

CLASSES E OBJETOS

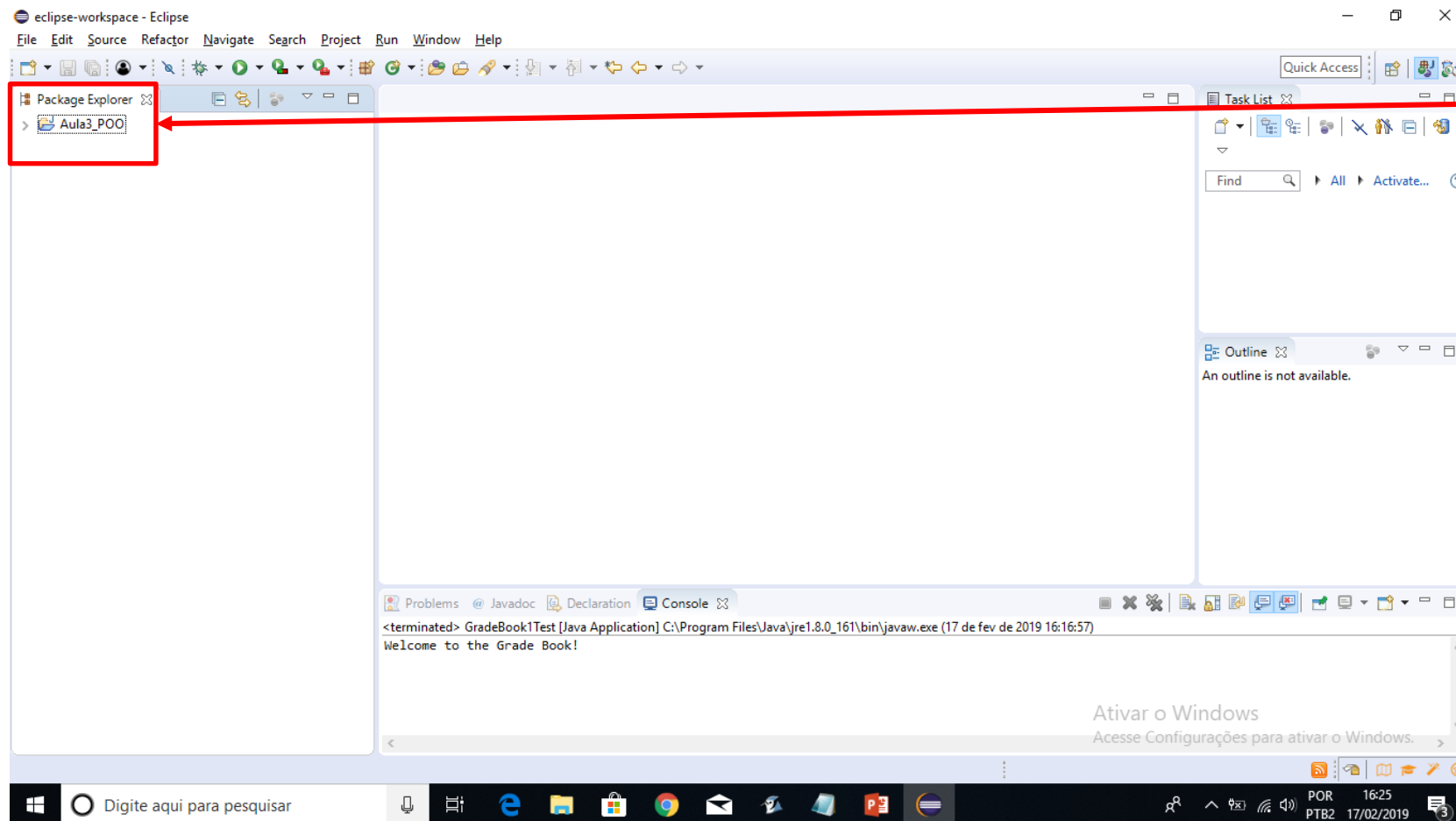
Prof. Ricardo Mesquita

Declarando uma Classe

- Cada declaração de classe que se inicie com a palavra-chave ***public*** deve ser armazenada em um arquivo que tenha o mesmo nome da classe.
 - ***Atenção:*** *declarar mais de uma classe public no mesmo arquivo é um erro de compilação.*
 - *Exemplo:*
 - Vamos criar um programa em Java com uma classe simples. Chamaremos essa classe de “Boletim”.

