Módulo 01

Introdução e Primeiros Conceitos de Java

- Estratégias e Princípios Adotados para as Aulas
- Instalação das Ferramentas
- Uso do Eclipse
- Semelhanças entre C e Java
- Primeiros Projetos

Créditos

Autor

Prof. Alessandro Cerqueira

(alessandro.cerqueira@hotmail.com)

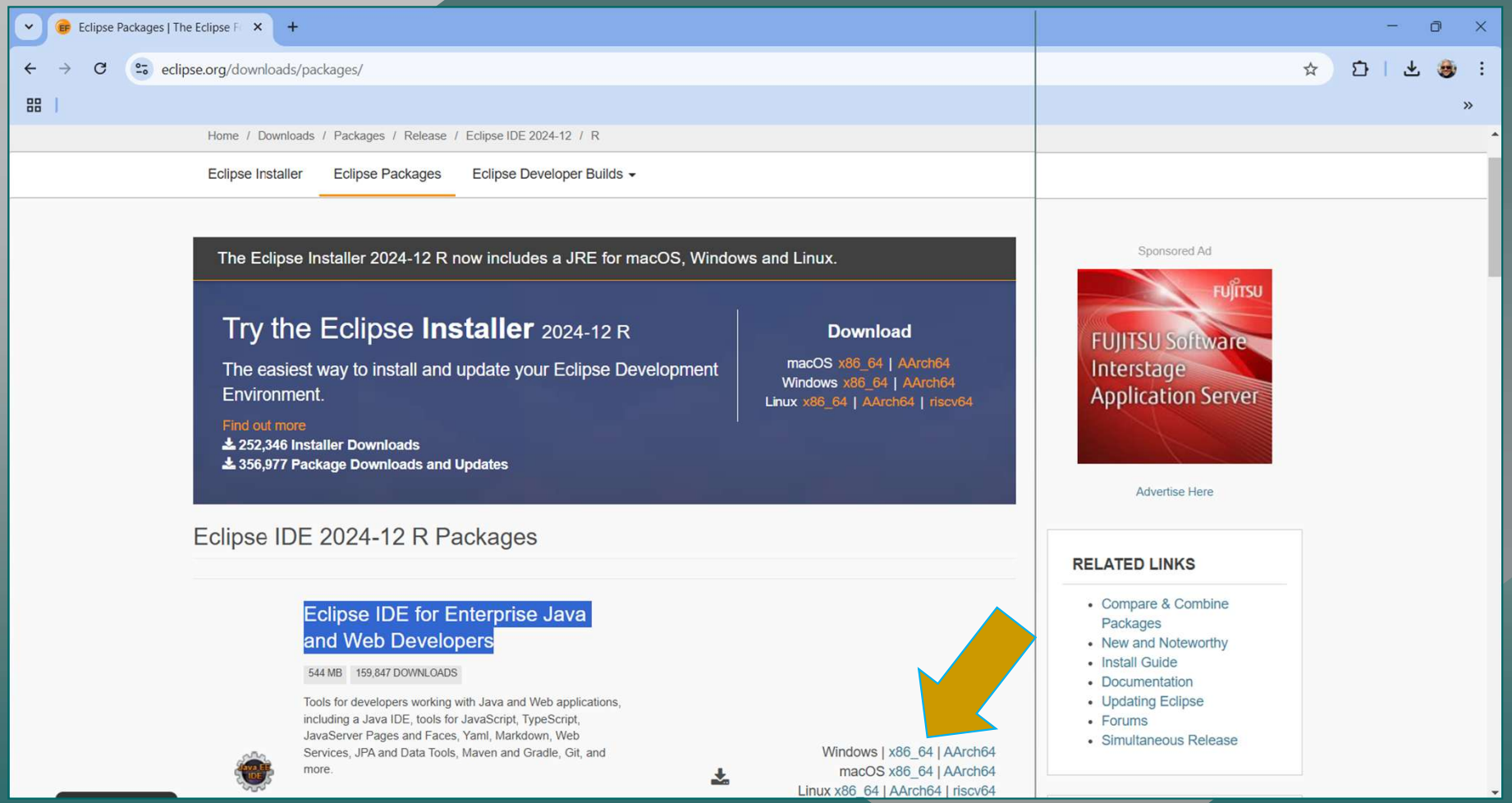
Princípios e Estratégias para Aulas

- Módulos destinados principalmente a quem conhece a Linguagem **C** (comandos, instruções e operadores)
- Para não trazer uma sobrecarga de conceitos, nem sempre serão apresentadas inicialmente as melhores práticas de programação ou de Java
  - Simplificações para termos uma progressão em ritmo moderado
  - Sequência próxima da evolução histórica de Java/princípios de programação
  - *Poderemos redefinir ou aperfeiçoar conceitos apresentados anteriormente*
- É **muitíssimo importante** dominar todos os conteúdos dos módulos anteriores ao que se estará apresentando.
  - Não acumule matéria para poder acompanhar os conteúdos

Instalação das Ferramentas

- **IDE**
  - Integrated Development Environment (Ambiente de Desenvolvimento de Software)
  - Há várias IDE's que podem ser utilizadas para desenvolvimento com Java, como IntelliJ, Visual Studio Code e NetBeans e IntelliJ, mas nos módulos vamos trabalhar com o **Eclipse**.
- **Eclipse**
  - Há várias instalações de "Eclipse": PHP, C/C++, ...
  - Endereço: *https://www.eclipse.org/downloads/packages/*
  - Baixe a versão Eclipse IDE for Enterprise Java and Web Developers
    - *Versão já vem com plugins para desenvolvimento de aplicações web e corporativas*

Eclipse IDE



Faça o download do arquivo .zip; descompacte; coloque a pasta descompactada preferencialmente em C:\ e faça um *shortcut* para o arquivo C:\Eclipse\Eclipse.exe

Java SE Development Kit (JDK)

- **JDK (*Java Development Kit*)**
  - Instalação opcional (**Eclipse** já possui as ferramentas!)
  - Composto de uma série de ferramentas básicas para o desenvolvimento com Java
  - Engloba:
    - ***JRE (Java Runtime Enviroment)*** → *Basicamente a Máquina Virtual Java, Arquivos JAR (Java Archives) com o bytecode das classes Standard Edition (Java SE)*
    - ***Compilador*** *(Javac) e outros programas acessórios*
  - Há a várias versões (inclusive da Oracle), mas prefira a **OpenJDK** (*https://jdk.java.net/*)

