDI.java`.

### → *Oi? Variável de ambiente?*

Exatamente. Podemos criar, alterar e excluir variáveis que são usadas pelo sistema operacional. Nesse caso, precisamos criar uma variável de ambiente chamada `JAVA_HOME`.

### → *E como eu crio essa variável JAVA\_HOME?*

Assumindo que você está usando o Windows 10, abra as *Configurações*, localizado acima do ícone *Desligar*, no menu Iniciar (Figura 4).



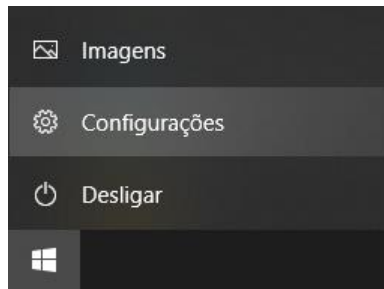


Figura 4: Abrindo as Configurações do Windows.

O aplicativo *Configurações* mostra uma caixa de texto de busca (“*Localizar uma configuração*”). Digite **var** e veja que o aplicativo exibe duas opções (Figura 5). Escolha a opção “*Editar as variáveis de ambiente para sua conta*”.

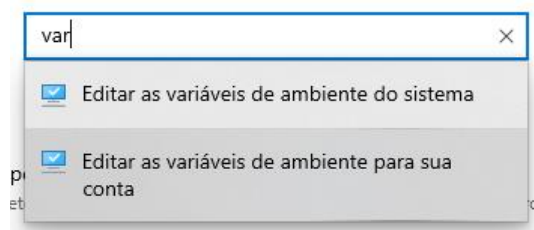


Figura 5: Busca por "var" em Configurações.

Na janela que se abre (“*Variáveis de ambiente*”), clique no botão “*Novo...*” do primeiro grupo (“*Variáveis de usuário para <nome do usuário>*”), conforme a Figura 6.

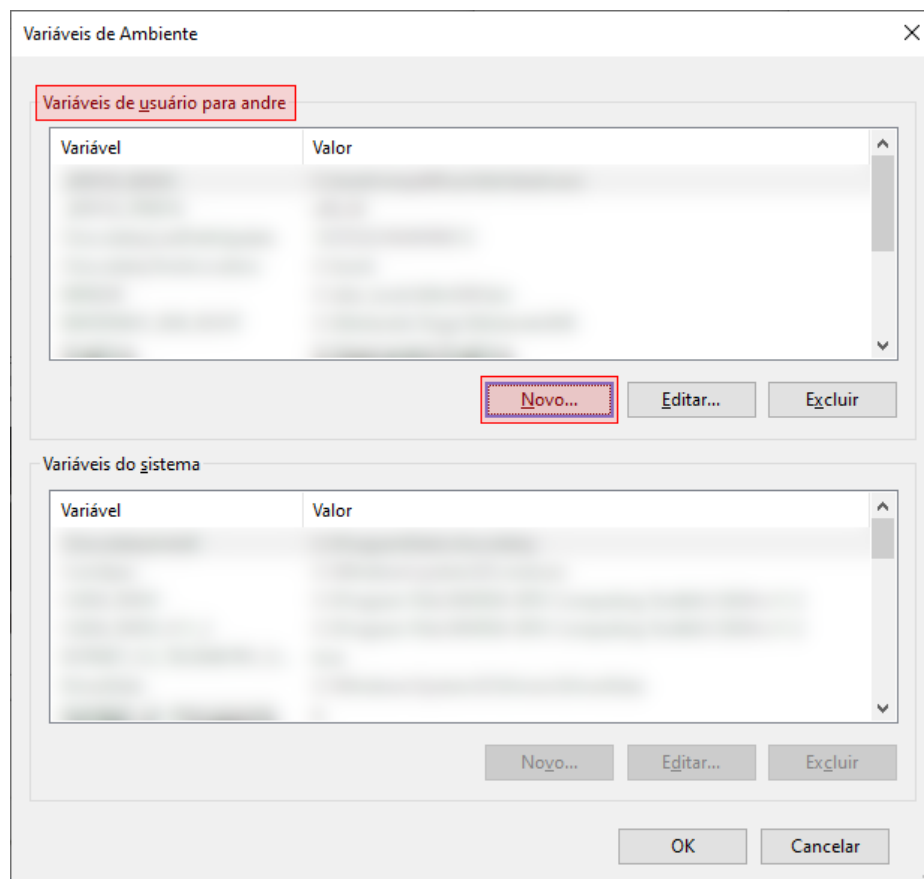


Figura 6: Começando a criar uma variável de ambiente no Windows.

Na nova janela (“Nova Variável de Usuário”), digite **JAVA\_HOME** no campo “Nome da variável:” e no campo “Valor da variável:”, insira o caminho onde você descompactou o JDK. No exemplo da Figura 7, o JDK foi descompactado em **D:\dev\tools\jdk-17**.

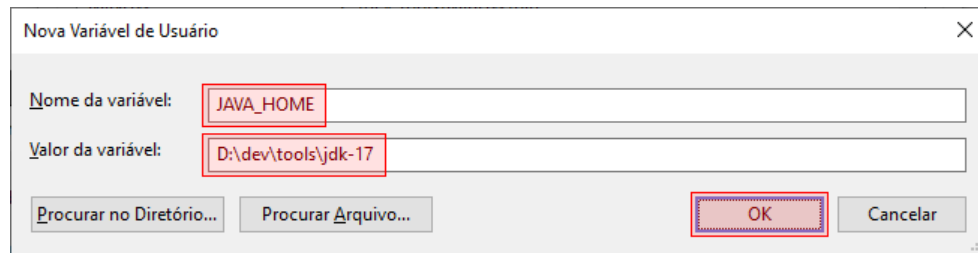


Figura 7: Criando a variável de ambiente JAVA\_HOME.

Após criarmos a variável **JAVA\_HOME**, precisamos alterar a variável **PATH**, que já existe no sistema operacional.

→ *O que devo alterar na variável PATH e como fazer a alteração?*

Na mesma janela e grupo da Figura 6, localize a variável **PATH** (ou **Path**). Com essa variável selecionada, clique no botão “*Editar...*” (Figura 8).

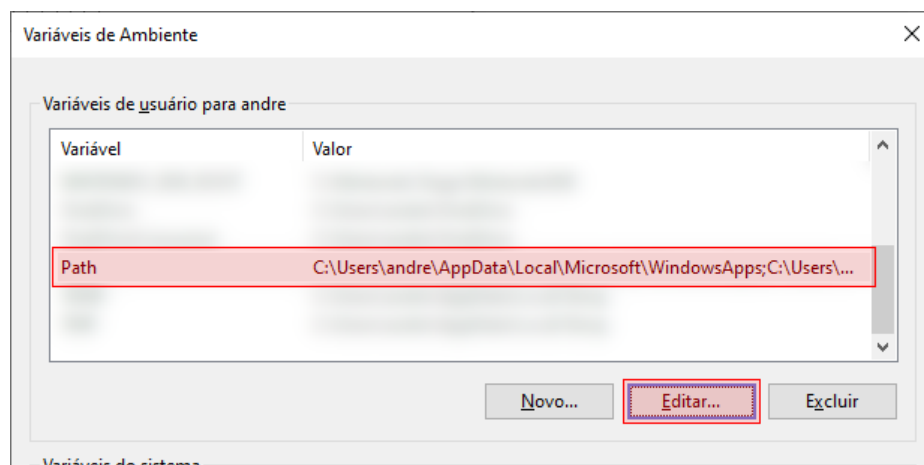


Figura 8: Começando a editar a variável de ambiente PATH.

Na janela que se abre (“*Editar a variável de ambiente*”), clique no botão “*Novo*” e, no campo de texto que aparece para inserir texto, digite **%JAVA\_HOME%\bin**, como destacado na Figura 9. Em seguida, clique em “*OK*” para confirmar a alteração.

### PARA PENSAR, PRATICAR E EXPLORAR...

Neste material, estamos criando uma variável de ambiente específica para um usuário do Windows. Portanto, a variável **JAVA\_HOME** só será reconhecida pelo sistema operacional para esse usuário específico.

Caso queira configurar o JDK para qualquer usuário do Windows, a variável **JAVA\_HOME** deve ser criada no segundo grupo (“*Variáveis do sistema*”) e a variável de ambiente **PATH** a ser alterada também deve ser a do segundo grupo.

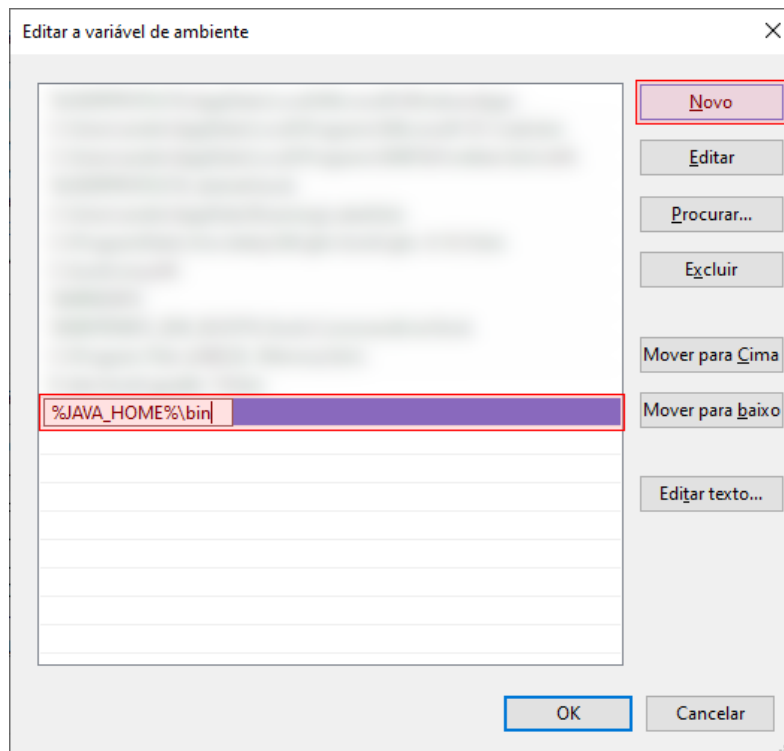


Figura 9: Alterando a variável PATH para incluir o JAVA\_HOME.

Por último, clique em “OK” para fechar a janela “Variáveis de Ambiente” (Figura 6).

### ATENÇÃO!

É importante que `%JAVA_HOME%\bin` esteja no começo do PATH, para que o sistema use a versão correta do JDK, como na Figura 10 a seguir (observe que que `%JAVA_HOME%\bin` vem antes de `C:\Program Files\Microsoft\jdk-11.0.16.101-hotspot\bin` – caso a ordem estivesse invertida, o sistema usaria o JDK 11 ao invés do JDK 17).

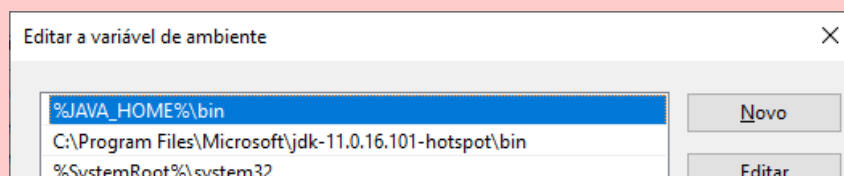


Figura 10: `%JAVA_HOME%\bin` no início de PATH.

→ *Tem algo mais que devo fazer?*

Não, a configuração do JDK é isso!

→ *Ufa!*

Mas tem mais um passo importante, que é verificar se configuramos corretamente o JDK.

→ *Suspeitava que tinha mais coisa... E agora, o que faço para verificar a configuração do JDK?*

Abra o *Prompt de Comando*<sup>3</sup> do Windows. Duas sugestões de como fazer isso:

1. Abra o menu Iniciar e digite `cmd`. O Windows deve filtrar o menu Iniciar e mostrar como melhor correspondência o aplicativo *Prompt de Comando*. Clique na opção “Abrir”.
2. Use o atalho **Windows** + R (teclas Windows + R) para abrir a janela “Executar”. No campo “Abrir:”, digite `cmd` e clique em “OK” (Figura 11).