Declarando uma Classe

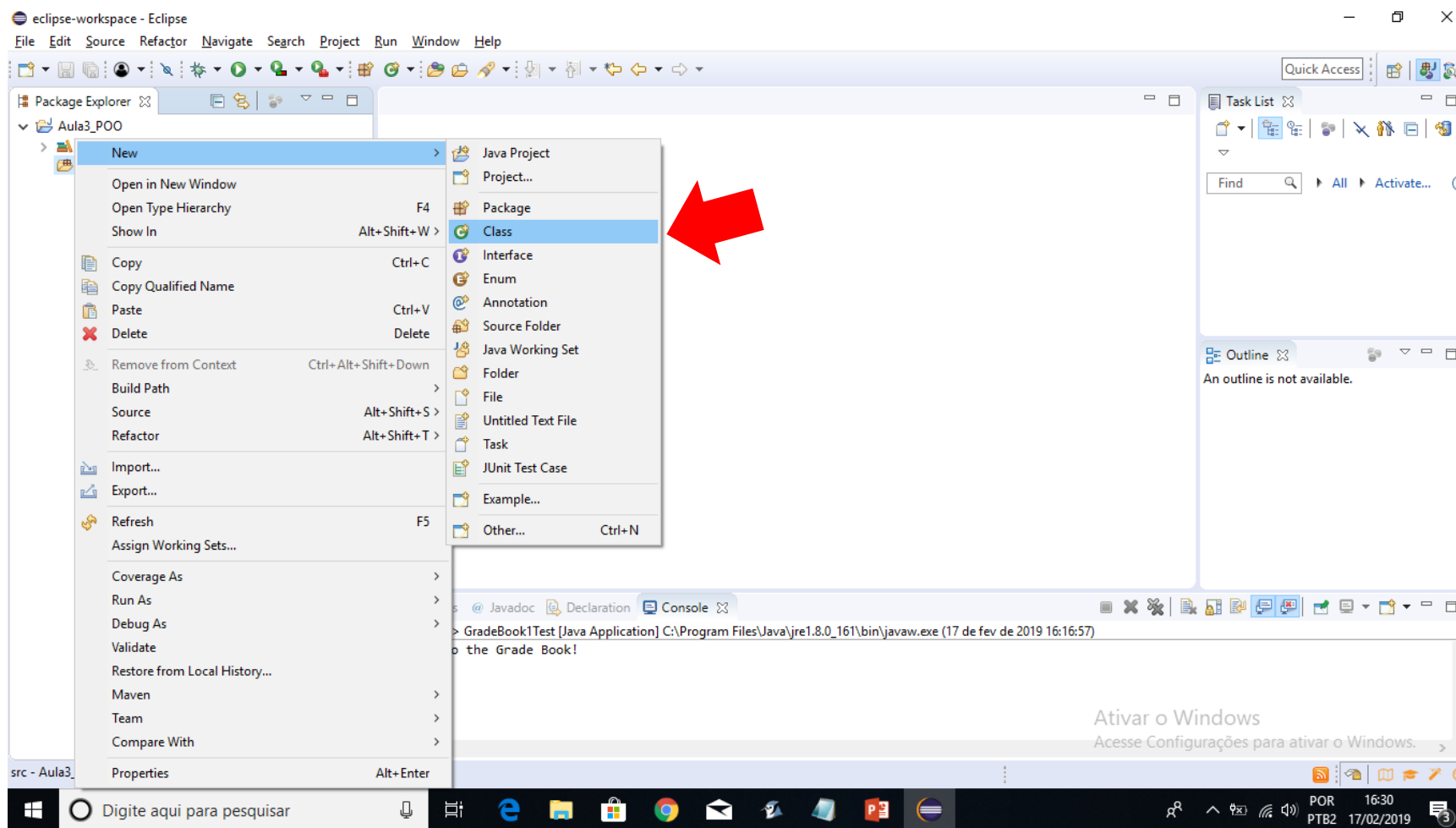
- Em primeiro lugar, *crie um projeto*.



Dê um nome.
Por exemplo:
Aula3_POO

Declarando uma Classe

- Vamos declarar agora a classe Boletim

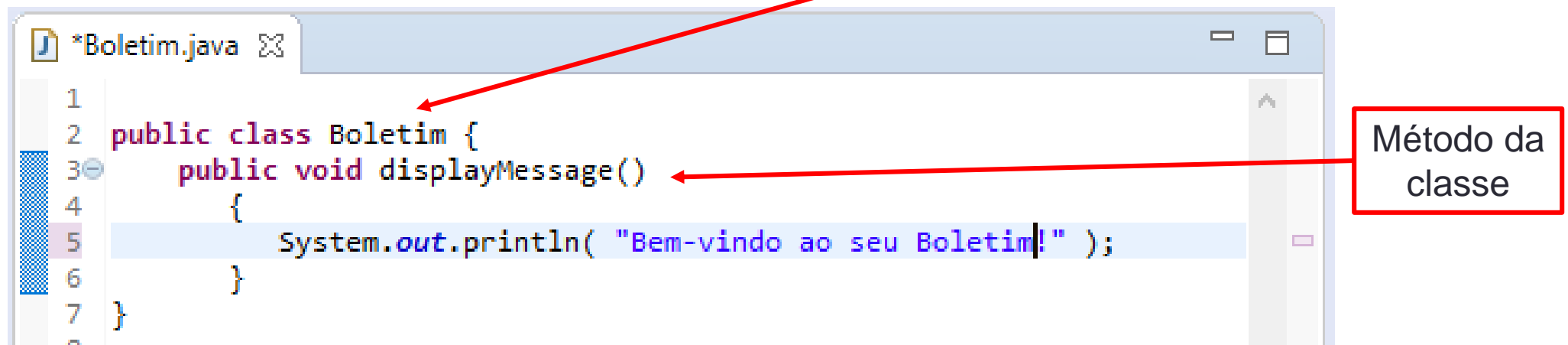


Escolha o nome:
Boletim

Atenção: o arquivo
java tem que ter o
mesmo nome da
classe principal!

Declarando uma Classe

- No espaço de edição, escreva o código:



- O que acontece se tentarmos executar?
 - Dispara uma **exceção**

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

- Note que essa classe não contém o método main!