OpenJDK

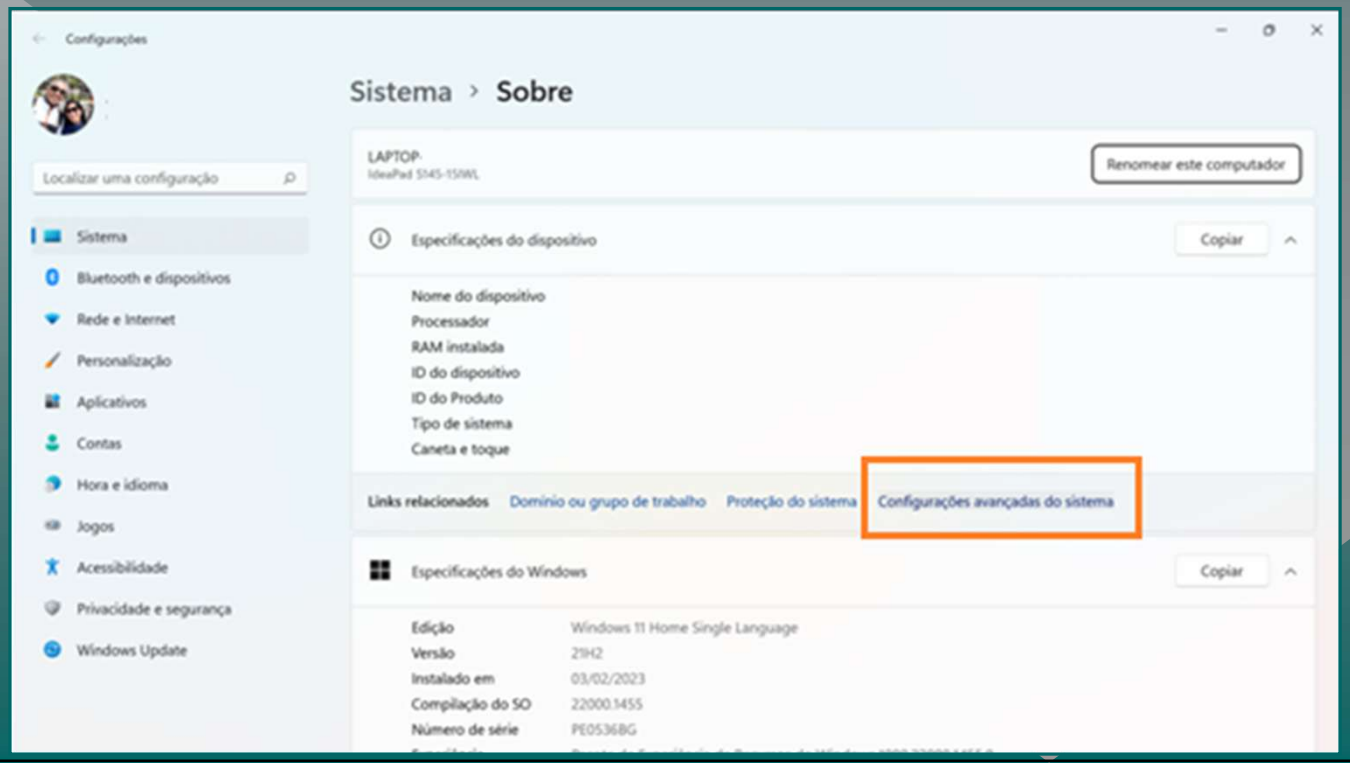

JDK Builds from Oracle
jdk.java.net
Erro
jdk.java.net
Production and Early-Access OpenJDK Builds, from Oracle
Ready for use: JDK 19, JMC 8
Early access: JDK 21, JDK 20, Generational ZGC, JavaFX 20, JavaFX 21, Jextract, Loom, Metropolis, Panama, & Valhalla
Looking to learn more about Java? Visit dev.java for the latest Java developer news and resources.
Looking for Oracle JDK builds and information about Oracle's enterprise Java products and services? Visit the Oracle JDK Download page.
ORACLE
© 2023 Oracle Corporation and/or its affiliates
Terms of Use · Privacy · Trademarks

JRE (Java Runtime Environment)

- Ambiente para execução de programas Java
  - Eclipse já incorpora uma JRE
  - Principal componente: Java Virtual Machine (tópico futuro)
  - https://www.java.com/pt-BR/download/

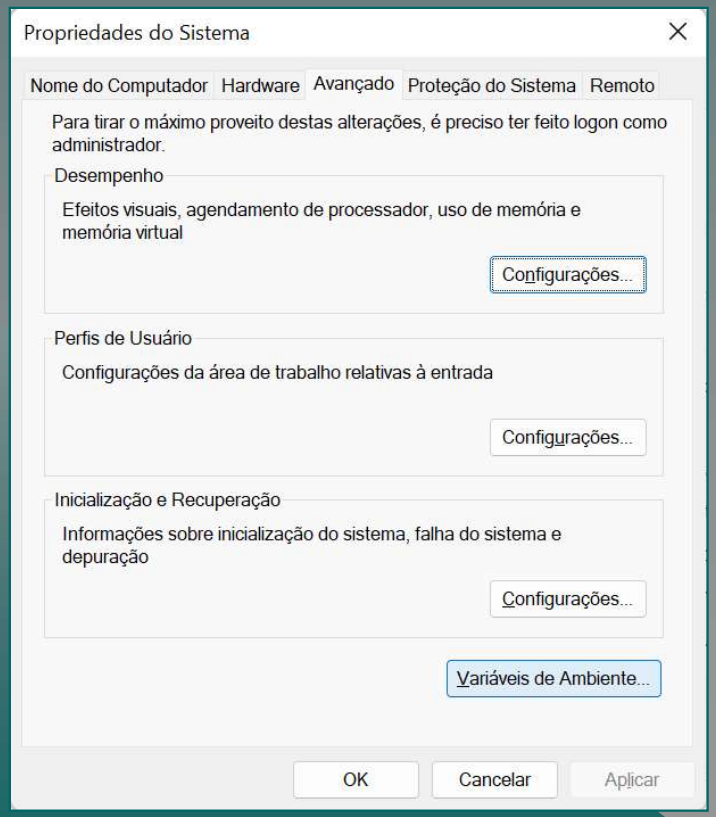
Execução sem o Eclipse

- Configuração no Painel de Controle
- Painel de Controle → [Sistema e Segurança] → [Sistema] → [Configurações Avançadas do Sistema]



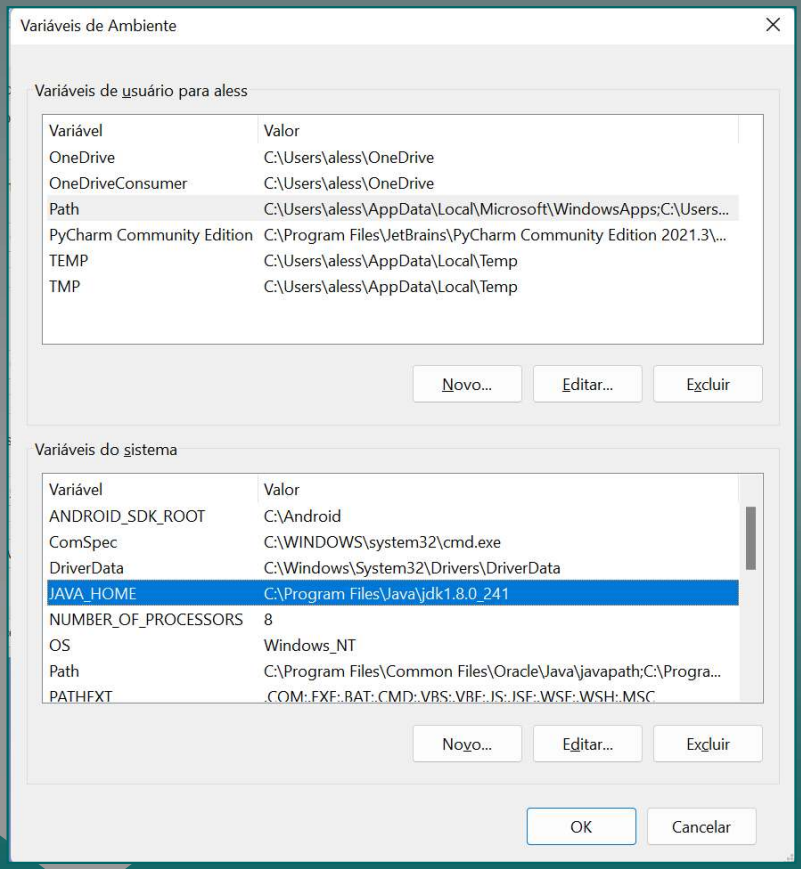
Configuração no Painel de Controle

- Acesse a opção [Variáveis de Ambiente]