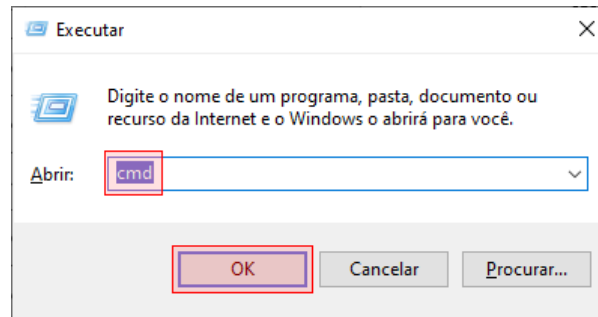


Figura 11: Abrindo o Prompt de Comando (`cmd`) via janela "Executar".

É provável que o *prompt* comece na pasta do seu usuário do Windows. Na Figura 12, podemos observar que o *prompt* da minha máquina começa em `C:\Users\andre`.

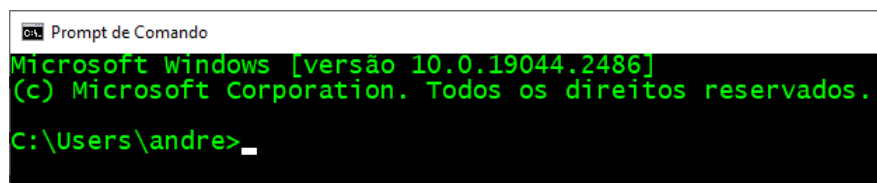


Figura 12: Prompt de Comando logo após aberto.

No prompt, digite `javac --version` e aperte a tecla ENTER. Caso o JDK tenha sido configurado corretamente, o compilador `javac` deve exibir sua versão. Observe na Figura 13 que o compilador `javac` reconhecido pelo sistema operacional é a versão 17.0.6.

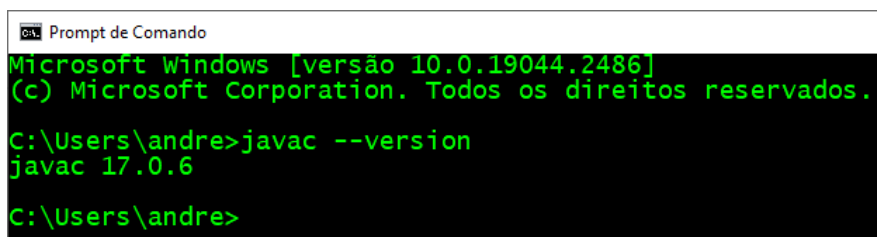


Figura 13: Compilador `javac` executado corretamente.

Na sequência, digite `java --version` e aperte ENTER (observe que agora estamos executando o `java` e não o compilador `javac`). Novamente, caso o JDK tenha sido configurado corretamente, devemos obter a versão do Java na saída do *prompt* (Figura 14).

<sup>3</sup> Ou simplesmente *prompt* – termo que será usado neste material a partir deste ponto para fazer referência ao *Prompt de Comando* do Windows. Também conhecido por *linha de comando*, *terminal* e/ou *console*.

```

C:\> Prompt de Comando

Microsoft Windows [versão 10.0.19044.2486]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\andre>javac --version
javac 17.0.6

C:\Users\andre>java --version
java 17.0.6 2023-01-17 LTS
Java(TM) SE Runtime Environment (build 17.0.6+9-LTS-190)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.6+9-LTS-190, mixed mode, sharing)

C:\Users\andre>
```

Figura 14: `java` executado corretamente.

Para fechar o *prompt*, digite `exit` e aperte ENTER.

## SAIBA MAIS

Caso o JDK não tenha sido configurado corretamente, você receberá uma mensagem de erro ao tentar executar o `javac` e/ou `java`. Veja a Figura 15:

```

C:\> Prompt de Comando

Microsoft Windows [versão 10.0.19044.2486]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\andre>javac --version
'javac' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.

C:\Users\andre>java --version
'java' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.

C:\Users\andre>_
```

Figura 15: Erro quando o JDK não está configurado corretamente.

Um problema de configuração que pode acontecer é errar algum caractere do `JAVA_HOME` e/ou `PATH`, ou mesmo indicar um local incorreto para o `JAVA_HOME` (isto é, um local onde o JDK não existe).

No primeiro caso, um exemplo de erro é definir o valor `JAVA_HOME\bin` na variável `PATH` – observe que esqueci de colocar `%` antes e depois de `JAVA_HOME` (o correto é `%JAVA_HOME%\bin`).

Um exemplo para o segundo caso: descompactei o JDK em `D:\dev\tools\jdk-17`, mas defini que `JAVA_HOME=C:\Java` (uma pasta que não existe na minha máquina).

Em qualquer uma dessas situações, obtemos erros como a Figura 15 ao tentar executar `javac`, `java` ou qualquer outro componente do JDK.

Chegou até aqui? Parabéns! Você configurou o JDK e agora podemos começar a desenvolver programas em Java. Obviamente, nosso primeiro programa será o clássico *Hello, World!*.

## Hello, World! em Java

Antes de começarmos a usar o IDE Eclipse, vamos escrever o *Hello, World!* e aproveitar esse primeiro exemplo para entender como compilar e executar um código Java via linha de comando (*prompt*).

### Um pouco sobre o *prompt*

Abra novamente o *prompt*, caso o tenha fechado. Também é possível abrir mais de um *prompt* ao mesmo tempo.

#### ATENÇÃO!

Os próximos passos que realizaremos no *prompt* vão depender da sua máquina e dos nomes e locais do computador que você decidir usar.

No exemplo deste material, será usada a unidade **D:** da máquina e assumimos que o JDK 17 está localizado em **D:\dev\tools\jdk-17** e o Eclipse – que veremos nas próximas páginas – está localizado em **D:\dev\tools\eclipse**.

Vamos criar uma pasta onde colocaremos o arquivo contendo nosso código Java. Você pode fazer isso pelo *Windows Explorer* (*Explorador de Arquivos*), mas como estamos com o *prompt* aberto, faremos tudo por linha de comando.

Conforme indicado na caixa *Atenção!* acima, o exemplo deste material será realizado na unidade **D:** da máquina. Portanto, como o *prompt* é iniciado na pasta do usuário do Windows (**C:\Users\NomeDoUsuario**), precisamos alterar da unidade **C:** para a **D:**.

Se este também for o seu caso, no *prompt*, digite o comando abaixo para fazer com que o local atual do *prompt* mude para a unidade **D:**.

```
> D: <enter>
```

#### SAIBA MAIS

Três detalhes importantes sobre como os comandos do *prompt* estão escritos neste material:

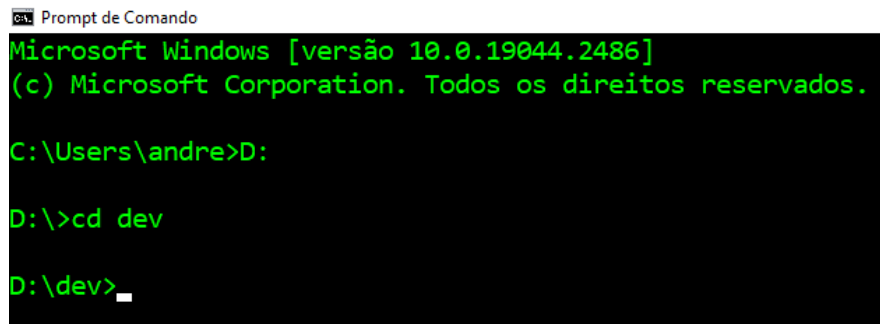
1. Considere que o primeiro caractere **>** não precisa ser inserido. Observe o seu *prompt* – ele sempre mostra um caminho e termina com o caractere **>**, não é? Por exemplo, **C:\Users\andre>** indica que o local atual do *prompt* é a unidade **C:**, na pasta **Users** e, dentro desta pasta, estamos na pasta **andre**.
2. O **<enter>** significa que você deve apertar a tecla ENTER.
3. Não existe um espaço em branco antes do **<enter>** (o espaço em branco só existe no texto para que os comandos fiquem mais legíveis).

Ou seja: no comando anterior, você só deve digitar a letra **D**, seguida dos dois pontos e apertar a tecla ENTER.

Na sequência, como a máquina de exemplo possui a pasta **dev** na raiz da unidade **D:**, podemos navegar dentro dessa pasta com o comando **cd**:

```
> cd dev <enter>
```

A Figura 16 mostra o *prompt* até esse momento.



```
Prompt de Comando
Microsoft Windows [versão 10.0.19044.2486]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\andre>D:

D:\>cd dev

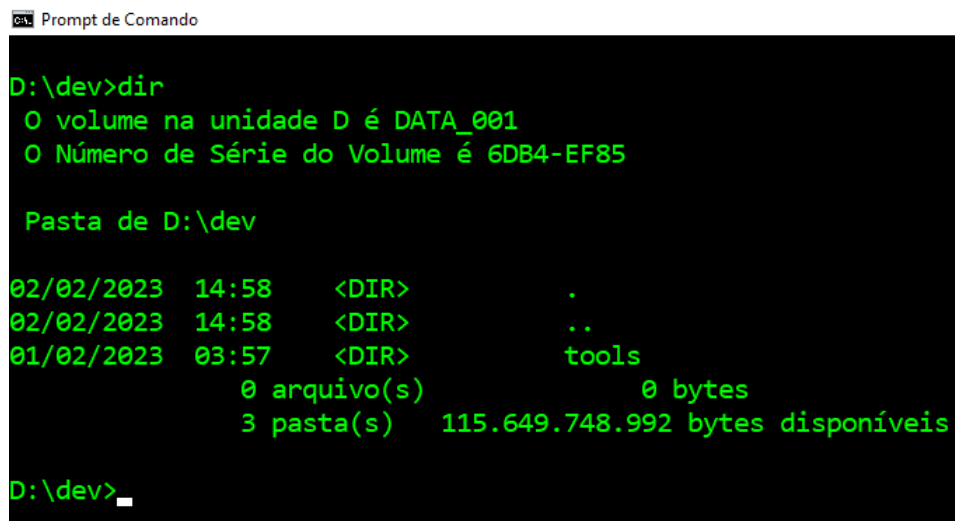
D:\dev>
```

Figura 16: Navegando pelas pastas do computador via *prompt*.

Para visualizar o conteúdo (pastas e arquivos) do local atual do *prompt*, usamos o comando **dir**:

```
> dir <enter>
```

A saída do comando **dir** da máquina de exemplo pode ser vista na Figura 17.



```
Prompt de Comando

D:\dev>dir
O volume na unidade D é DATA_001
O Número de Série do Volume é 6DB4-EF85

Pasta de D:\dev

02/02/2023  14:58    <DIR>          .
02/02/2023  14:58    <DIR>          ..
01/02/2023  03:57    <DIR>          tools
                0 arquivo(s)                0 bytes
                3 pasta(s) 115.649.748.992 bytes disponíveis

D:\dev>
```

Figura 17: Conteúdo da pasta **D:\dev**.

Nesse momento, a pasta **dev** contém apenas a pasta **tools** (veja que **tools** está indicado como **<DIR>**) e nenhum arquivo.

Há outros dois nomes que estão indicados como **<DIR>** também: um ponto (**.**) e um ponto-ponto (**..**). Eles não são pastas, mas símbolos especiais para navegação e localização.

Um único ponto (.) indica o local atual do *prompt* e o ponto-ponto (..) indica um nível anterior/acima do local atual do *prompt*.

Nesse caso, o . é o mesmo que D:\dev e o .. aponta para D:\ (a raiz da unidade D:).

### SAIBA MAIS

Esse conceito de ponto (.) e ponto-ponto (..) é bastante usado quando trabalhamos com caminhos relativos ao invés de caminhos absolutos.

Um caminho absoluto é definido desde a unidade de disco até a pasta ou arquivo que desejamos indicar, por exemplo: D:\dev\tools.

Um caminho relativo trabalha com relação à localização atual do *prompt*. Por exemplo, considerando que já estamos em D:\dev:

O caminho relativo .\tools ficaria D:\dev\.\tools e equivale ao caminho absoluto D:\dev\tools.

O caminho relativo ../tmp ficaria D:\dev\..\tmp e equivale ao caminho absoluto D:\tmp.

Caminhos relativos e absolutos também funcionam com URLs e HTML. Por exemplo, a tag  tenta carregar o arquivo icon.png que está localizado um nível antes/acima da URL atual.