Declarando uma Classe

- Uma saída para o problema: declare o método *main* na própria classe:

The screenshot shows a Java IDE window titled "Boletim.java". The code is as follows:

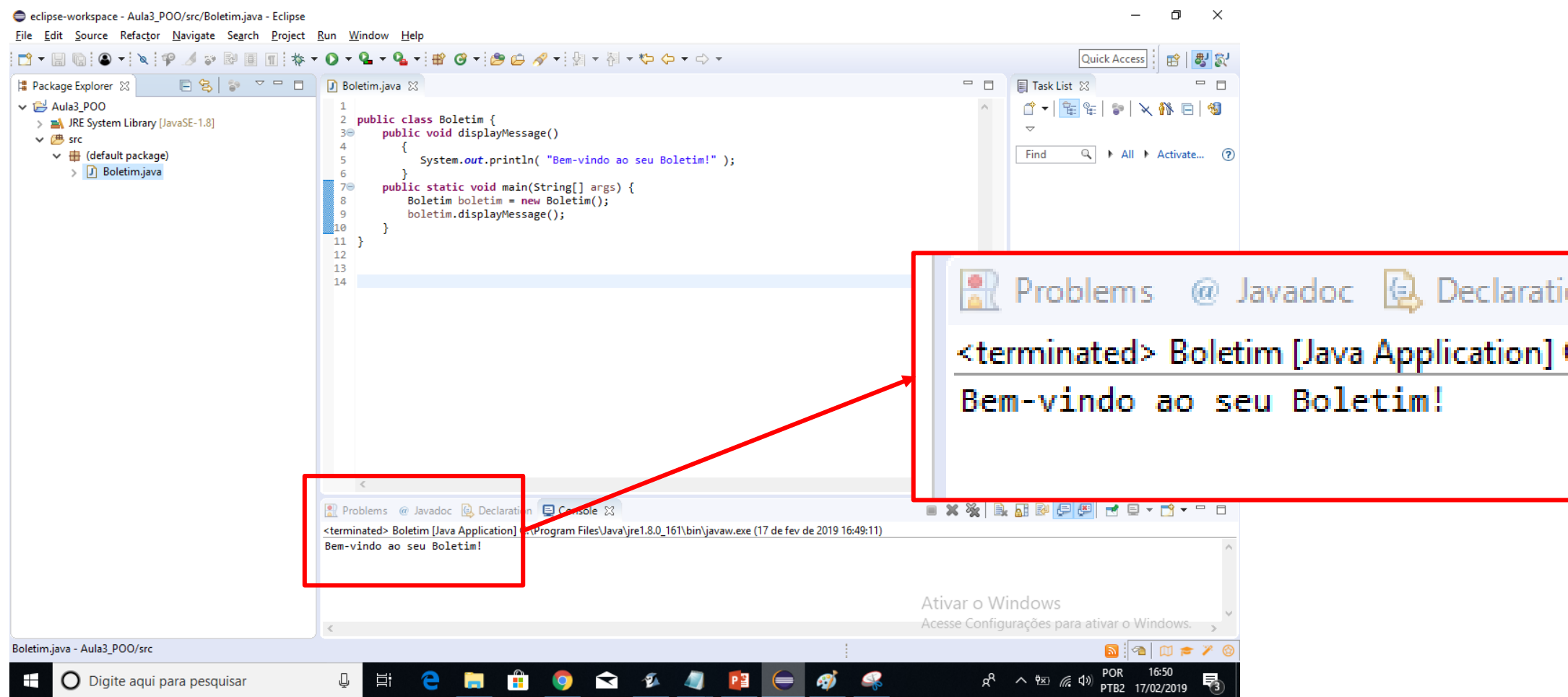
```
1
2 public class Boletim {
3     public void displayMessage()
4     {
5         System.out.println( "Bem-vindo ao seu Boletim!" );
6     }
7     public static void main(String[] args) {
8         Boletim boletim = new Boletim();
9         boletim.displayMessage();
10    }
11 }
12
```

Three red boxes with arrows point to specific parts of the code:

- Chamada ao construtor da classe**: Points to the `new` keyword in line 8.
- Instanciando um objeto da classe Boletim**: Points to the `Boletim()` constructor call in line 8.
- Chamando o método sobre o objeto instanciado**: Points to the `boletim.displayMessage();` call in line 9.

Executando

- Ao executar, será exibida a mensagem que definimos:

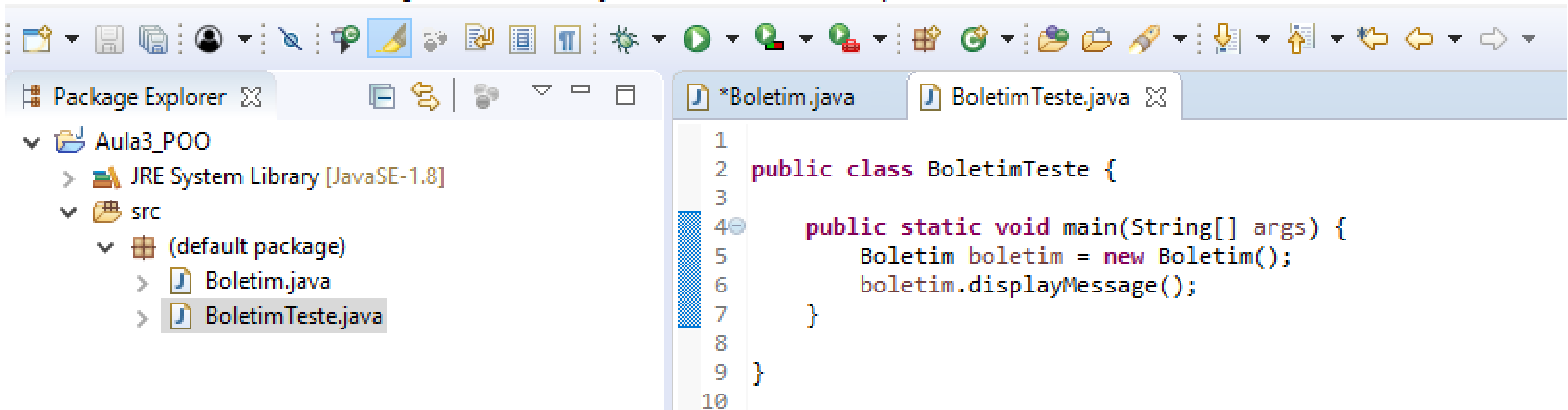


Outra Forma

- Use a classe original (sem o *main*) e declare o método main em uma “classe de teste”, ou **classe driver**
- Chame a nova classe de BoletimTeste e declare-a dentro do mesmo projeto!