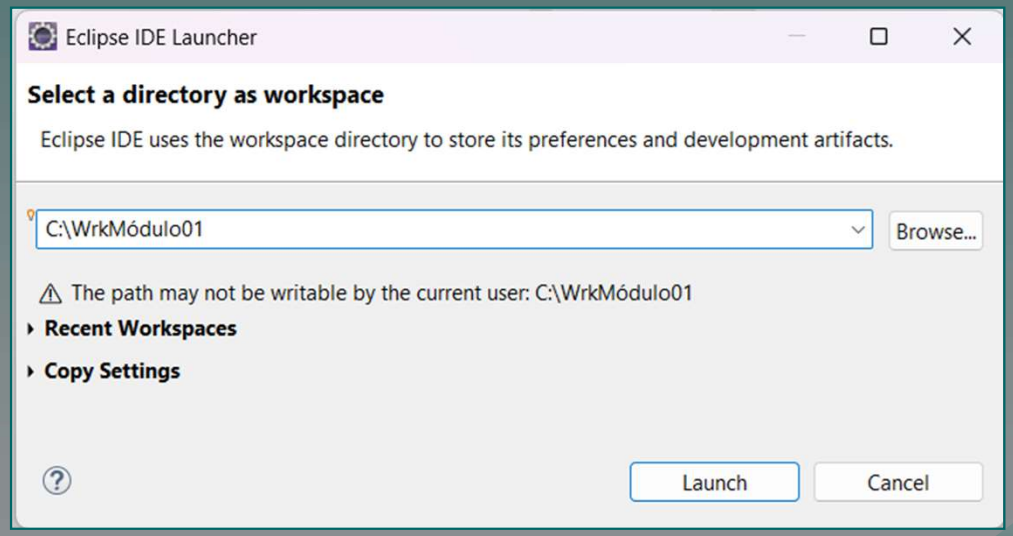
Configuração no Painel de Controle

- Adicionar a variável de ambiente JAVA_HOME indicando o caminho para a pasta do JDK ou JRE.
- Verifique a variável de ambiente PATH incorpora a indicação para a pasta ***bin*** do seu JDK ou JRE.



Introdução ao Uso do Eclipse

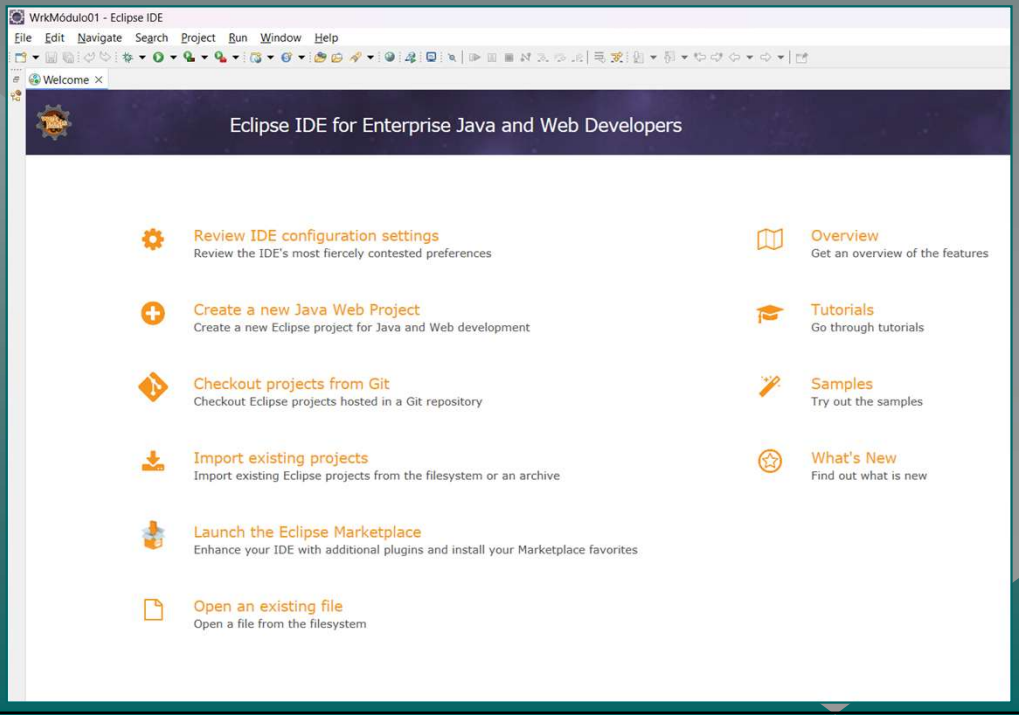
- Após iniciar o Eclipse, será solicitada a indicação do **Workspace**



- **Workspace**
  - Pasta/Diretório que contém um ou mais **Projetos**
- **Projeto**
  - Pasta/Diretório dentro do Workspace que representa um **Programa Java**.
  - Veremos que um Programa Java deverá ter uma ou mais **Classes**.

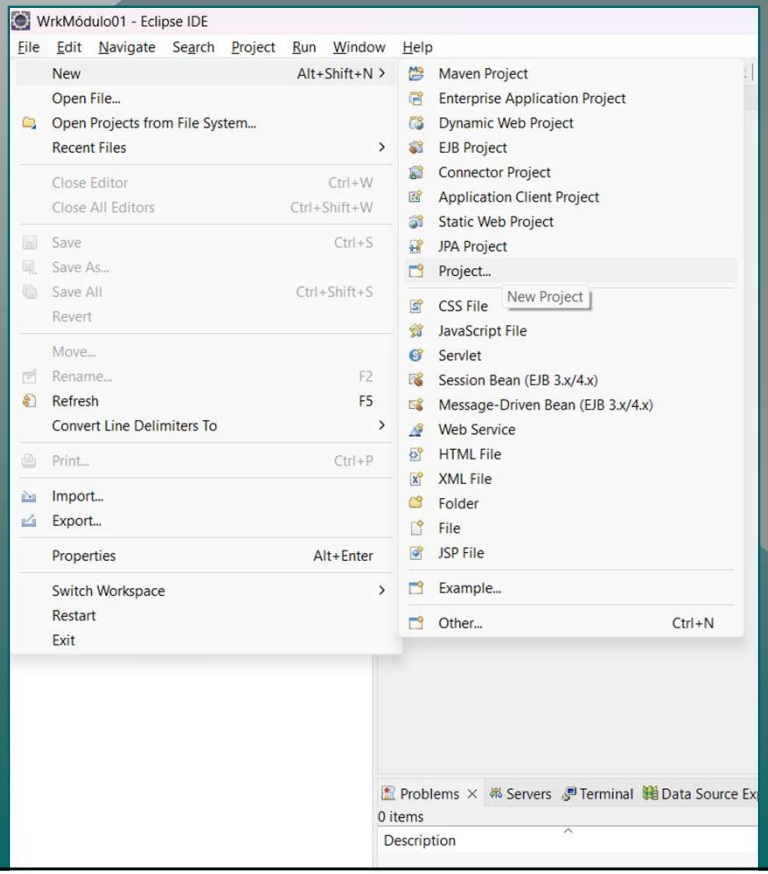
Primeiro Projeto no Eclipse

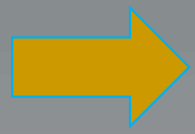
- Escolha uma pasta para conter o seu Workspace
  - Memorize a localização da pasta
  - Em **nossa convenção**, vamos sempre dar o prefixo **"Wrk"** para essa pasta
  - Na 1ª vez que o Workspace for aberto, será exibida a aba [Welcome] (pode fechá-la)