Por exemplo, após executar o seguinte comando:

```
> cd . <enter>
```

Nada acontece, isto é, continuamos no mesmo local atual do *prompt*. Porém, com o comando:

```
> cd .. <enter>
```

Saímos da pasta dev e navegamos de volta para a raiz da unidade D:. Veja o resultado na Figura 18.



```
CA Prompt de Comando
D:\dev>cd .
D:\dev>cd ..
D:\>_
```

Figura 18: Diferença entre . e .. no *prompt*.



## SAIBA MAIS

De maneira bem superficial e sem entrar em detalhes de implementação, a palavra `dir` vem de *directory* (diretório), que é o nome usado antes de “pasta” (*folder*) virar um termo comum em Computação. Neste material, diretório e pasta podem aparecer de forma intercambiável.

Agora que o ponto (.) e o ponto-ponto (..) estão explicados, vamos voltar para dentro do diretório `D:\dev` e criar um subdiretório (uma nova pasta dentro de `D:\dev`).

## Criando um diretório/pasta

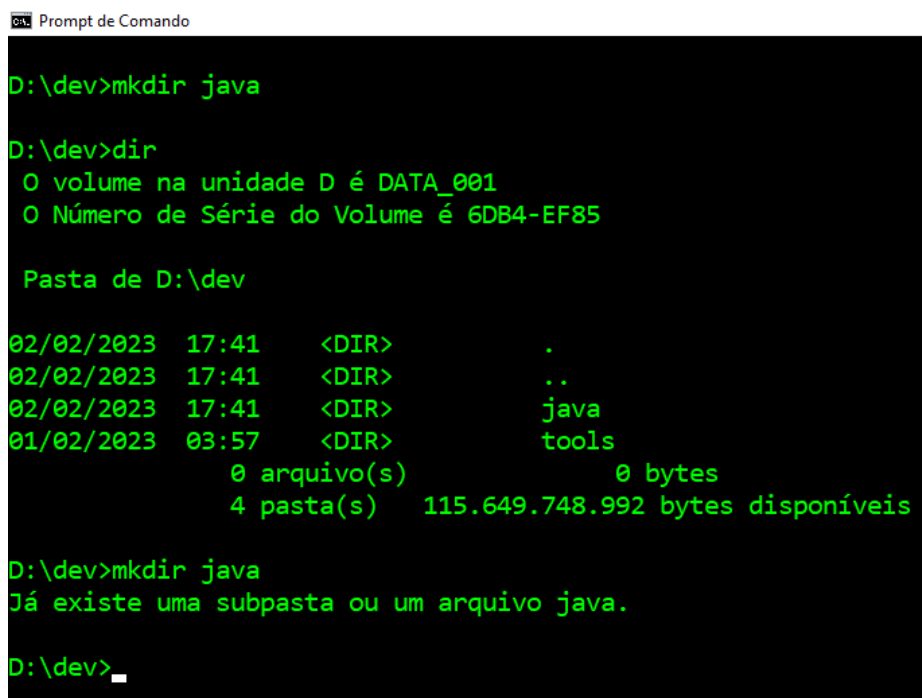
Para criar uma pasta no computador, usamos o comando `mkdir`. Assumindo que estamos no local correto onde a nova pasta será criada (no exemplo, `D:\dev`), inserimos o seguinte comando para criar uma pasta chamada `java`:

```
> mkdir java <enter>
```

O comando não exibe nenhuma mensagem, a não ser que tenha ocorrido algum erro. Para verificar se a pasta `java` foi criada, use o comando `dir`.

Na Figura 19, podemos ver a execução do `mkdir` com sucesso (sem nenhuma mensagem de erro). Na sequência, usamos o comando `dir` para listar o conteúdo de `D:\dev`. Observe que agora existe um novo diretório chamado `java`.

Em seguida, tentamos criar novamente a mesma pasta, mas o `mkdir` exibe uma mensagem de erro, informando que uma subpasta ou arquivo com o nome `java` já existe no local.



```

Prompt de Comando

D:\dev>mkdir java

D:\dev>dir
O volume na unidade D é DATA_001
O Número de Série do Volume é 6DB4-EF85

Pasta de D:\dev

02/02/2023  17:41    <DIR>          .
02/02/2023  17:41    <DIR>          ..
02/02/2023  17:41    <DIR>          java
01/02/2023  03:57    <DIR>          tools
                0 arquivo(s)                0 bytes
                4 pasta(s)  115.649.748.992 bytes disponíveis

D:\dev>mkdir java
Já existe uma subpasta ou um arquivo java.

D:\dev>
```

Figura 19: Criando o subdiretório `java` dentro de `dev`.

Com o diretório `java` criado, vamos navegar dentro dele:

```
> cd java <enter>
```

### PARA PENSAR, PRATICAR E EXPLORAR...

Já usamos alguns comandos do *prompt* do Windows, mas não vimos o significado dos nomes de cada comando. Os comandos usados até o momento foram:

- `cd`: *change directory* (mudar de diretório);
- `dir`: *directory listing* (listar o diretório);
- `mkdir`: *make directory* (criar diretório).

Outro comando muito comum é o `cls` (*clear screen*, limpar a tela). Quer explorar mais comandos do *prompt*? Então use o comando `help` para o *prompt* exibir uma lista de comandos. Para cada comando, podemos usar `help <comando>` ou `<comando> /?` para obter mais informações sobre o `<comando>` específico. Por exemplo:

```
> help dir <enter>  
> dir /? <enter>
```

Ambos os comandos exibem o texto de ajuda sobre o comando `dir`. Ao ler o texto de ajuda, descobrimos que usar o parâmetro `/s` faz com que `dir` liste todo o conteúdo do local atual do *prompt*, incluindo subdiretórios, e que `/p` inclui uma pausa quando a exibição do `dir` ultrapassa o limite da tela do *prompt*.

## Criando o arquivo Hello.java

No começo deste material, na seção *Ferramentas*, foi dito que para programar em Java só precisamos do JDK e de um editor de texto simples.

Toda instalação do Windows já vem com um editor de texto simples: o *Bloco de Notas* (*Notepad*). Apesar de não ter funcionalidades como as ferramentas voltadas para programação, o *Notepad* será suficiente para escrevermos o código do *Hello, World!*.

Assumindo que estamos no *prompt* e no diretório `D:\dev\java`, podemos abrir o *Notepad* usando o seguinte comando:

```
> notepad Hello.java <enter>
```

Esse comando executa o *Notepad* (rodando o arquivo executável `notepad.exe`) e passa como parâmetro o arquivo que queremos editar (no caso, `Hello.java`).

## SAIBA MAIS

A extensão dos arquivos de texto que contêm o código-fonte de um programa Java é `.java`.

## ATENÇÃO!

A linguagem Java é *case-sensitive*, ou seja, é sensível a letras maiúsculas e minúsculas, incluindo os nomes de arquivos.

Portanto, Java considera o arquivo `Hello.java` (com H maiúsculo) diferente de `hello.java` (com h minúsculo).

O Windows executa o *Notepad*, mas como o arquivo `Hello.java` não existe, o *Notepad* exibe a janela *pop-up* da Figura 20 assim que é aberto.

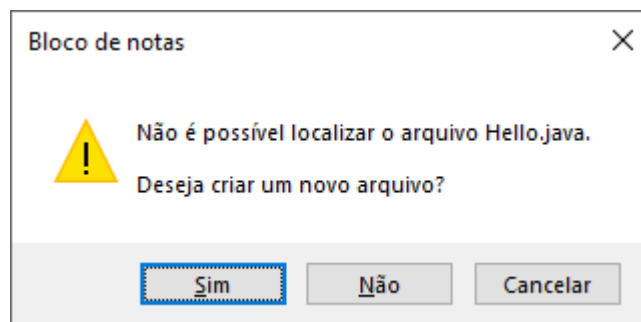


Figura 20: Criando o arquivo `Hello.java` no *Notepad*.

Clique no botão “*Sim*” para o *Notepad* criar o arquivo `Hello.java`.

→ *Espera um pouco... Onde que esse arquivo `Hello.java` é criado?*

Boa pergunta! Como executamos o *Notepad* a partir do caminho `D:\dev\java` e passamos `Hello.java` como parâmetro, o programa entende que o arquivo `Hello.java` está no local onde ele foi chamado. Então, nesse caso, o *Notepad* cria o arquivo `Hello.java` em `D:\dev\java`, como mostra a Figura 21.



```
C:\ Prompt de Comando

D:\dev\java>notepad Hello.java

D:\dev\java>dir
O volume na unidade D é DATA_001
O Número de Série do Volume é 6DB4-EF85

Pasta de D:\dev\java

02/02/2023  20:14    <DIR>          .
02/02/2023  20:14    <DIR>          ..
02/02/2023  20:14                0 Hello.java
               1 arquivo(s)                0 bytes
               2 pasta(s)  115.649.748.992 bytes disponíveis

D:\dev\java>
```

Figura 21: Arquivo `Hello.java` criado pelo *Notepad* na pasta `D:\dev\java`.

Observe que há um arquivo `Hello.java` com 0 bytes em `D:\dev\java`, exatamente o arquivo que foi criado pelo *Notepad* (o arquivo possui 0 bytes pois não possui conteúdo).

### PARA PENSAR, PRATICAR E EXPLORAR...

Há algumas maneiras de criar um arquivo vazio pelo *prompt*, como os três exemplos a seguir (*arquivo* é o nome do arquivo que você quer criar e *ext* a extensão do arquivo):

```
> copy nul arquivo.ext <enter>
> type nul > arquivo.ext <enter>
> echo off > arquivo.ext | echo on <enter>
```

Curioso? Faça uma pesquisa sobre esses comandos para entender por que eles criam um arquivo vazio! Uma referência muito boa, que inclui comandos de outros sistemas operacionais, pode ser conferida em [4].

De volta ao *Notepad*, digite o seguinte código no arquivo `Hello.java` (Listagem 1), lembrando que a linguagem Java diferencia letras maiúsculas e minúsculas.

```
01 class Hello {
02     public static void main(String[] args) {
03         System.out.println("Hello, World!");
04     }
05 }
```

Listagem 1: `Hello.java`.

Em seguida, salve o arquivo e volte ao *prompt*.

## Compilando o arquivo Hello.java

Agora que o arquivo `Hello.java` possui o código da Listagem 1 e estamos no *prompt*, vamos compilar o código usando o compilador Java (`javac`), com o seguinte comando:

```
> javac Hello.java <enter>
```

Caso o código esteja livre de erros, não haverá nenhuma saída (em tela) do compilador `javac`, mas um arquivo chamado `Hello.class` será criado no mesmo local do `Hello.java`, como na Figura 22.



```
Prompt de Comando

D:\dev\java>javac Hello.java

D:\dev\java>dir
O volume na unidade D é DATA_001
O Número de Série do Volume é 6DB4-EF85

Pasta de D:\dev\java

02/02/2023  22:57    <DIR>          .
02/02/2023  22:57    <DIR>          ..
02/02/2023  22:57                417 Hello.class
02/02/2023  22:57                110 Hello.java
                2 arquivo(s)                527 bytes
                2 pasta(s)  115.649.748.992 bytes disponíveis

D:\dev\java>_
```

Figura 22: Arquivo `Hello.class` após compilação.

### SAIBA MAIS

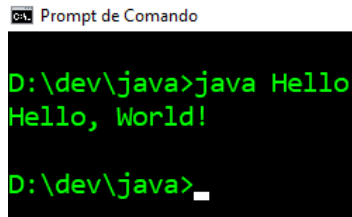
O arquivo com extensão `.class` contém o que é conhecido como *Java bytecode*, um conjunto de instruções usado pela JVM para executar os programas escritos em Java.

## Executando o programa Hello(.class)

Uma vez que temos o código-fonte Java compilado e transformado em um arquivo `.class`, podemos executar o nosso programa usando o comando a seguir:

```
> java Hello <enter>
```

A saída do programa `Hello` é exibida na Figura 23.



```
Prompt de Comando

D:\dev\java>java Hello
Hello, World!

D:\dev\java>
```

Figura 23: Executando o programa `Hello`.

Observe que, no caso do `java`, não devemos incluir a extensão `.class` do arquivo, diferente do `javac` que exige que os arquivos Java a serem compilados devem ter a extensão `.java` inclusa no nome do arquivo.