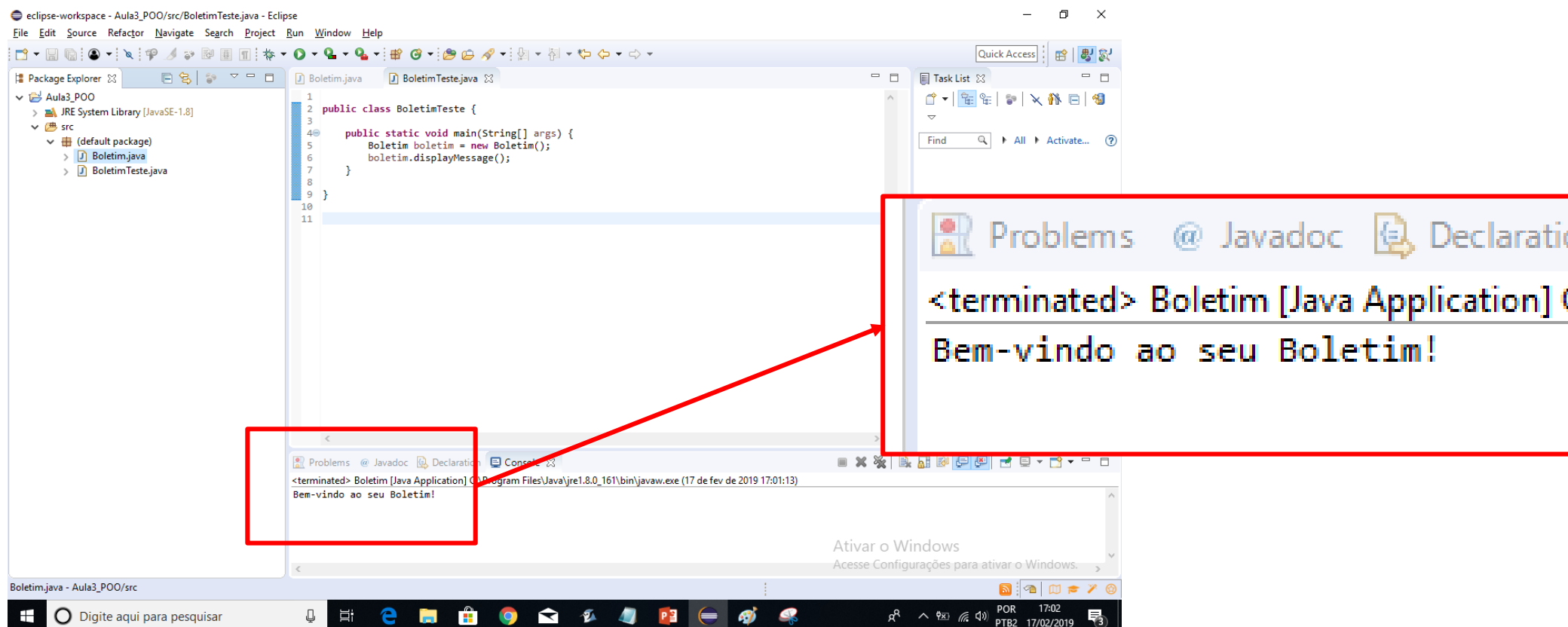
eclipse-workspace - Aula3_POO/src/BoletimTeste.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help



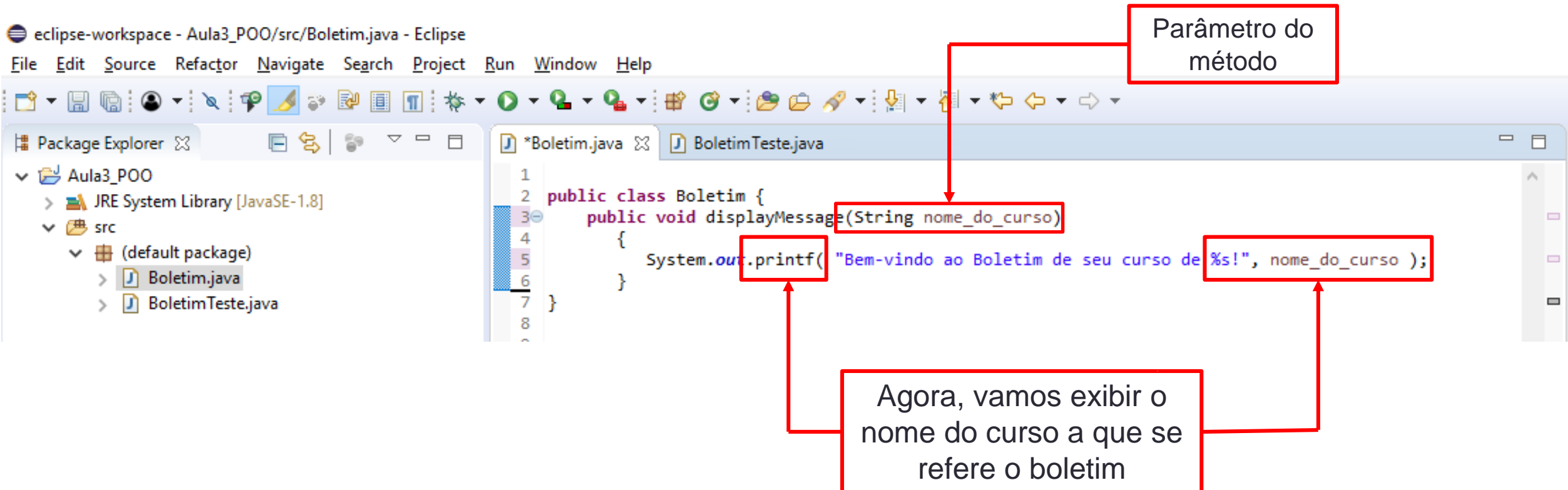
Outra Forma

- Note que transferimos o código do método main para essa classe de teste.
- Compilamos e executamos ambos os arquivos.