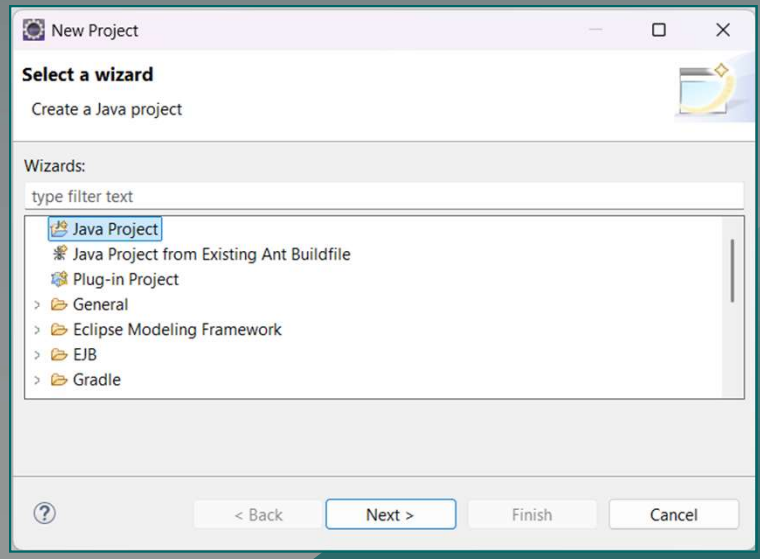


Primeiro Projeto no Eclipse

- Criando o primeiro Projeto: [File][New][Project] e depois [Java Project]

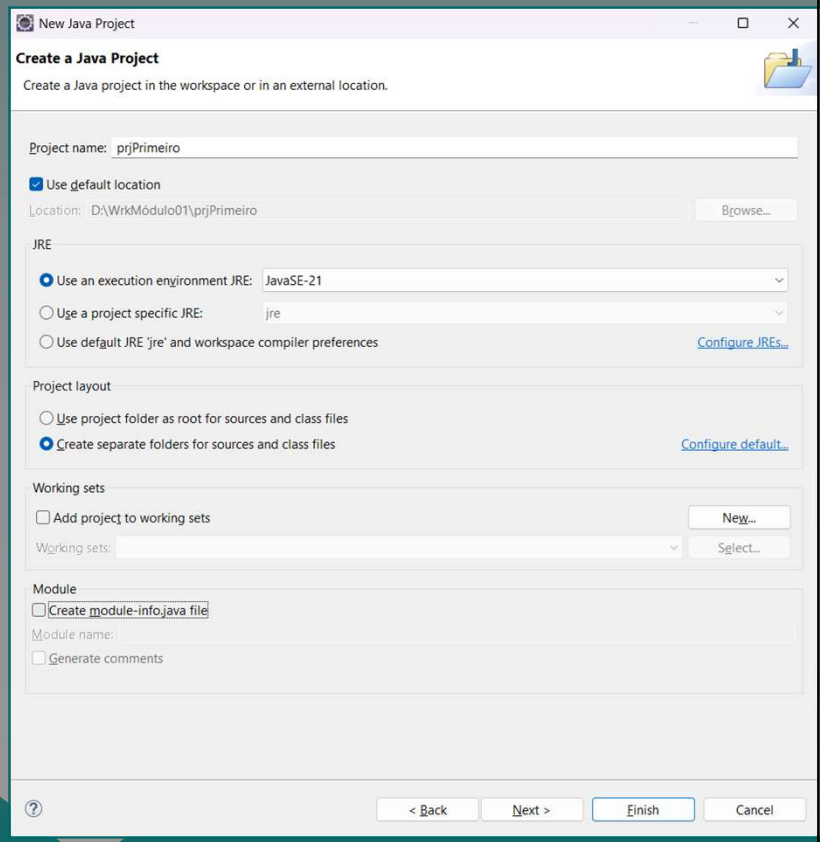






Primeiro Projeto no Eclipse

- Vamos chamar o projeto de “prjPrimeiro”
  - Em nossa convenção, vamos sempre dar o prefixo “**prj**” para a pasta de um projeto.
  - Importante: Desmarque a opção “*create module-info.java file*” (tópico futuro)
  - Clique em **[Finish]**