### PARA PENSAR, PRATICAR E EXPLORAR...

É importante entendermos que o JDK possui as ferramentas `javac` e `java` que processam os arquivos `.java` e `.class`, respectivamente, e que não precisamos usar um IDE para escrever, compilar e executar programas em Java.

Entretanto, no dia a dia, dificilmente as pessoas que trabalham em projetos Java fazem a compilação do código-fonte dos projetos chamando o `javac` diretamente no *prompt*. O mais usual é usar uma ferramenta que gerencia o processo de compilação (também conhecido como “processo de *build*”), como o Apache Ant [5], Apache Maven [6] e Gradle Build Tool [7].

O uso desse tipo de ferramenta está fora do escopo deste material e da disciplina, mas para quem tiver interesse em conhecer mais e se aprofundar no assunto, recomendo consultar as referências indicadas em cada ferramenta.

## Entendendo o código do `Hello.java`

O código que escrevemos para o nosso *Hello, World!* em Java é um dos menores programas Java válidos que podemos escrever<sup>4</sup>. Apesar disso, o exemplo já nos mostra informações importantes de um programa escrito nessa linguagem.

Java é uma **linguagem de programação orientada a objetos** (comumente abreviada como POO ou OOP em inglês, de *object-oriented programming*).

POO é um paradigma de programação baseado no conceito de “objetos”, isto é, implementamos um sistema a partir de objetos que possuem dados e funções, sendo que objetos podem ser/representar qualquer coisa (real ou abstrata) e podem interagir entre si (de acordo com as especificações do sistema)

Outra maneira que podemos pensar em objetos é considerar que eles possuem propriedades e comportamentos/ações. As propriedades (dados) definem o estado atual

<sup>4</sup> A alteração para criar um código Java válido menor do que o nosso *Hello, World!* é remover a instrução `System.out.println("Hello, World!");`.

do objeto (valores dos dados em um determinado momento) e o estado atual pode ser alterado a partir das ações do objeto (funções que alteram valores de uma ou mais propriedades do objeto).

A maioria das linguagens de programação orientada a objetos são baseadas em classes, e Java é uma delas.

### SAIBA MAIS

**Classe** é uma definição de um objeto, isto é, uma classe define a estrutura/esqueleto (*blueprint*) e comportamento dos objetos que são de um determinado tipo (sendo que o tipo é definido pelo nome da classe).

Por conta disso, em Java, tudo<sup>5</sup> é definido dentro de uma classe e as classes são definidas com a palavra-chave `class`, seguida do nome da classe (que é escolhida por quem estiver escrevendo o código). O conteúdo de uma classe é delimitada pelos símbolos abre e fecha chaves `{ }`.

Por conveniência, vamos listar novamente o código do arquivo `Hello.java` (Listagem 2).

```
01 class Hello {  
02     public static void main(String[] args) {  
03         System.out.println("Hello, World!");  
04     }  
05 }
```

Listagem 2: `Hello.java` (novamente).

Observe que nosso código tem uma classe chamada `Hello` (linha 1). Dentro dessa classe `Hello`, após o abre-chaves `{`, temos um método chamado `main` que recebe como parâmetro um vetor/array de string, indicado por `String[] args`, e que não retorna valor, indicado pela palavra-chave `void` antes de `main` (linha 2).

### SAIBA MAIS

**Método** é um sinônimo para algo que já conhecemos em programação como “função”. Em POO, as funções que existem dentro de uma classe são chamadas de métodos. Obviamente, se alguém usar a palavra *função* no lugar de *método*, as pessoas entenderão sobre o que está sendo falado.

O método `main()` é o **ponto de partida** do nosso programa Java e é um método especial usado pela JVM.

---

<sup>5</sup> Nem tudo, na verdade: podemos definir `interfaces`, `enums` e `records` fora de uma classe. Mas esses são assuntos que fogem do escopo deste material.

A assinatura da `main()` deve ser exatamente como está escrito na linha 2 da Listagem 2: `public static void main(String[] args)`<sup>6</sup>.

Caso o programa não tenha esse método dentro de alguma classe ou o método `main()` não siga essa assinatura, o compilador `javac` até consegue compilar o código, mas a JVM não conseguirá executar o programa.

### SAIBA MAIS

Nesse momento alguns detalhes podem parecer complicados e não fazer muito sentido, mas aos poucos, no decorrer das aulas, vamos entendendo melhor o significado completo da assinatura da `main()`.

Por enquanto, basta ter em mente que, quando executamos um programa Java, a JVM procura por esse método `main()` específico para iniciar a execução do programa.

A implementação de um método também é delimitado pelos símbolos abre e fecha chaves `{ }`. Ou seja, todo o código que pertence a um método deve estar entre esses dois símbolos. Observe que o abre-chaves `{` do método `main()` aparece logo após a assinatura do método, no final da linha 2, e o fecha-chaves `}` aparece na linha 4. Já o fecha-chaves `}` que aparece na linha 5 indica o fechamento da classe `Hello`.

Por fim, na linha 3 da Listagem 2, temos a instrução `System.out.println("Hello, World!");`. Pela saída do programa e talvez pelo nome da instrução, você provavelmente já sabe o que essa instrução faz, certo?

Nessa instrução, estamos usando a classe `System` oferecida pela API do Java. Essa classe possui um objeto chamado `out`, que fornece métodos para enviar dados para a saída do sistema (no nosso caso, o *prompt*). Um dos métodos é o `println()` que recebe uma string como parâmetro: a string passada como parâmetro é exibida no *prompt* e `println()` adiciona uma quebra de linha logo depois de exibir a string.

### ATENÇÃO!

Algo importante para não esquecer! Observe que, após a instrução `System.out.println()`, adicionamos um ponto-e-vírgula `;`.

Em Java, o ponto-e-vírgula `;` é usado para indicar o fim de uma instrução. Já um bloco de instruções é agrupado/delimitado pelos símbolos abre e fecha chaves `{ }`, não sendo necessário incluir o ponto-e-vírgula `;` após o fecha chaves `}`.

<sup>6</sup> Com exceção do nome do parâmetro `args`, que pode ser outro de sua preferência. E você também pode encontrar a seguinte variação da assinatura da `main()`, válida a partir do Java 5:

```
public static void main(String... args)
```

As reticências `...` após `String` indicam que a `main()` pode receber um array de string ou uma sequência de parâmetros do tipo string.



E, sem entrar em alguns detalhes da linguagem Java, esse é o nosso código do *Hello, World!*

Se você chegou até aqui depois de seguir todos os passos dessa seção *Hello, World! em Java*, meus parabéns! Você passou pelo processo inicial do Java e tenho certeza de que está pronto(a) para começar a usar o Eclipse e a desenvolver seus próprios programas em Java!

## Eclipse IDE

O site oficial para baixar o Eclipse é <https://www.eclipse.org> [8]. Na página inicial, clique no botão “Download”. Atualmente, esse link encontra-se no campo superior direito do site, conforme destacado na Figura 24.

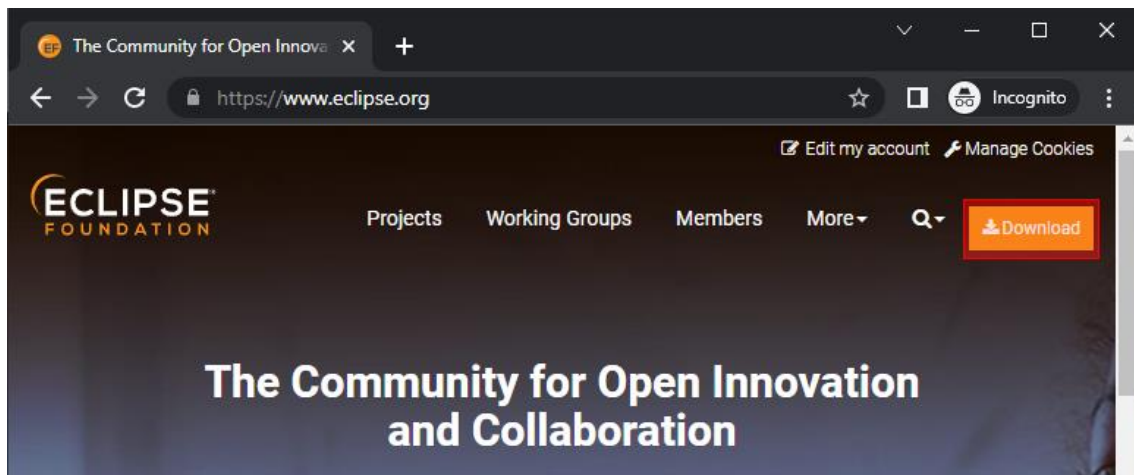


Figura 24: Página inicial do <https://www.eclipse.org>.

O link de download nos leva para a URL <https://www.eclipse.org/downloads/>, que lista um conjunto de ferramentas disponíveis pela Eclipse Foundation. Clique no link “Download Packages” do Eclipse IDE (em destaque na Figura 25).

O link “Download Packages” abre a URL <https://www.eclipse.org/downloads/packages/>. Nessa página, podemos baixar diferentes versões do Eclipse que são específicas para desenvolvimento Java, C/C+, PHP, entre outros.

A versão que nos interessa é a **Eclipse IDE for Java Developers**. Porém, vamos ignorar o início da página, que destaca o download do *Eclipse Installer*.

Assim como fizemos com o JDK, vamos baixar o Eclipse no formato zip, para não haver a necessidade de uma conta de administrador para instalar a ferramenta.

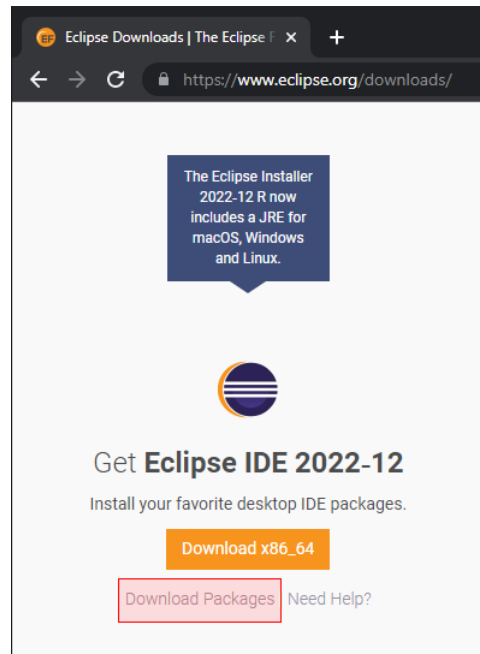


Figura 25: Vamos baixar o Eclipse via link "Download Packages".

Na página de download dos *Eclipse Packages*, localize o Eclipse IDE for Java Developers e clique no link “x86\_64”<sup>7</sup>. Na Figura 26, o link “x86\_64” para Windows está destacado. Caso você use Linux ou macOS, use os links que aparecem logo abaixo.



Figura 26: Download do Eclipse IDE for Java Developers no formato zip.

Ao clicar no link “x86\_64”, somos enviados para outra página, onde é possível selecionar o servidor de onde vamos baixar o Eclipse. Essa página já nos indica um servidor mais próximo de onde estamos localizados (no exemplo da Figura 27, o servidor é do Brasil).

Nessa página, clique no botão “Download”. O link vai para uma página de agradecimentos e doação (opcional) para o Eclipse Foundation e o download do arquivo zip finalmente começará.

<sup>7</sup> **Atualização 07/02/2024:** o link Windows x86\_64 da página de download de *Eclipse Packages* agora aponta para o Eclipse for Java Developers versão 2023-12 R.

**Atualização 03/08/2023:** o link Windows x86\_64 da página de download de *Eclipse Packages* agora aponta para o Eclipse for Java Developers versão 2023-06 R. Assim como a versão do JDK, este material ainda possui textos e imagens fazendo referência ao Eclipse 2022-12 R, mas o conteúdo continua válido para a versão 2023-06 R.