Passando Parâmetros para o Método

- Vamos agora reescrever o método para que este admita um parâmetro



Passando Parâmetros para o Método

- As alterações precisarão ser refletidas na classe driver

The screenshot shows a Java IDE with two tabs: `Boletim.java` and `*BoletimTeste.java`. The `BoletimTeste.java` file is open, showing the following code:

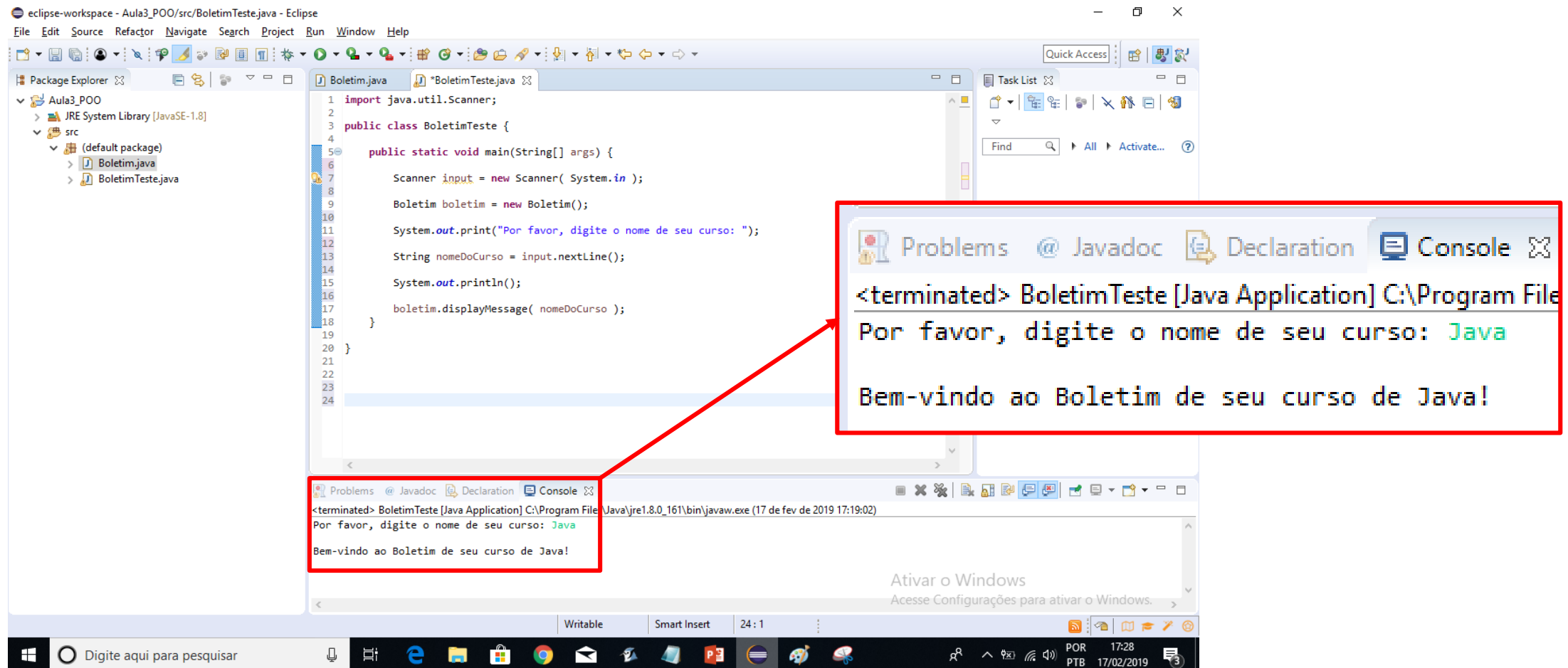
```
1 import java.util.Scanner;
2
3 public class BoletimTeste {
4
5     public static void main(String[] args) {
6         Scanner input = new Scanner( System.in );
7
8         Boletim boletim = new Boletim();
9
10        System.out.print("Por favor, digite o nome de seu curso: ");
11
12        String nomeDoCurso = input.nextLine();
13
14        System.out.println();
15
16        boletim.displayMessage( nomeDoCurso );
17    }
18 }
19
20
21
```

Annotations with red boxes and arrows point to specific lines of code:

- Vamos usar leitura do teclado** points to line 1: `import java.util.Scanner;`
- Objeto input para receber os dados da leitura** points to line 6: `Scanner input = new Scanner(System.in);`
- Variável String para manipular o conteúdo o objeto input** points to line 12: `String nomeDoCurso = input.nextLine();`
- O argumento nomeDoCurso é passado para o método** points to line 16: `boletim.displayMessage(nomeDoCurso);`

Passando Parâmetros para o Método

- Ao executar obteremos como saída:



Variáveis de Instância

- Objeto = *instância* da classe
- Uma variável de instância é um ***campo*** (no objeto) em que se representa um ***atributo*** da classe.

public × private

- São *modificadores de acesso*
- A maioria das declarações de variáveis de instância são **private**
- Variáveis e métodos **private** só são *acessíveis aos métodos da classe em que são declarados*
 - **Objetivo:** *Ocultação (encapsulamento) de dados*