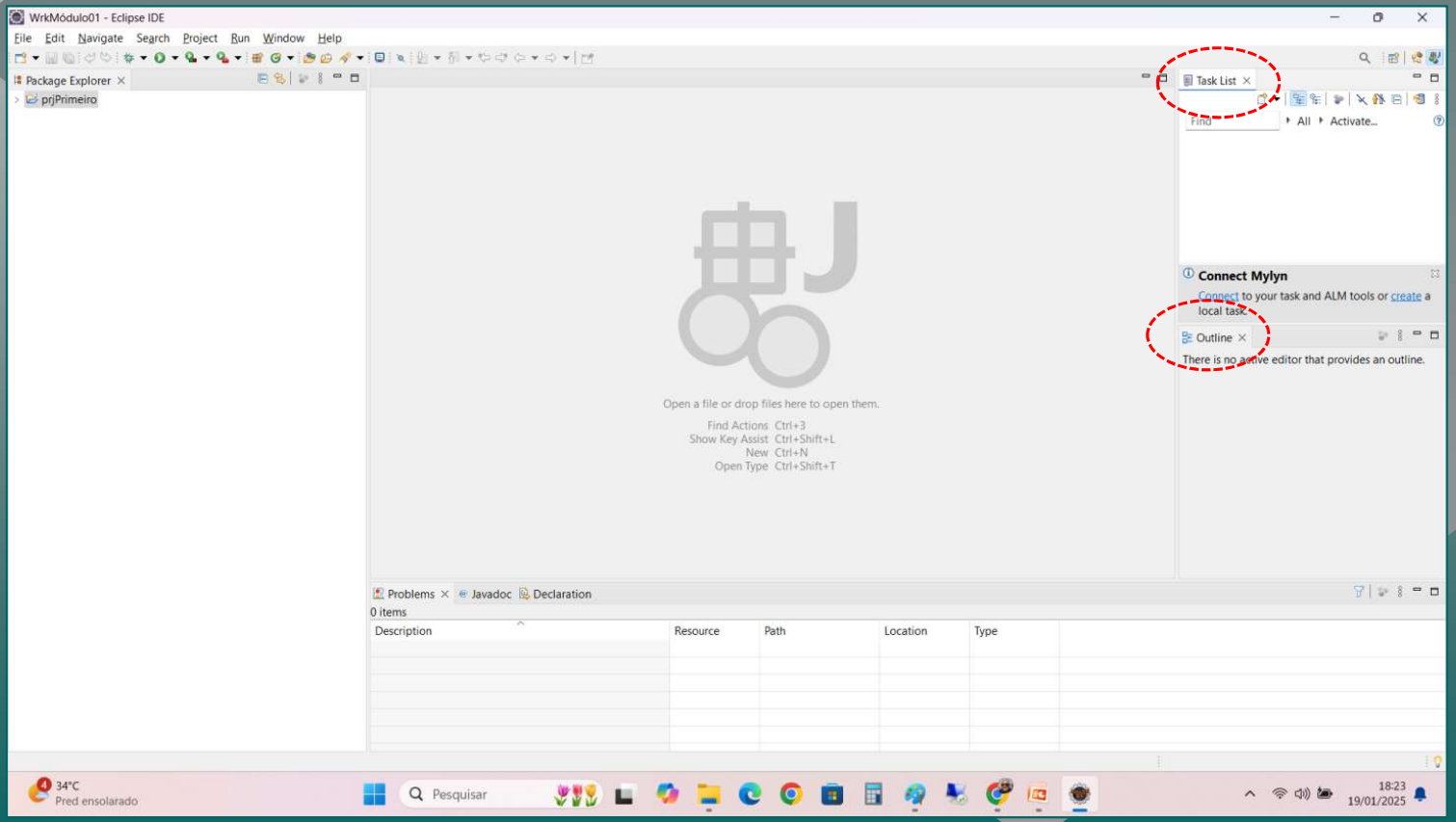
Primeiro Projeto no Eclipse

- O Eclipse perguntará se você deseja abrir a [Java Perspective]. Clique em [Open Perspective]

- **Perspectiva**
  - Conjunto de paineis com visões e editores organizados para apoiar um certo tipo de atividade no Eclipse.
  - Principais perspectivas: **Java**, Java EE e Debug.

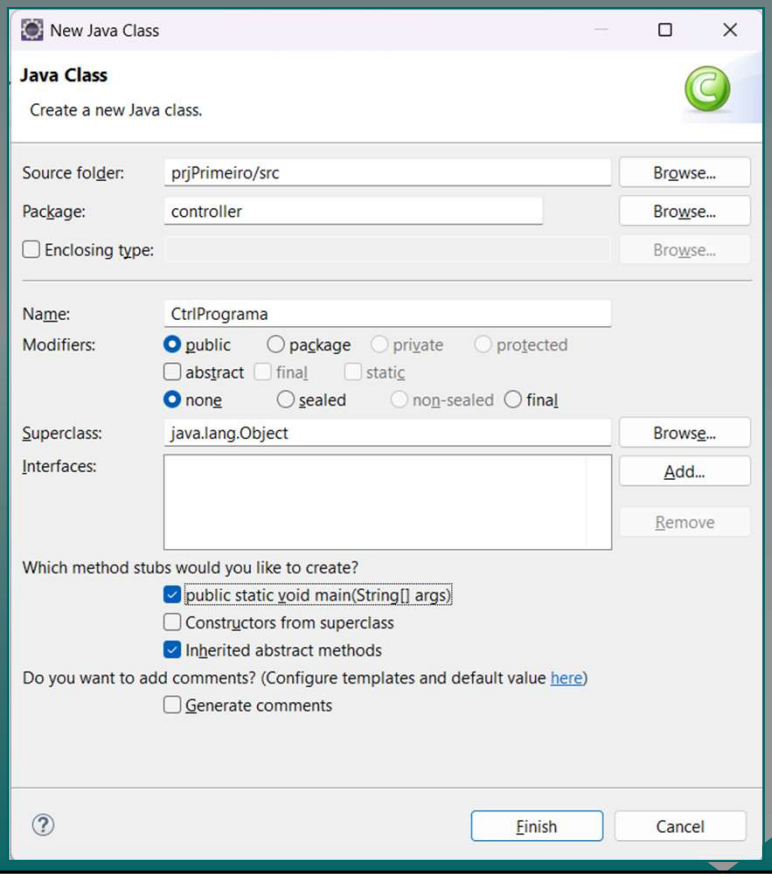
Primeiro Projeto no Eclipse

- Perspectiva Java
  - Sugerimos fechar os paineis/abas [Task List] e [Outline]



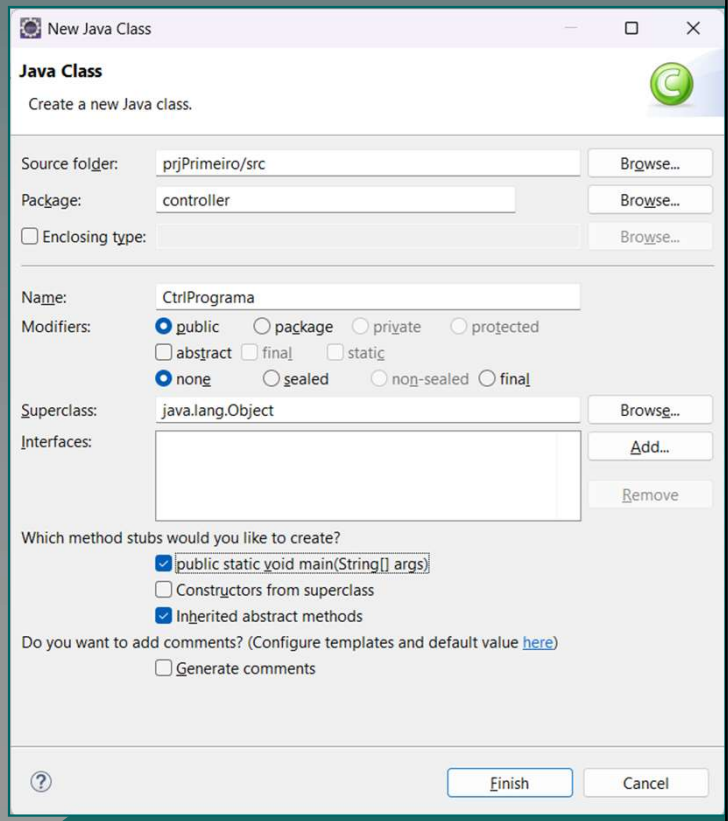
Primeiro Projeto no Eclipse

- Todo código em Java precisa estar em uma **Classe**.
- Criação de classe: `[File][New][Class]`