Neste momento (fevereiro/2023), o link Windows x86\_64 da página de download de *Eclipse Packages* aponta para a URL a seguir, que é o Eclipse for Java Developers versão 2022-12 R:

[https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2022-12/R/eclipse-java-2022-12-R-win32-x86\\_64.zip](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2022-12/R/eclipse-java-2022-12-R-win32-x86_64.zip)

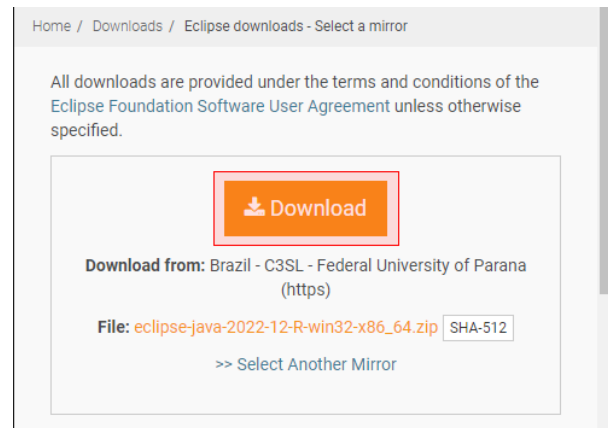


Figura 27: Botão “Download” para baixar o Eclipse IDE.

→ ***OK, consegui baixar o arquivo `eclipse-java-2022-12-R-win32-x86_64.zip`<sup>8</sup>. Tenho que descompactar esse arquivo, né?***

Isso mesmo! Da mesma maneira que fizemos com o JDK, descompacte o arquivo no local de sua preferência (por exemplo, `C:\eclipse`).

No exemplo deste material, o arquivo zip do Eclipse foi descompactado no drive `D:` da máquina, dentro da pasta `\dev\tools` (Figura 28).

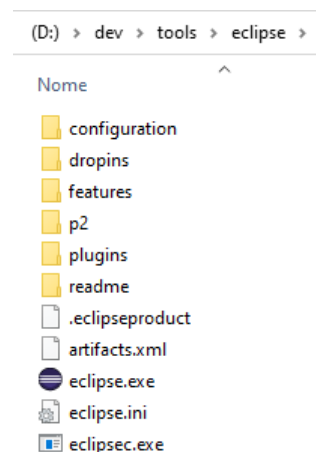


Figura 28: Conteúdo do arquivo zip do Eclipse IDE for Java Developers.

→ ***Olhando o conteúdo do arquivo zip do Eclipse, podemos abrir o IDE com um duplo-clique no arquivo `eclipse.exe`, certo?***

Perfeito! Ao abrir o arquivo `eclipse.exe`, iniciamos o Eclipse – a Figura 29 mostra a tela de *splash* do Eclipse.

<sup>8</sup> Ou outro equivalente, dependendo de quando este material está sendo consultado.



Figura 29: Tela de *splash* do Eclipse.

Agora que temos o Eclipse rodando na máquina, vamos reconstruir o exemplo *Hello, World!* no Eclipse.

## Hello, World! no Eclipse

Quando o Eclipse é iniciado, ele pede para escolhermos um diretório do computador que servirá como *workspace* (espaço de trabalho) do IDE, isto é, um local no computador em que o Eclipse armazenará nossas preferências e outras configurações do IDE.

Você pode escolher o local de sua preferência. No exemplo deste material, foi escolhido o diretório `D:\dev\java` que criamos no *prompt*. Após escolher o local do *workspace*, clique no botão “Launch” (Figura 30).

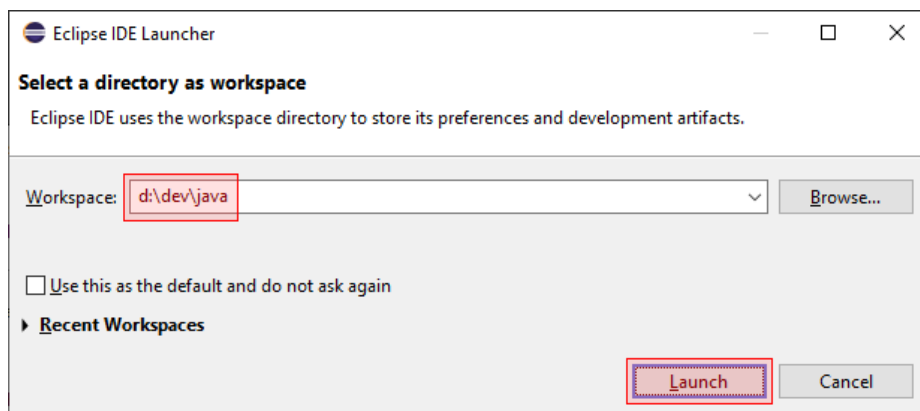


Figura 30: Definindo o *workspace* do Eclipse.

Na sequência, o Eclipse é aberto. Caso seja a sua primeira vez abrindo o Eclipse, o IDE mostrará a aba *Welcome*, como podemos ver na Figura 31.

Nesse momento, você pode fechar a aba *Welcome* para começarmos a desenvolver o nosso *Hello, World!* no Eclipse. Caso queira abrir a aba *Welcome* no futuro, você pode acessar o menu *Help > Welcome*.

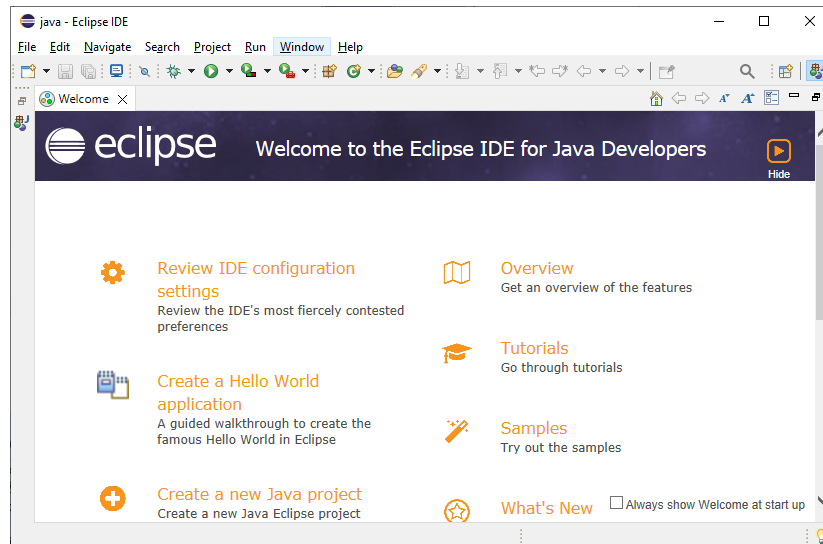


Figura 31: Tela de boas-vindas do Eclipse.

### PARA PENSAR, PRATICAR E EXPLORAR...

Existem diversos links para você explorar na página de boas-vindas – recomendo você reservar um tempo para verificá-los. Inclusive, há um guia para criar um *Hello World* no Eclipse, bem próximo do que faremos aqui.

Ao fechar a aba *Welcome*, nos deparamos com a interface da Figura 32.

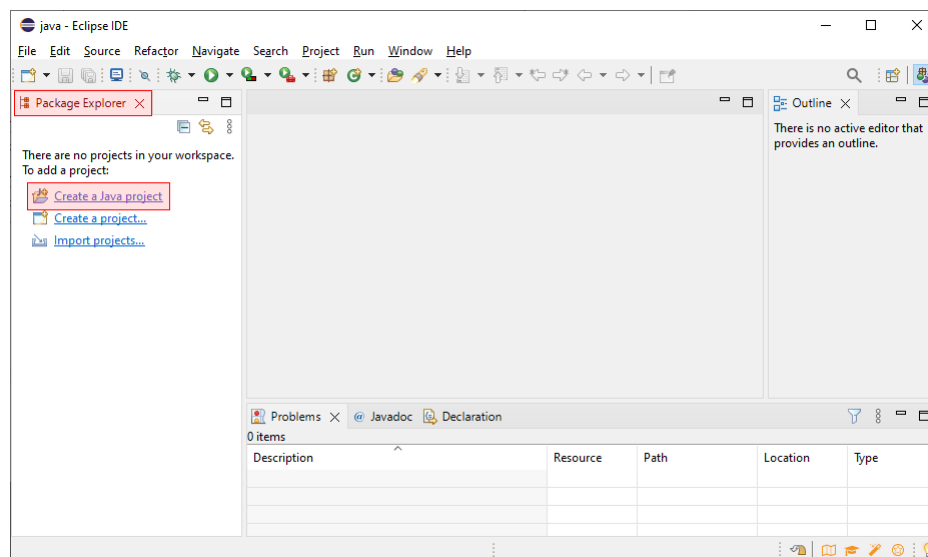


Figura 32: Eclipse IDE.

## Criando um projeto Java

Para começar o nosso *Hello, World!*, devemos criar um projeto Java. Você pode clicar na opção “*Create a Java project*”, na aba “*Package Explorer*” (ambas em destaque na Figura 32), ou acessar o menu *File > New > Java Project*.

Ao fazer isso, o Eclipse nos mostra uma janela pop-up “*New Java Project*”. Nessa janela, devemos preencher o nome do projeto no campo “*Project name:*”. Para esse exemplo, vamos nomear o projeto como `HelloWorld`.

Por enquanto, mantenha todas as outras opções intactas, com exceção do último grupo (“*Module*”): nesse grupo, desative a opção “*Create module-info.java file*”.

As propriedades que foram alteradas na janela “*New Java Project*” estão destacadas na Figura 33.

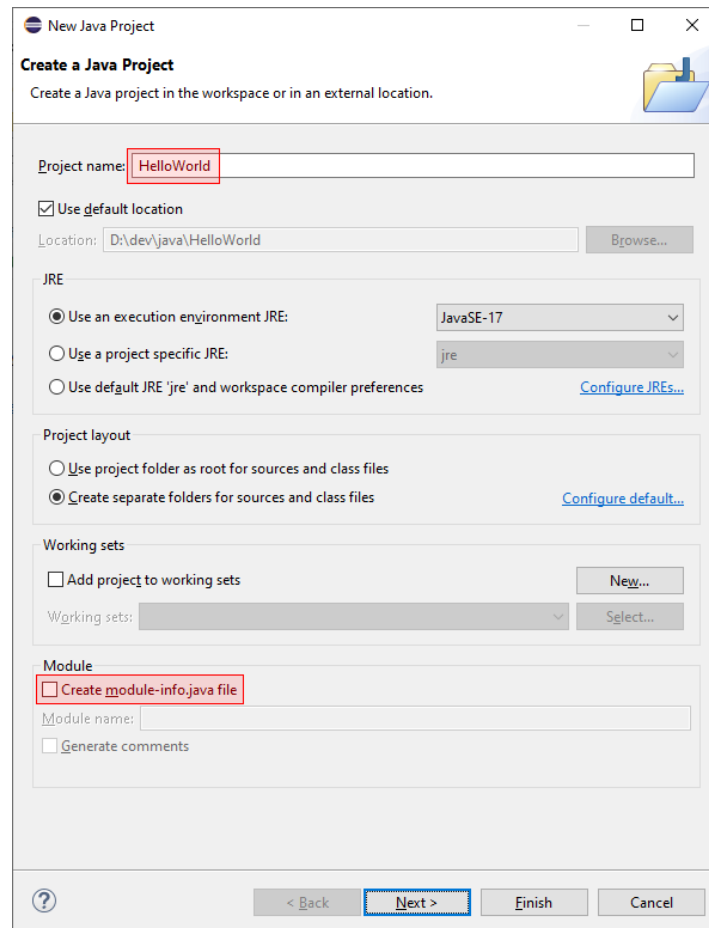


Figura 33: Configuração do novo projeto Java no Eclipse.

Em seguida, podemos clicar diretamente no botão “*Finish*” ou, caso você queira alterar outras configurações do projeto, clique no botão “*Next*”. Para esse exemplo (e outros que desenvolveremos na disciplina), não precisamos configurar nada na janela seguinte. Portanto, podemos clicar no botão “*Finish*”.

Feito isso, o Eclipse cria um projeto Java e agora podemos incluir o nosso arquivo `Hello.java` nesse novo projeto.

## PARA PENSAR, PRATICAR E EXPLORAR...

A partir do Java 9, há uma funcionalidade conhecida como módulos (JPMS – *Java Platform Module Systems*) que permite modularizar códigos Java a fim de melhorar o gerenciamento de dependências de bibliotecas, facilitar a escalabilidade do Java, melhorar segurança, manutenção e desempenho dos programas Java, entre outros.

Na nossa disciplina, o que faremos em termos de organização e modularização de projeto é criar diferentes classes de acordo com as responsabilidades de um sistema, sendo que o tópico de módulos não será incluso nos nossos estudos.