Os Métodos *set* e *get*

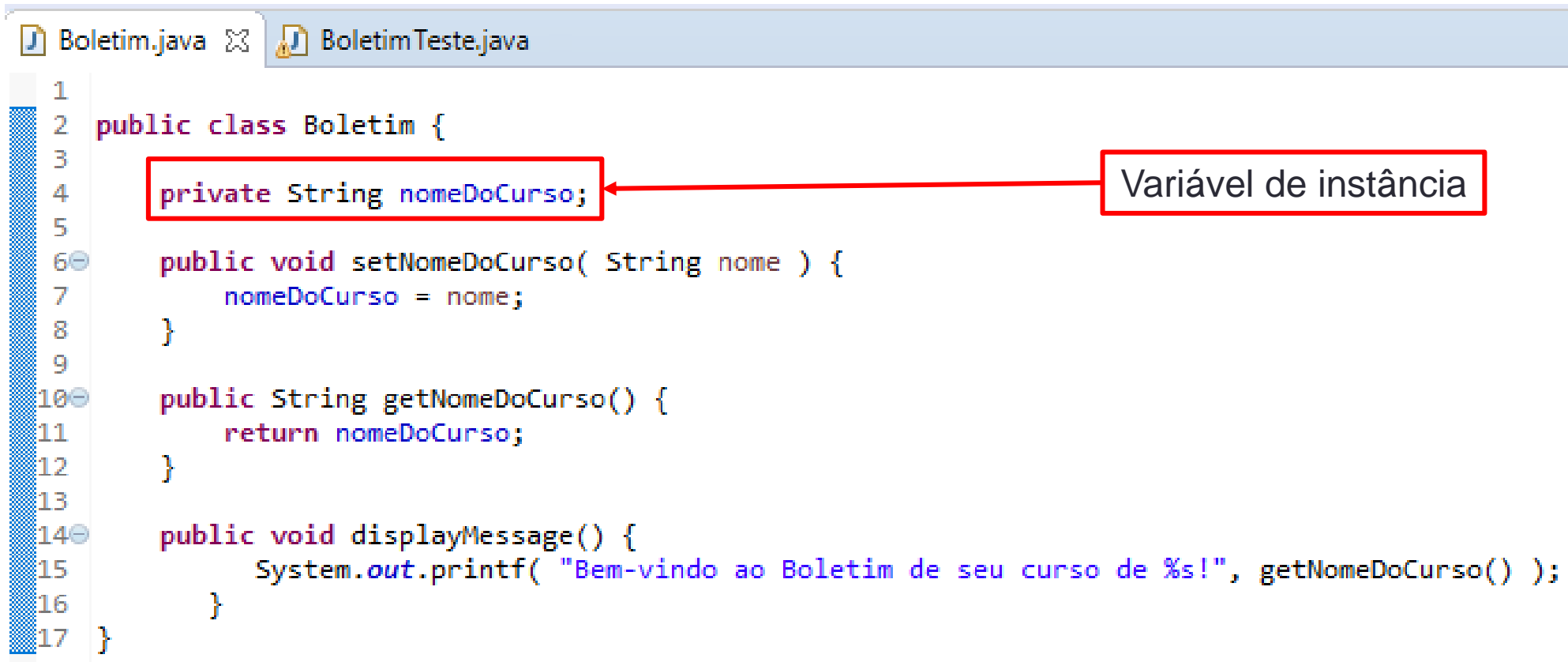
- As classes costumam oferecer métodos **public** para permitir que clientes
 - *Configurem* (*set* – atribuir valor a) **ou**
 - *Obtenham* (*get* – obtenham valores de)*variáveis de instância **private***
- Esses métodos **não precisam** começar com *set* ou *get*, mas esta é uma convenção **recomendada** e requerida para componentes **JavaBeans**.



Componentes de
software reutilizáveis

Exemplificando

- Vamos acrescentar ao nosso programa exemplo uma variável de instância e os métodos set e get



```
1 public class Boletim {
2
3     private String nomeDoCurso;
4
5     public void setNomeDoCurso( String nome ) {
6         nomeDoCurso = nome;
7     }
8
9     public String getNomeDoCurso() {
10        return nomeDoCurso;
11    }
12
13    public void displayMessage() {
14        System.out.printf( "Bem-vindo ao Boletim de seu curso de %s!", getNomeDoCurso() );
15    }
16 }
17 }
```

Variável de instância

Exemplificando

- A classe BoletimTeste tem que refletir as alterações:

```
Boletim.java  *BoletimTeste.java
1 import java.util.Scanner;
2
3 public class BoletimTeste {
4
5     public static void main(String[] args) {
6
7         Scanner input = new Scanner( System.in );
8
9         Boletim boletim = new Boletim();
10
11         System.out.printf("O nome inicial do curso é %s", boletim.getNomeDoCurso() );
12
13         System.out.print("\n\nPor favor, digite o nome de seu curso: ");
14
15         String nome = input.nextLine();
16
17         boletim.setNomeDoCurso( nome );
18
19         System.out.println();
20
21         boletim.displayMessage();
22     }
23
24 }
```

Estamos lendo a
variável de instância
ainda não inicializada

Setando a variável
de instância

Executando...

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with 'Aula3_POO' containing 'src' and 'BoletimTeste.java'.
- Editor:** Displays the code for 'BoletimTeste.java'. The code is as follows:

```
1 import java.util.Scanner;
2
3 public class BoletimTeste {
4
5     public static void main(String[] args) {
6
7         Scanner input = new Scanner( System.in );
8
9         Boletim boletim = new Boletim();
10
11         System.out.printf("O nome inicial do curso é %s", boletim.getNomeDoCurso());
12
13         System.out.print("\n\nPor favor, digite o nome de seu curso: ");
14
15         String nome = input.nextLine();
16
17         boletim.setNomeDoCurso( nome );
18
19         System.out.println();
20
21         boletim.displayMessage();
22     }
23 }
24
25
26
27
28
```
- Console:** Shows the output of the program. It starts with a red line indicating the application has terminated, followed by the text 'O nome inicial do curso é null'. Then, it prompts 'Por favor, digite o nome de seu curso: Java' and finally displays 'Bem-vindo ao Boletim de seu curso de Java!'.