Primeiro Projeto no Eclipse

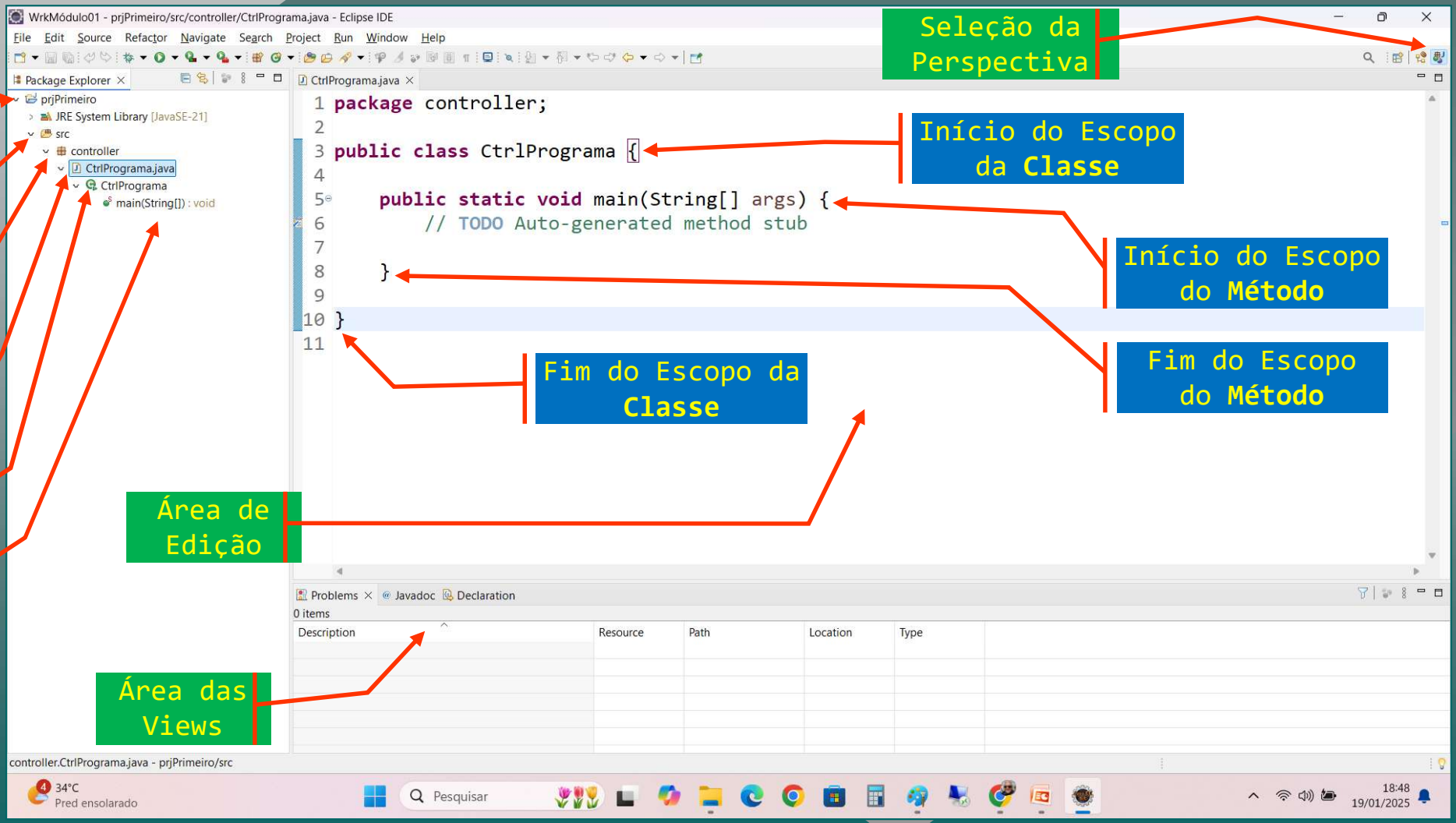
- Neste primeiro exemplo:
  - Preencha controller em Package (em letras minúsculas),
  - Preencha CtrlPrograma em Name (com C e P maiúsculos e sem espaço em branco – **Notação CamelCase**)
  - Marque a opção
    `public static void main(String[] args)`
- No Eclipse, os códigos das classes ficam na pasta **[src]** do Projeto.



Primeiro Projeto no Eclipse

- **Package (pacote)**
  - **Conjunto de classes** agrupadas por algum motivo determinado pelo desenvolvedor
  - Fisicamente é uma pasta dentro de [src]
- **Classe**
  - Especificação para geração de **objetos** (tópico futuro)
  - Toda classe fica em um arquivo cujo nome é formado pelo nome da classe + extensão **".java"**
- **Método**
  - **Código** de uma Função ou Procedimento presente na classe
- **Método main -** `public static void main(String[] args)`
  - Representa o método por onde um Programa Java é iniciado

Primeiro Projeto no Eclipse



Primeiro Projeto no Eclipse

- **Escopo (ou bloco)**
  - Área de Codificação para Classes, Métodos e Comandos da Linguagem
  - Em Linguagens cuja sintaxe foi baseada em C, um escopo é delimitado por **{** e **}**
- **Indentação**
  - Prática adotada pelos programadores para organizar os escopos de um código.
  - Em linguagens cuja sintaxe foi baseada em C, indentar não obrigatório, mas muitíssimo importante!

Regras de Indentação

- O código sempre começa na coluna **1** da classe.
- Sempre que colocarmos "{" para um escopo, na linha abaixo deveremos **acrescentar** um novo nível de indentação. Para isto devemos usar a tecla "**tab**"
- Mantemos o mesmo número de tabs da linha anterior, a não ser que na linha anterior tenha outro "{".
- Antes de colocar um "}" retiramos um nível de indentação.
- A mesma regra de indentação vale para os comandos for, if, while, do...while contendo uma única instrução ou um comando.

Configurações e *Shortcuts* no Eclipse

- **Reiniciar Configuração da Perspectiva:**
  `[Window] [Perspective] [Reset Perspective]`
- **Adicionar View [Tasks]:**
  `[Window] [Show View] [Tasks]`
  - View útil para localizarmos os comentários `//TODO ....`
- **Troca de Fonte na Edição:**
  `[Window] [Preferences] [General] [Appearence] [Colors and Fonts] [Java] [Java Editor] [Edit]`
- **Troca de Fonte na Console:**
  `[Window] [Preferences] [General] [Appearence] [Colors and Fonts] [Basic] [Text Font] [Edit]`
- **Troca do Tema de Fundo:**
  `[Window] [Preferences] [General] [Appearence] [Theme]`

Configurações e *Shortcuts* no Eclipse

- **Para Salvar:**
  `[Ctrl + S]`
- **Para Autoindentação:**
  `[Ctrl + Shift + F]` (o código precisa estar sem erro)
- **Para Organização dos imports:**
  [Alt + Shift + O]
- **Para Renomear Classes, Atributos, Métodos ou Variáveis por Todo o Código (Refatoração):**
  Clicar sobre o elemento e teclar [Alt + Shift + R]
- **Para Autocompletar:**
  <termo> **e** `[Ctrl + Espaço]`
  - Exemplo: `Sysout` `[Ctrl + Espaço]` → irá escrever `System.out.println`