## Criando o arquivo Hello.java

Para criar um arquivo no projeto *HelloWorld*, podemos acessar o menu *File > New > Class* ou clicar com o botão direito no projeto *HelloWorld* ou na pasta *src* (ambas na aba *Package Explorer*) e selecionar *New > Class*, conforme indicado na Figura 34.

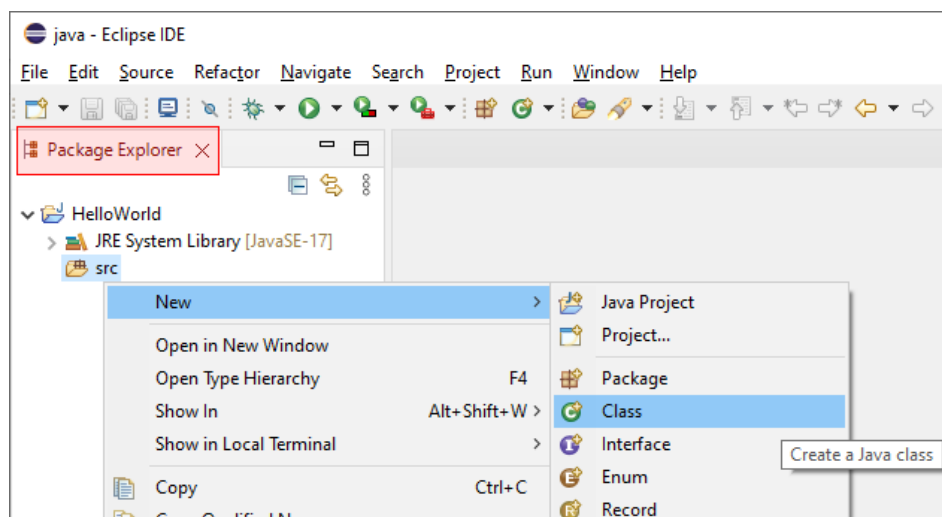


Figura 34: Criando um arquivo `.java` no projeto do Eclipse.

Ao selecionar a opção de criar uma classe, o Eclipse mostra a janela *pop-up* “*New Java Class*” (Figura 35).

Como estamos começando os estudos da linguagem Java agora, essa janela de criação de nova classe pode assustar, pois contém muitas informações que não vimos ainda.

Vamos fazer apenas duas alterações nessa janela: definir o nome da classe no campo “*Name:*” (podemos usar o mesmo nome do nosso primeiro exemplo, `Hello`) e pedir para o Eclipse incluir o método `main()` no arquivo, que pode ser feito ativando o *checkbox* “*public static void main(String[] args)*” do grupo “*Which method stubs would you like to create?*” (uma tradução livre, assumindo como o Eclipse se comporta com as opções dessa pergunta é: “*Quais métodos você quer que já exista no código quando a classe for criada?*”).

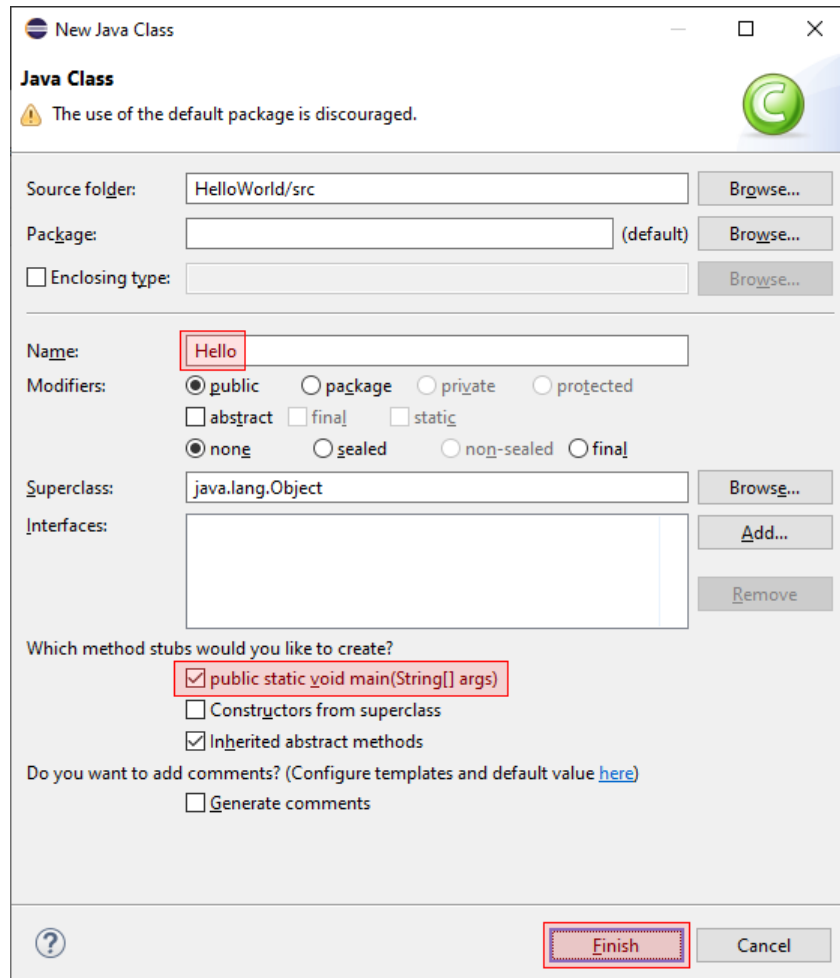


Figura 35: Definindo o nome da classe que contém a `main()`.

Para finalizar a operação, clicamos em “Finish”. O resultado dessa operação é um novo arquivo `Hello.java` adicionado na pasta `src` do projeto, com um conteúdo muito parecido com o que escrevemos na seção *Hello, World! em Java* (Figura 36).

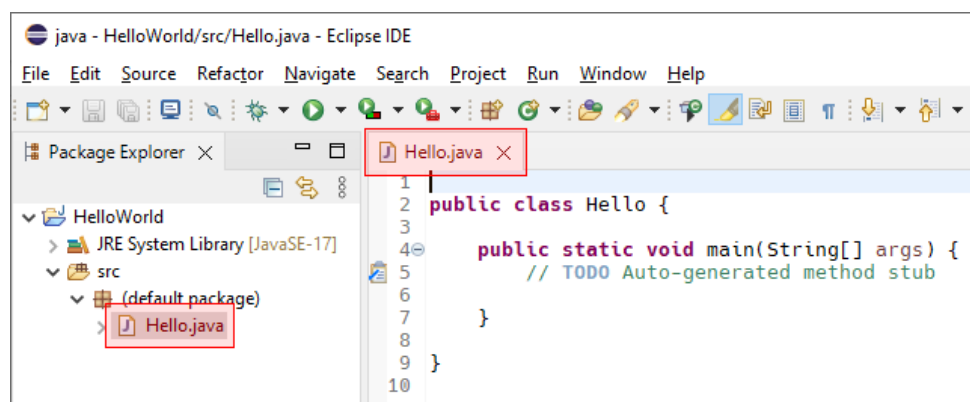


Figura 36: Arquivo `Hello.java` criado e adicionado no projeto.

A linha 5 desse arquivo contém o texto `// TODO Auto-generated method stub`.

Esse é um **comentário de linha única** em Java: tudo que aparece depois do `//` até o final da linha é ignorado pelo compilador (no exemplo da Figura 36, o comentário é apenas um aviso que o método foi gerado automaticamente pelo Eclipse).



## SAIBA MAIS

Além do comentário de linha única, Java possui o **comentário de múltiplas linhas**, que também pode ser usado para comentar trechos em uma mesma linha: o comentário começa com `/*` e termina com `*/`. Por exemplo:

```
/* Esse é um comentário  
de múltiplas linhas */
```

```
public class Hello/*World*/ {
```

No segundo exemplo, a palavra `World` é ignorada, pois está entre `/*` e `*/`.

Com a classe `Hello` criada, acredito que você já saiba o que deve fazer para concluir o código do *Hello, World!* no Eclipse, certo?

→ *Posso apagar a linha 5, que é um comentário, e adicionar a instrução `System.out.println("Hello, World!");`, certo?*

Certíssimo! Com essa alteração, concluímos o código do *Hello, World!* no Eclipse. A versão final do `Hello.java` deve ficar algo parecido com a Figura 37.

## PARA PENSAR, PRATICAR E EXPLORAR...

Observe que o nome do arquivo `.java` é o mesmo que definimos para a nova classe. Existem algumas regras da linguagem Java sobre a criação e nomenclatura de classes e arquivos:

- Um arquivo `.java` só pode conter uma classe pública (marcada com a palavra `public` antes da palavra-chave `class`), mas pode conter mais de uma classe não-pública (embora essa prática não seja recomendada<sup>9</sup>).
- Quando a classe é pública, o nome do arquivo `.java` deve ser idêntico ao nome da classe.

Assim como muitos tópicos da linguagem, estudaremos o significado do modificador `public` em aulas futuras.

Porém, vale a pena fazer o seguinte experimento: altere o nome da `public class Hello` no projeto do Eclipse para `public class Teste` e veja o que acontece.

<sup>9</sup> A prática recomendada e que é seguida pela maioria em projetos Java é que cada classe esteja declarada em seu próprio arquivo `.java`, independente se é pública ou não.

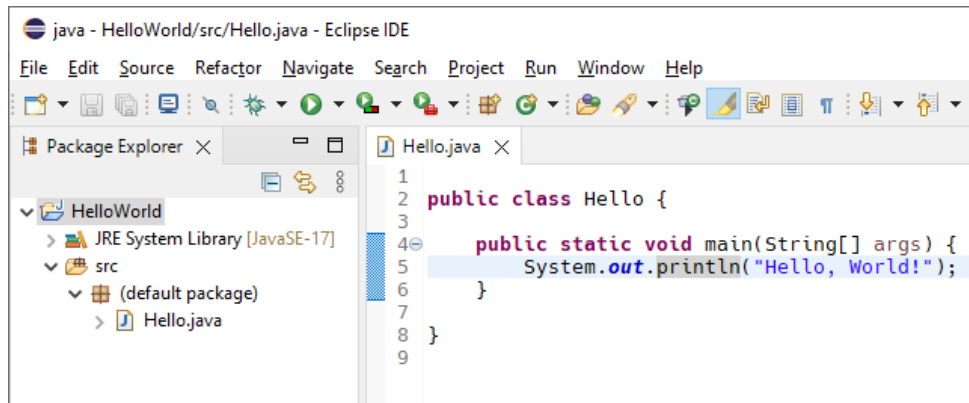


Figura 37: Código final do `Hello.java` no Eclipse.

## Compilando o projeto

Diferente do que vimos com o `javac` via *prompt*, o Eclipse, por padrão, detecta alterações realizadas no código e recompila o que foi alterado no projeto automaticamente. Mas isso só acontece porque a opção *Build Automatically*, sob o menu *Project*, geralmente já está ativada (Figura 38).

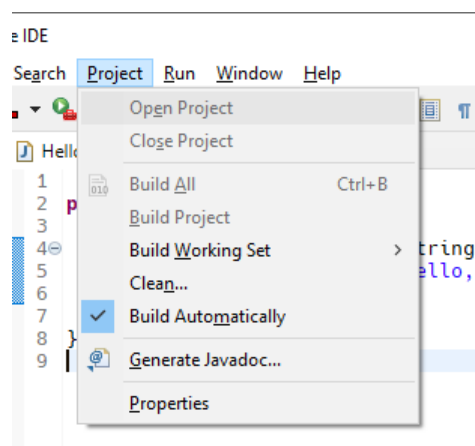


Figura 38: Opção *Build Automatically* do Eclipse.

Caso a opção *Build Automatically* não esteja ativada, você pode compilar o projeto pelo menu *Project > Build Project* (ou *Build All*).

### SAIBA MAIS

Mesmo se o *Build Automatically* estiver desativado e não usarmos a opção *Build Project/All*, o Eclipse automaticamente compila o projeto quando pedimos para ele executar o nosso código.

## Executando o projeto

Para executar o projeto, podemos usar o menu *Run > Run As > Java Application* (Figura 39). A mesma opção pode ser encontrada clicando com o botão direito do mouse no projeto (aba *Package Explorer*).