Annotations on the image:

- A purple box points to the console output 'O nome inicial do curso é null' with the text: "Note a leitura da variável não inicializada".
- A red box highlights the console output from the prompt to the final message, with the text: "Diferentemente das variáveis locais, que não são automaticamente inicializadas, todo campo tem um valor inicial padrão".

Tipos Primitivos x Tipos por Referência

- Tipos primitivos: **boolean, byte, char, short, int, long, float, double**
 - Variáveis de instância de tipos primitivos são inicializadas por *default*:
 - byte, char, short, int, long, float e double: inicializadas com 0
 - boolean: inicializadas com *false*
 - **Atenção:** variáveis locais **não são** inicializadas por padrão!
- Todos os tipos *não*-primitivos são **tipos por referência**
 - Portanto, as classes, que especificam os tipos dos objetos, são ***tipos por referência***
 - Os objetos referenciados podem conter muitas variáveis de instância e métodos!

Inicializando Objetos com Construtores

- O construtor é um método especial usado para inicializar os objetos quando estes são criados
- *Java requer uma chamada ao construtor para todo objeto criado*
- Um construtor deve ter o *mesmo nome da classe*
- Por padrão, o compilador fornece um **construtor padrão**, sem parâmetros, para todas as classes quando não definidos explicitamente
 - *Quando uma classe tem somente seu construtor padrão, suas variáveis de instância são inicializadas de acordo com seus valores padrões*
- Você pode fornecer o construtor, a fim de especificar uma inicialização personalizada para os objetos de sua classe

Inicializando Objetos com Construtores

- Vamos acrescentar um construtor à nossa classe exemplo:

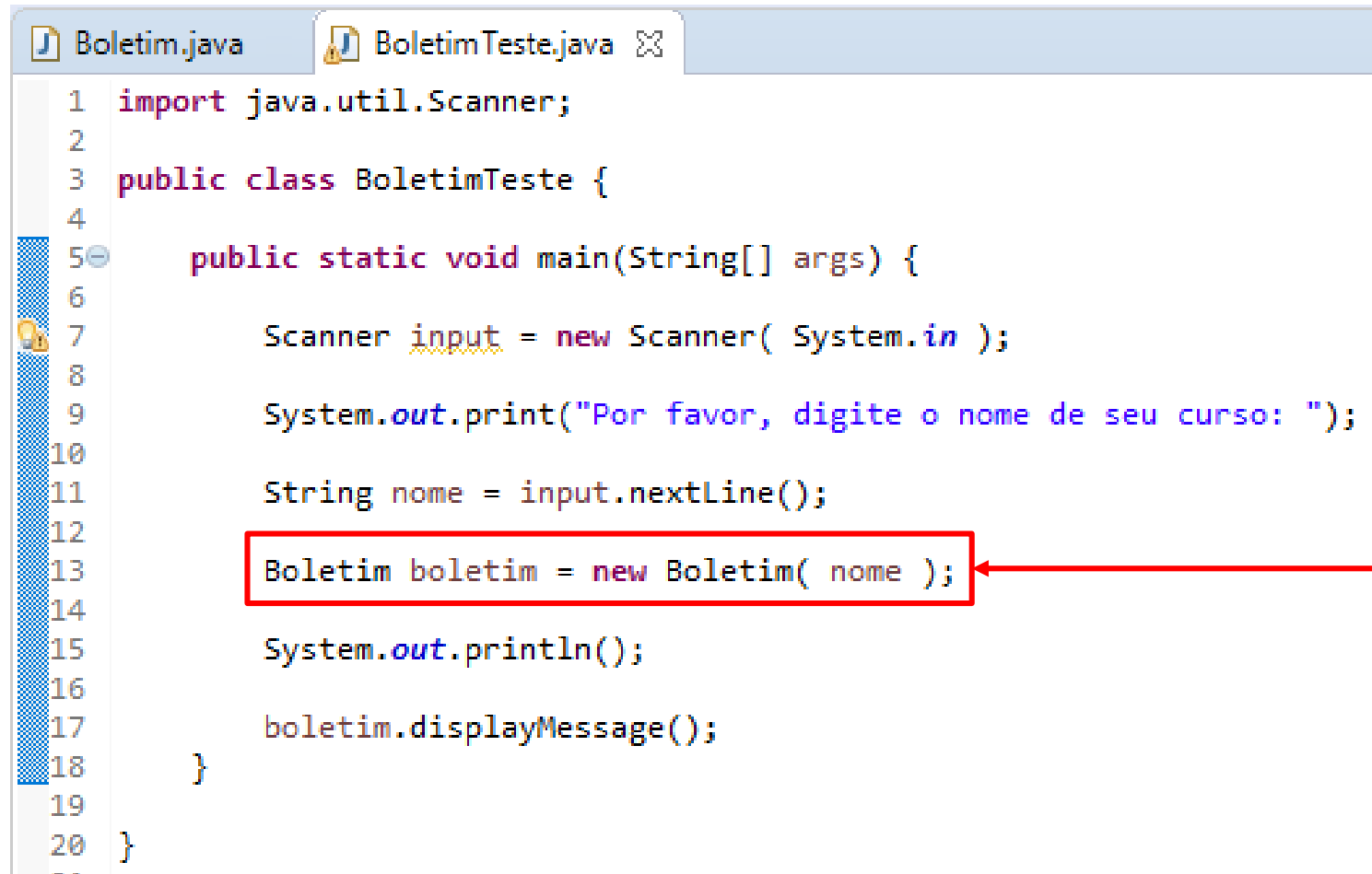
```
1 public class Boletim {
2
3
4     private String nomeDoCurso;
5
6     public Boletim( String nome ) {
7         nomeDoCurso = nome;
8     }
9
10    public void setNomeDoCurso( String nome ) {
11        nomeDoCurso = nome;
12    }
13
14    public String getNomeDoCurso() {
15        return nomeDoCurso;
16    }
17
18    public void displayMessage() {
19        System.out.printf( "Bem-vindo ao Boletim de seu curso de %s!", getNomeDoCurso() );
20    }
21 }
22
```

Note: mesmo nome da classe!