Primeiro Projeto no Eclipse

- `System.out.println`
  - Escreve na console e realiza a quebra de linha

*CtrlPrograma.java ×

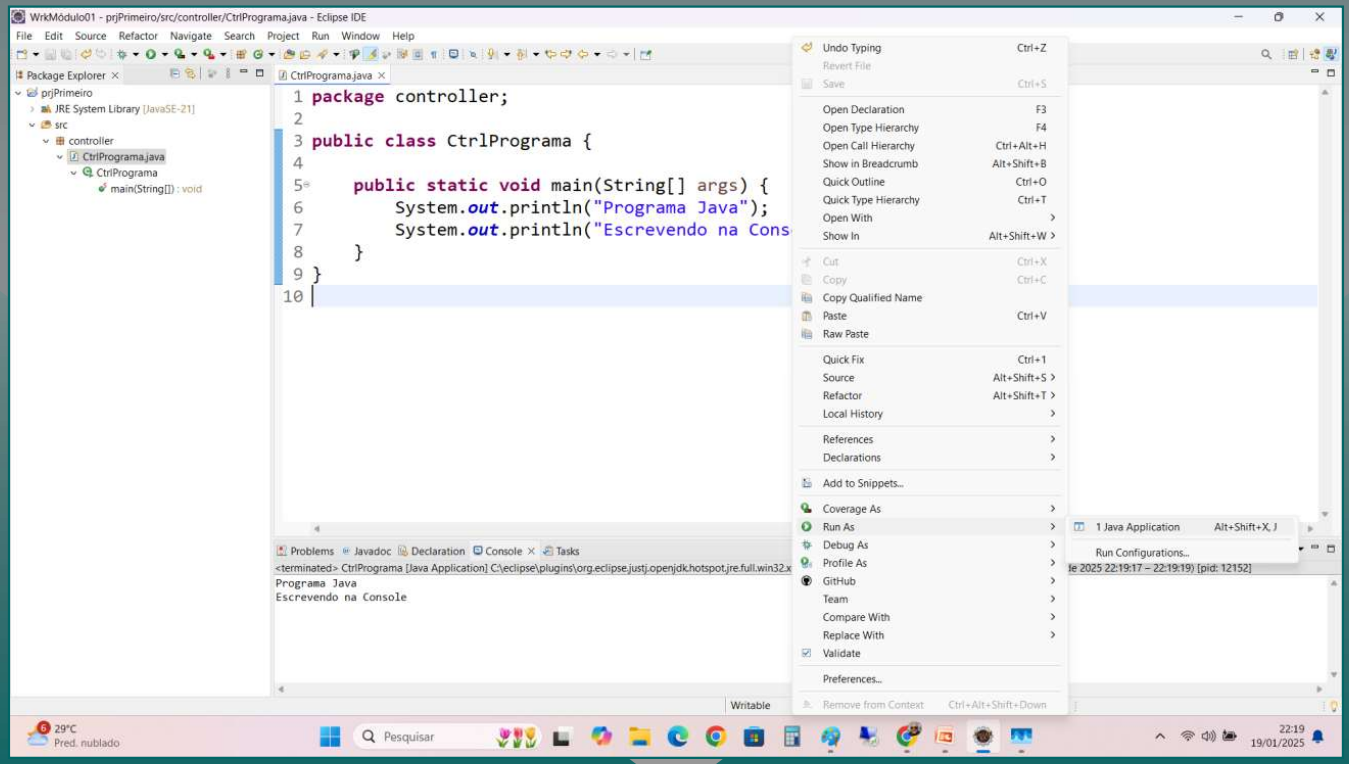
```
1 package controller;
2
3 public class CtrlPrograma {
4
5     public static void main(String[] args) {
6         System.out.println("Programa Java");
7         System.out.println("Escrevendo na Console");
8     }
9 }
```

Observe que na aba do arquivo há a presença de um asterisco '*'. Isso indica que o arquivo ainda não foi salvo.

- Java é uma linguagem **Case Sensitive**; ou seja, faz diferença entre letras maiúsculas e minúsculas. Assim, atenção durante a codificação.
- Literal String
  - Expressa com aspas duplas (parâmetros de `System.out.println` nas linhas 6 e 7)
  - Representa um objeto String (tópico futuro)

Primeiro Projeto no Eclipse

- Executando o Programa
  - Clique com o botão direito do mouse no código da classe que contém o método main: `[Run As][Java Application]`
  - A saída será exibida na aba `[Console]`
  - Para reexecutar o último programa clique em 



Tipagem

- Ao declaramos uma variável, parâmetro ou atributo (tópico futuro), o objetivo é reservar uma **área de memória** para guardar um **valor** durante a execução do programa.
- Java é uma **linguagem fortemente tipada**
  - Isso significa dizer que, ao declararmos uma variável, parâmetro ou atributo, precisamos indicar o **tipo de valor** esses elementos poderão armazenar.
  - Ao analisar uma atribuição ou passagem de parâmetro, o compilador verifica se o tipo dos elementos envolvidos é igual ou compatível.
- Dessa forma, é importante conhecer o sistema de tipos da linguagem.

Sistemas de Tipos em Java

Tipos Primitivos (em letras minúsculas)

TIPO	Default	Tamanho	Faixa de Valores
byte	0	(1 byte)	[-128 .. 127]
short	0	(2 bytes)	[-32768 .. 32767]
int	0	(4 bytes)	[-2147483648 .. 2147483647]
long	0L	(8 bytes)	[-9223372036854775808 .. -9223372036854775807]
float	0.0f	(4 bytes)	[1.401298464324817 e -45 .. 34028234663852886 e 38]
double	0.0d	(8 bytes)	[4.9 e -324 .. 1.7976931348623157 e 308]
char	'\u0000'	(2 bytes)	
boolean	false	(1 byte)	

Demais Tipos (**ponteiro para Arrays ou para Objetos** - tópico futuro)

int[]	null	(8 bytes)
String	null	(8 bytes)

Tipagem

- A declaração de variáveis, parâmetros ou atributos sempre seguirá o padrão empregado na Linguagem C:

```
<Tipo> <Identificador>
```

- Quando as variáveis, parâmetros ou atributos são de **Tipo Primitivo**, de fato eles irão armazenar um valor do tipo indicado na declaração.

Ex: `int i = 10;` `i:` **10**

O **Tipo** de **‘i’** é **‘int’** pois esse é um **Tipo Primitivo**; Assim, de fato, **‘i’** pode armazenar valores **‘int’**.

Operador de Atribuição

- A atribuição tem a tarefa de **pegar o resultado** da **expressão à direita** e **copiar esse resultado** na variável ou atributo indicado do lado esquerdo.
- Exemplos:

```
int i = 10; // O resultado da expressão à direita é 10.
            // Então esse valor é copiado na variável i

int j = i; // O resultado da expressão à direita é 10, pois o
           // valor de i é 10. Esse valor é copiado em j

int k = (i * j) - 8; // O resultado da expressão à direita é 92.
                     // Assim, esse valor será copiado em k
```

- Quando a variável ou atributo à esquerda é de **Tipo Primitivo**, dizemos que temos uma **Atribuição por Valor** (*de tipo primitivo*).