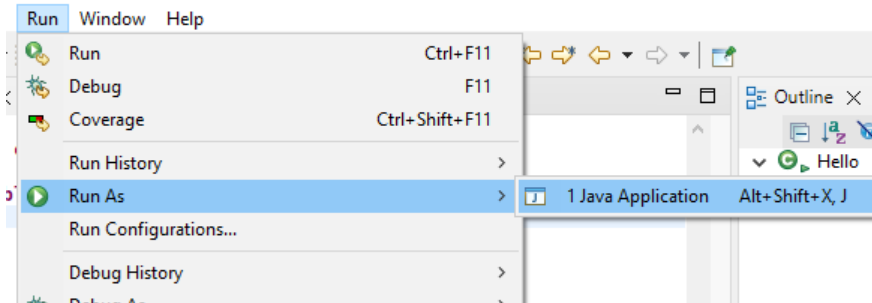


Figura 39: Executando o projeto no Eclipse.

Ao rodar o projeto, o Eclipse mostra qualquer saída do programa (`System.out.println()`) na aba *Console*, geralmente localizada na parte inferior do IDE, como mostra a Figura 40. Como o programa só exibe a mensagem *Hello, World!* e encerra a sua execução ao chegar no final da `main()`, a execução e finalização do programa no Eclipse são tarefas rápidas (para esse projeto).

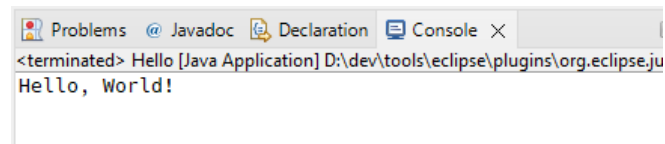


Figura 40: Saída do programa no Eclipse.

E nesse ponto, chegamos ao final do *Hello, World!* no Eclipse. Parabéns por concluir mais uma etapa deste material!

### → *Acabou?*

O exemplo do *Hello, World!* no Eclipse, sim. Mas quero fazer alguns comentários sobre o Eclipse. O primeiro é sobre a estrutura de pastas e arquivos do projeto. Na aba *Package Explorer*, conseguimos ver o `Hello.java`, mas o projeto compilado (convertido em arquivo `.class`) não aparece no Eclipse.

### → *Verdade. Onde que encontro o .class gerado pelo Eclipse?*

Isso vai depender da opção que selecionamos na hora de criar o projeto Java. Volte à Figura 33: *Configuração do novo projeto Java no Eclipse*. Há um grupo chamado “*Project layout*”, cuja opção selecionada é “*Create separate folders for sources and class files*”.

Quando essa opção está selecionada, o padrão das pastas é `src` para código-fonte e `bin` para o código compilado (`.class`). Esse padrão pode ser alterado no link “*Configure default...*”, no mesmo grupo “*Project layout*”.

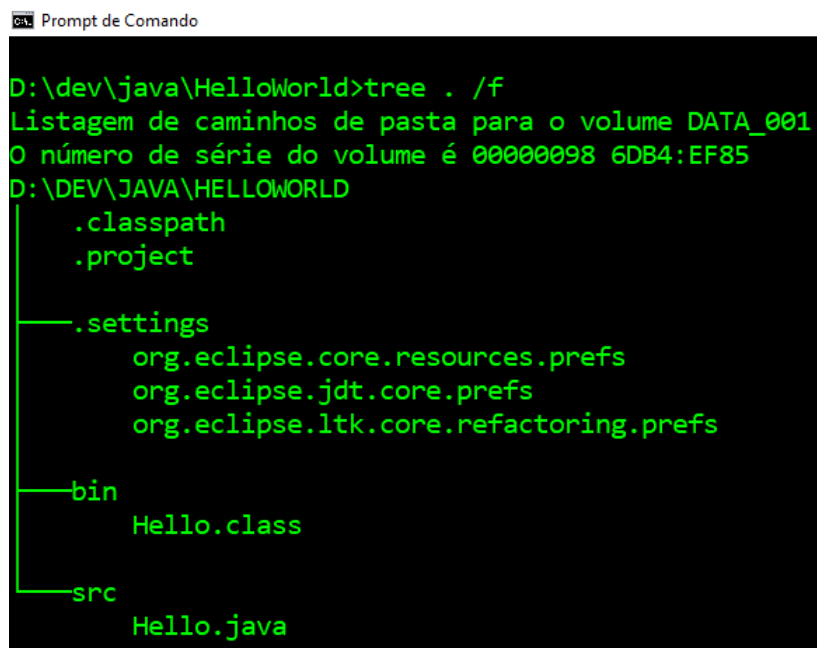
Quando a outra opção está selecionada (“*Use project folder as root for sources and class files*”), tanto os arquivos `.java` quanto os `.class` são salvos na raiz do projeto.

Ao invés de verificar a estrutura de pastas e arquivos do nosso projeto Eclipse no *Windows Explorer*, que tal usar o *prompt*? Caso você tenha seguido os passos deste material, seu projeto Eclipse deve estar na mesma pasta `java` que criamos para o primeiro *Hello, World!*, sendo que o Eclipse deve ter criado uma pasta `HelloWorld` (ou o nome que você escolheu para o projeto) dentro da pasta `java`.

Usando o comando de navegação `cd` no *prompt*, navegue até a raiz do projeto Eclipse. Para listar o conteúdo de um diretório, sabemos que existe o comando `dir`. Mas, ao invés desse comando, vamos usar o comando `tree`, que exibe o conteúdo de um diretório de forma gráfica (ainda que em modo texto):

```
> tree . /f <enter>
```

O ponto `.` após o comando `tree` indica que queremos listar a estrutura do diretório atual e o `/f` é um parâmetro que inclui os arquivos dos diretórios na exibição da estrutura. O resultado desse comando pode ser visto na Figura 41.



```
Prompt de Comando
D:\dev\java\HelloWorld>tree . /f
Listagem de caminhos de pasta para o volume DATA_001
O número de série do volume é 00000098 6DB4:EF85
D:\DEV\JAVA\HELLOWORLD
.
├── .classpath
├── .project
├── .settings
│   ├── org.eclipse.core.resources.prefs
│   ├── org.eclipse.jdt.core.prefs
│   └── org.eclipse.ltk.core.refactoring.prefs
├── bin
│   └── Hello.class
└── src
    └── Hello.java
```

Figura 41: Conteúdo do projeto `HelloWorld` no Eclipse.

Veja que, além do `Hello.java` na pasta `src` e `Hello.class` na pasta `bin`, temos cinco arquivos com configurações do projeto Eclipse.

→ *Interessante. Quais são os outros comentários que você queria fazer sobre o Eclipse?*

Veja a Figura 40: *Saída do programa no Eclipse*. Acima da saída com a mensagem *Hello, World!*, há uma informação indicando “*<terminated> Hello [Java Application]* D:\dev\tools\eclipse\plugins\(...)”.

Se você procurar no Eclipse, perceberá que não estamos usando o JDK 17 que configuramos anteriormente. Mesmo se não configurarmos as variáveis de ambiente `JAVA_HOME` e `PATH`, o Eclipse funciona e consegue compilar nosso código Java.

Curioso, não? Há uma explicação para isso.

### SAIBA MAIS

Reveja a Figura 25, que mostra parte da página em que clicamos no link “*Download Packages*”. Ela possui uma observação logo acima do logo do Eclipse:

***“The Eclipse Installer 2022-12 R now includes a JRE for macOS, Windows and Linux.”***

Pelo que vimos na seção *JDK – Java Development Kit*, o JRE contém a JVM e arquivos necessários para *executar* programas Java, mas precisamos do JDK para *desenvolver* programas em Java.

No entanto, quando você usar o Eclipse, vai perceber que não precisamos configurar o JDK dentro do IDE e, mesmo assim, o Eclipse consegue compilar e executar código Java. Como isso é possível?

Talvez a mensagem no site do Eclipse não seja das melhores: ao ler que o instalador inclui um JRE, parece que mesmo assim precisamos do JDK, já que o JRE é usado para executar programas Java. O que não está explicado na mensagem do site é que o Eclipse também inclui seu próprio compilador Java via plugin **JDT Core** [9].

É por conta desse plugin que o Eclipse consegue compilar código Java sem precisar do JDK. Ainda, seu compilador é incremental, permitindo executar e depurar código que ainda contém erros.

### → *Ué. Então eu nem precisava ter baixado e configurado o JDK?*

Caso você vá usar única e exclusivamente o **Eclipse IDE for Java Developers**, então, sim, você não precisava baixar e configurar o JDK.

Mas é interessante ter o JDK configurado corretamente na máquina – caso queira usar outras ferramentas que não o Eclipse (ou até mesmo usar plug-ins dentro do Eclipse), será necessário ter o JDK devidamente configurado.

### → *Configurei o JDK na máquina, é possível usar um JDK específico no Eclipse?*

Sim! Para isso, abra as preferências do Eclipse pelo menu *Window > Preferences*. Na janela que se abre (“*Preferences*”), vá em *Java > Installed JREs* e clique no botão “*Add...*” (em destaque na Figura 42).

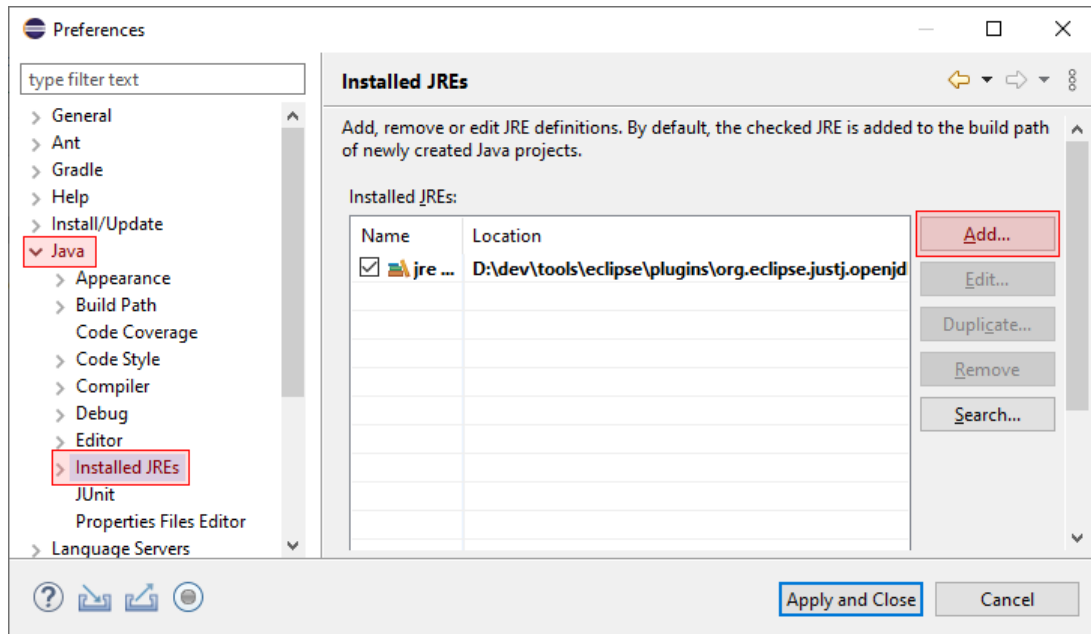


Figura 42: Preferências do Java no Eclipse.

Na janela “Add JRE”, escolha a opção “Standard VM” e clique em “Next”. Em seguida, no campo “JRE home:”, digite o local onde você descompactou o JDK ou use o botão “Directory...” para localizar o local do JDK que você quer incluir no Eclipse (Figura 43).

Quando o Eclipse detecta que o local indicado em “JRE home:” possui um JDK, o campo “JRE name:” é preenchido com uma sugestão inicial (você pode alterar esse nome) e aparece um ou mais itens no campo “JRE system libraries”. Quando isso acontecer, você pode clicar em “Finish” para concluir a inclusão do JDK no Eclipse.