Construtor

Inicializando Objetos com Construtores

- Refletindo as alterações na classe driver



```
1 import java.util.Scanner;
2
3 public class BoletimTeste {
4
5     public static void main(String[] args) {
6
7         Scanner input = new Scanner( System.in );
8
9         System.out.print("Por favor, digite o nome de seu curso: ");
10
11         String nome = input.nextLine();
12
13         Boletim boletim = new Boletim( nome );
14
15         System.out.println();
16
17         boletim.displayMessage();
18     }
19
20 }
```

Chamada ao
construtor com o
argumento

Construtores

- Não podem retornar valores
 - Não especificam um tipo de retorno, *nem mesmo void*
 - São normalmente declarados **public**
 - Se uma classe não incluir um construtor, as variáveis de instância são inicializadas por seus valores padrões
 - *Se você declarar qualquer construtor para uma classe, o compilador Java **não** criará um construtor padrão para essa classe*
- *Forneça sempre um construtor para garantir que as variáveis de instância sejam adequadamente inicializadas!*

GUI – Graphical User Interfaces

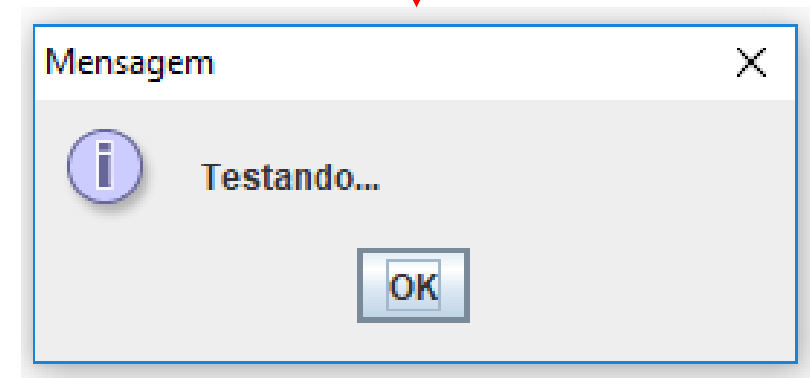
- Vamos testar o uso de uma janela para interação com o usuário.
- Abra um novo projeto. Chame de algo como “Aula3_grafico”, ou algo assim.
- Neste projeto, defina a seguinte classe:

```
1
2 import javax.swing.JOptionPane;
3
4 public class Janela {
5     public static void main(String[] args) {
6         JOptionPane.showMessageDialog(null, "Testando...");
7     }
8 }
```

Fornece
caixas de
diálogo

Exibe a janela
no centro da tela

Ao executar, teremos como
saída a janela abaixo



Observação: `JOptionPane.showMessageDialog` é um **método static**.

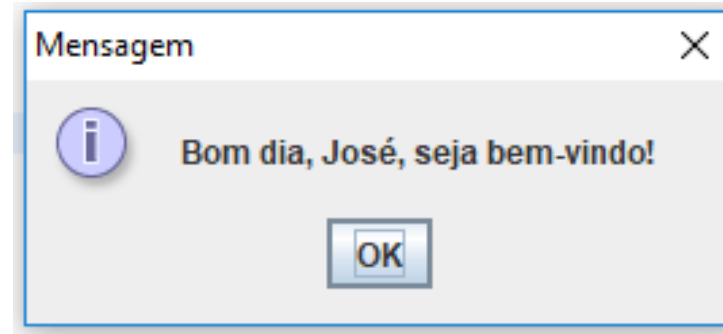
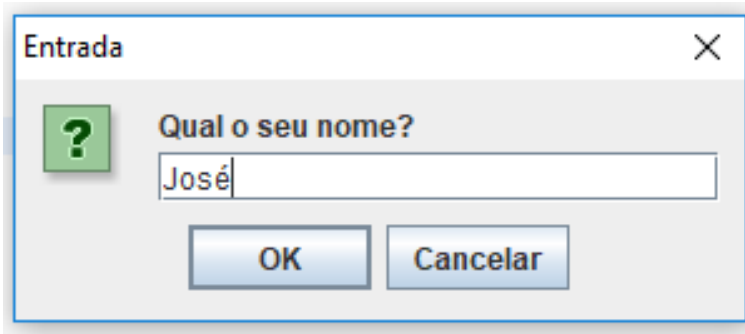
➤ *Métodos static* definem tarefas que são realizadas muitas vezes!

Inserindo Texto na Caixa de Diálogo

- Vamos agora usar uma janela para a entrada de um texto.
- Escreva a seguinte classe:

```
1
2 import javax.swing.JOptionPane;
3
4 public class Janela {
5     public static void main(String[] args) {
6
7         String nome = JOptionPane.showInputDialog( "Qual o seu nome?" );
8
9         String mensagem = String.format( "Bom dia, %s, seja bem-vindo!", nome );
10
11         JOptionPane.showMessageDialog(null, mensagem );
12     }
13 }
```

Obs: Se você pressionar o botão “cancelar” ou a tecla “esc”, o programa exibirá “null” como nome.



Dúvidas?