Primeiros Exemplos de Código Java

```
1  package controller;
2
3  public class CtrlPrograma {
4
5      public static void main(String[] args) {
6          int i = 10;
7          System.out.println(i);
8          i = i * 2;
9          System.out.println("A variável 'i' está com o valor = "  +   i);
10     }
11
12 }
```

- Alterando o método `main` do exemplo, podemos ver que `System.out.println` pode escrever o valor de variáveis (linha 7)
- O operador **+** não é somente aritmético. Quando um dos operandos envolver uma String, ele se torna um operador de concatenação (linha 9)

Primeiros Exemplos de Código Java

```
1  package controller;
2
3  public class CtrlPrograma {
4      public static void main(String[] args) {
5          for(int i = 1; i <= 10; i++)
6              if(i % 2 == 0)
7                  System.out.println(i + " é um número par!");
8              else
9                  System.out.println(i + " é um número ímpar!");
10     }
11 }
```

- Os comandos `if`, `for`, `while`, `switch`, `do...while` da Linguagem C também estão em Java. Também quase todos os operadores de C também fazem parte de Java. O operador `%` (*módulo* ou *resto de divisão inteira*) está entre eles (linha 6)
- No exemplo acima não é necessário empregar { e } para o comando `for` pois seu escopo é composto por somente um outro comando (`if`/`else`). Por sua vez, também não é necessário usar { e } para o `if` ou `else`, pois seu escopo é formado por uma única instrução.

Primeiros Exemplos de Código Java

```
1  package controller;
2
3  import java.util.Scanner;
4
5  public class CtrlPrograma {
6      public static void main(String[] args) {
7          // Declaração da variável 'teclado' para leitura de valores
8          Scanner teclado = new Scanner(System.in);
9          // Leitura de valor do teclado
10         System.out.print("Entre com um valor: ");
11         int valor = teclado.nextInt();
12         // Verificando a paridade do valor lido
13         if (valor % 2 == 0)
14             System.out.println(valor + " é um número par!");
15         else
16             System.out.println(valor + " é um número ímpar!");
17     }
18 }
```

- Para fazer a leitura de valores na Console, é necessário termos uma variável `Scanner` (linha 8). Futuramente trataremos dos operadores = e new que estão presentes na instrução.
- Observe que é necessário colocar a instrução `import` (linha 3) para essa classe. Use Ctrl+Shift+O no Eclipse para organização dos imports necessários.

Primeiros Exemplos de Código Java

```
1  package controller;
2
3  import java.util.Scanner;
4
5  public class CtrlPrograma {
6      public static void main(String[] args) {
7          // Declaração da variável 'teclado' para leitura de valores
8          Scanner teclado = new Scanner(System.in);
9          // Leitura de valor do teclado
10         System.out.print("Entre com um valor: ");
11         int valor = teclado.nextInt();
12         // Verificando a paridade do valor lido
13         if (valor % 2 == 0)
14             System.out.println(valor + " é um número par!");
15         else
16             System.out.println(valor + " é um número ímpar!");
17     }
18 }
```

- Observe que usamos `print` e não `println` (linha 10). Essa instrução irá escrever na Console sem a quebra de linha.
- A inclusão de comentários segue os mesmos padrões da **C**. O uso de **//** faz com que o restante da linha seja ignorado pelo compilador (Comentário de Linha)
- Para execução no Eclipse, é necessário clicar na área da Console.

Primeiros Exemplos de Código Java

```java
public class CtrlPrograma {

    public static boolean ehPar(int numero) {
        if (numero % 2 == 0)
            return true;
        return false;
    }

    public static void main(String[] args) {
        // Declaração da variável 'teclado' para leitura de valores
        Scanner teclado = new Scanner(System.in);
        // Leitura de valor do teclado
        System.out.print("Entre com um valor: ");
        int valor = teclado.nextInt();
        // Verificando a paridade do valor lido
        if (ehPar(valor))
            System.out.println(valor + " é um número par!");
        else
            System.out.println(valor + " é um número ímpar!");
    }
}
```

- Podemos adicionar métodos à Classe, assim como na Linguagem C podemos criar funções. Por enquanto, todos os métodos que serão criados terão a indicação "`public static`" no início de sua declaração (tópico futuro).
- No exemplo, `ehPar` recebe um inteiro como parâmetro e retorna um booleano. Observe que Java tem formalmente o tipo primitivo `boolean`.

Primeiros Exemplos de Código Java

```java
public class CtrlPrograma {

    public static boolean ehPar(int numero) {
        if (numero % 2 == 0)
            return true;
        return false;
    }

    public static void main(String[] args) {
        // Declaração da variável 'teclado' para leitura de valores
        Scanner teclado = new Scanner(System.in);
        // Leitura de valor do teclado
        System.out.print("Entre com um valor: ");
        int valor = teclado.nextInt();
        // Verificando a paridade do valor lido
        if (ehPar(valor))
            System.out.println(valor + " é um número par!");
        else
            System.out.println(valor + " é um número ímpar!");
    }
}
```

- Podemos adicionar métodos à Classe, assim como na Linguagem C podemos criar funções. Por enquanto, todos os métodos que serão criados terão a indicação "`public static`" no início de sua declaração (tópico futuro).
- No exemplo, `ehPar` recebe como parâmetro um inteiro e retorna um booleano. Observe que Java possui o tipo primitivo `boolean`, além dos valores `true` e `false`.

Primeiros Exemplos de Código Java

```
7    public static int contaVogais(String texto) {
8        int contador = 0;
9        for(int i = 0; i < texto.length(); i++) {
10           char c = texto.charAt(i);
11           if(c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||
12              c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')
13               contador++;
14       }
15       return contador;
16   }
```

- Nesse novo método, `contaVogais` tem o parâmetro **`String texto`** e retorna um inteiro. Nos próximos módulos, veremos que o tipo de `texto` não é exatamente uma `String`, mas **ponteiro** (ou referência) **para `String`** .
- No exemplo, há dois exemplos de operações (**métodos**) que podemos realizar com `String`. Na linha 9, `texto.length()` é utilizado para saber quantos caracteres há na String referenciada por `texto`.
- Já na linha 10, para sabermos o caracter presente em uma determinada posição da String usamos o método `.charAt(i)`, onde o parâmetro indica a posição desejada. Lembre-se que o primeiro caracter em uma String está presente na posição 0.

Primeiros Exemplos de Código Java

```
18      public static void main(String[] args) {
19          // Declaração da variável 'teclado' para leitura de valores
20          Scanner teclado = new Scanner(System.in);
21          // Leitura de valor do teclado
22          System.out.print("Entre com um texto: ");
23          String texto = teclado.nextLine();
24          // Informando o número de vogais
25          System.out.println("O texto possui " + contaVogais(texto) + " vogais!");
26      }
```

- Alteramos o método `main` para fazer a leitura de uma String na linha 23, usando `teclado.nextLine()`.
- Observe na linha 25 que fazemos duas concatenações e no meio temos a chamada a `contaVogais`.

Primeiros Exemplos de Código Java

```
18      public static String inverter(String texto) {
19          String resultado = "";
20          for(int i = texto.length() - 1; i > 0; i--)
21              resultado += texto.charAt(i);
22          return resultado;
23      }
```

- Nesse último exemplo, foi criado o método `inverter` cujo objetivo é produzir uma nova String com a inversão da ordem dos caracteres.
- Observe na linha 21 o operador **+=** que atuará primeiramente concatenando a String referenciada por `resultado` com o caracter da posição `i`, e fazendo uma nova atribuição à variável `resultado`.

Conclusão

- Neste módulo, foi possível:
  - Instalar e usar o IDE Eclipse
  - Codificar os primeiros exemplos com a Linguagem Java, a partir de conceitos presentes na Linguagem C
- No próximo módulo vamos aprender:
  - Modelo Orientado a Objeto
  - Codificar classes de dados usando conceitos de orientação a objetos