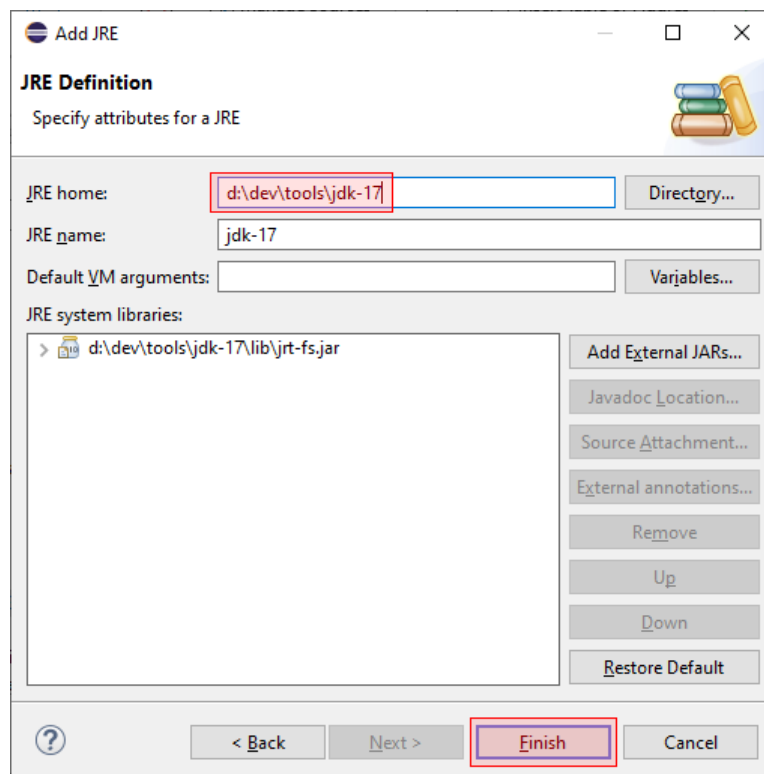


Figura 43: Adicionando um JDK no Eclipse.

Na Figura 43, o caminho `D:\dev\tools\jdk-17` foi indicado como *JRE home*, pois foi onde o JDK 17 foi descompactado na máquina usada para elaborar este material.

Após clicar em “*Finish*”, o Eclipse volta para a janela de preferências, agora mostrando o JDK que acabamos de adicionar e o *jre* padrão do Eclipse. Caso queira que o JDK seja o padrão do Eclipse para novos projetos, selecione-o na seção *Java > Installed JREs* das preferências, conforme a Figura 44. Em seguida, podemos clicar em “*Apply and Close*”.

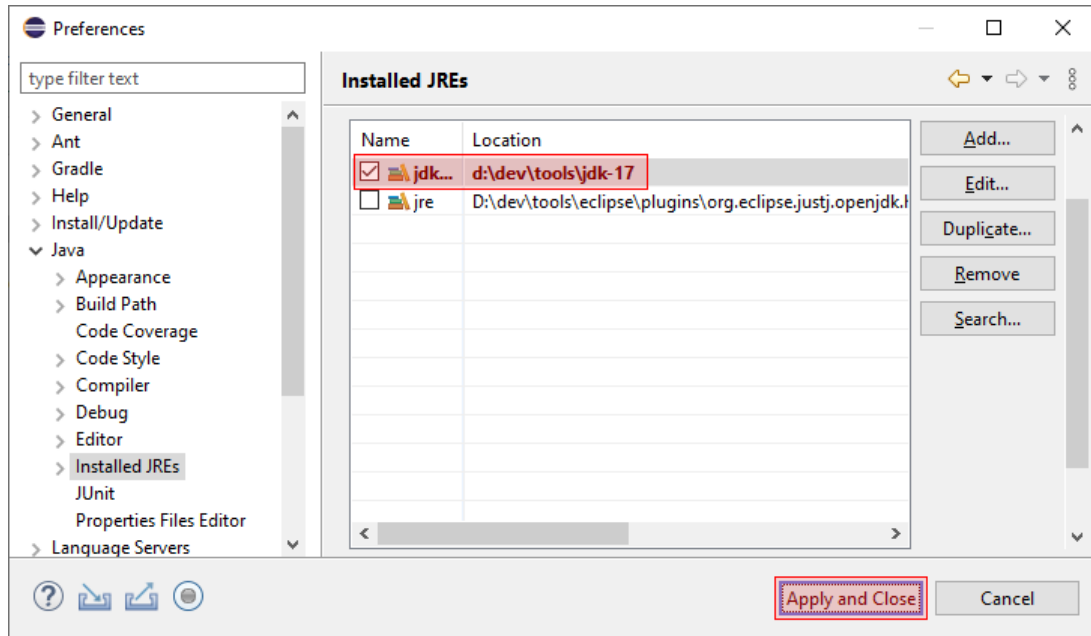


Figura 44: JDK adicionado e configurado como padrão para novos projetos do Eclipse.

Após realizar essa configuração do Eclipse, quando criar um projeto, escolha a opção “*Use default JRE and workspace compiler preferences*” do grupo “*JRE*” (Figura 45).

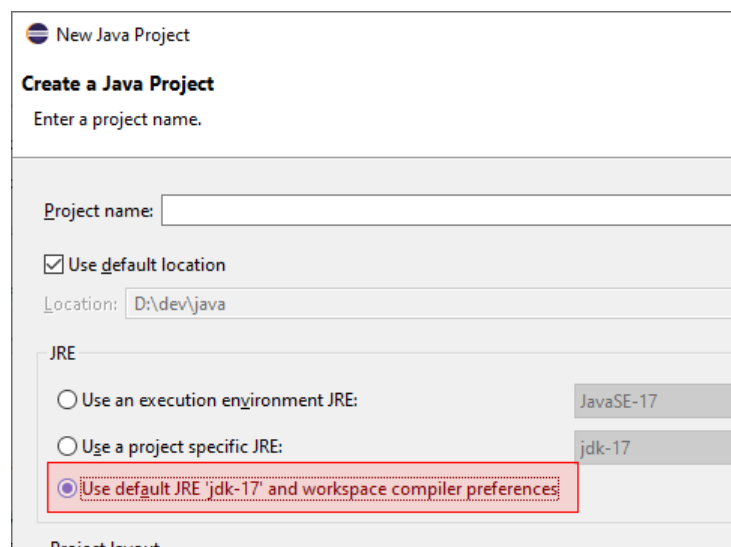


Figura 45: JDK selecionado durante a criação de um novo projeto.

→ **OK. Mas e os projetos que já existem, como o nosso *Hello, World*?**

Nesse caso, clique com o botão direito no item “*JRE System Library*” dentro do projeto *HelloWorld* e escolha a opção “*Properties*” (Figura 46).

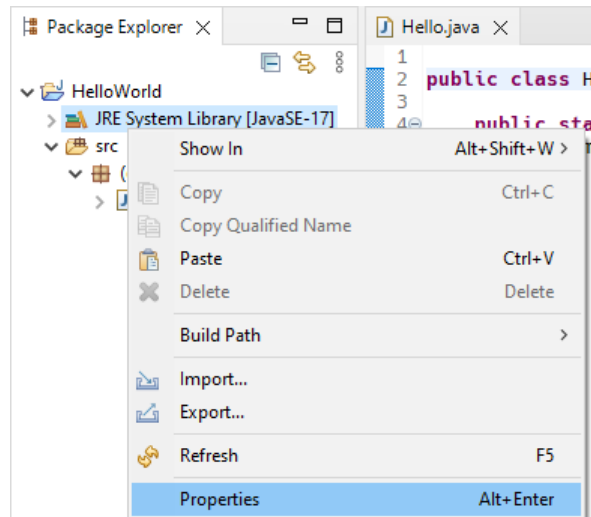


Figura 46: Propriedades do projeto (específico sobre JDK).

Na janela “*Properties for JRE System Library*”, escolha a opção “*Workspace default JRE*” ou use a opção “*Alternate JRE:*”, que permite a gente escolher um dos JDKs configurados no Eclipse. Por fim, clique no botão “*Apply and Close*”.

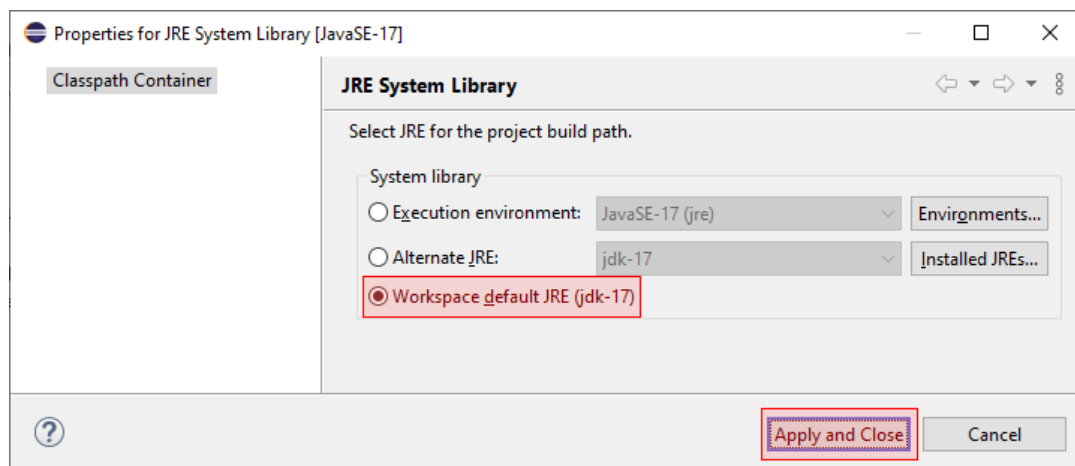


Figura 47: Alterando o JDK de um projeto já existente.

Observe que também é possível configurar os JDKs no Eclipse por meio dessa janela, clicando no botão “*Installed JREs...*”. Esse botão nos leva à mesma janela da Figura 42.

→ **OK! Algo mais?**

Não, por enquanto é isso. Finalmente chegamos ao final deste material!

Pode ser que eu faça alterações no material. Caso isso aconteça, as alterações estarão registradas no *Apêndice A: Histórico deste material*. Encontrou algum erro ou tem sugestões? Por favor, me avise por e-mail: [andre.kishimoto@mackenzie.br](mailto:andre.kishimoto@mackenzie.br). Agradeço antecipadamente!

Para finalizar, gostaria de indicar uma referência que acho muito boa para estudar Java e ficar por dentro das novidades da linguagem: o site Dev.java, principalmente as seções *Learn > Tutorials* e *Learn > FAQ* [10].



## Referências

- [1] SDKMAN!, “SDKMAN! the Software Development Kit Manager,” 2023. [Online]. Available: <https://sdkman.io/jdks>.
- [2] Oracle, “OpenJDK JDK 21.0.2 GA Release,” 2023. [Online]. Available: <https://jdk.java.net/21/>.
- [3] D. Smith, “The art of long-term support and what LTS means for the Java ecosystem,” 2021. [Online]. Available: <https://blogs.oracle.com/javamagazine/post/java-long-term-support-lts>.
- [4] S. Sheppard, “SS64 Command line reference,” 2023. [Online]. Available: <https://ss64.com/>.
- [5] The Apache Software Foundation, “Apache Ant,” 2022. [Online]. Available: <https://ant.apache.org/>.
- [6] The Apache Software Foundation, “Maven,” 2023. [Online]. Available: <https://maven.apache.org/>.
- [7] Gradle Inc., “Gradle Build Tool,” 2023. [Online]. Available: <https://gradle.org/>.
- [8] Eclipse Foundation, “The Community for Open Innovation and Collaboration,” 2023. [Online]. Available: <https://www.eclipse.org>.
- [9] Eclipse Foundation, “JDT Core - Eclipsepedia,” 2022. [Online]. Available: [https://wiki.eclipse.org/JDT\\_Core](https://wiki.eclipse.org/JDT_Core).
- [10] Oracle, “Learn Java - Dev.java,” 2023. [Online]. Available: <https://dev.java/learn/>.

## Apêndice A: Histórico deste material

### v1.2 (07/02/2024)

- Alterações no texto citando novas versões do JDK (ver nota de rodapé 2) e Eclipse (ver nota de rodapé 7).
- Referência [2] atualizada.
- O texto do material ainda não foi atualizado com JDK 21 e Eclipse 2023-12, mas os processo descritos aqui são válidos para essas novas versões.

### v1.1 (03/08/2023)

- Alterações no texto citando novas versões do JDK (ver nota de rodapé 2) e Eclipse (ver nota de rodapé 7) e inclusão da caixa de atenção sobre ordem do `%JAVA_HOME%\bin` no `PATH` (Figura 10, página 11).

### v1.0 (04/02/2023)

- Primeira versão do material contendo as seções **Objetivos**, **Introdução**, **TL;DR** (apenas aviso), **Ferramentas**, **JDK – Java Development Kit**, **Hello, World! em Java**, **Eclipse IDE**, **Hello, World! no Eclipse**, **Referências** e **Apêndice A: Histórico deste material**.