

# Programação Orientada a Objetos

Aula 2 – Introdução a Java  
Classes e Construtores em Java

Ana Patrícia F. Magalhães Mascarenhas  
anapatriciamagalhaes@gmail.com  
apmagalhaes@uneb.br

- Objetivo
  - Introduzir a linguagem Java
  - Entender a representação de classes e objetos em Java
  - Aprender a criar objetos em Java
- Bibliografia básica
  - BARNES, J. David, KÖLLING, Michael. **Programação Orientada a Objetos com Java**. Pearson, 2004.
  - HORSTMANN, Gay S., CORNELL, Gary. **Core Java 2. Volume I – Fundamentos**. Makron Books, 2000.
  - Santos, Rafael. **Introdução a Programação Orientada a Objetos com Java**. Ed. Campos, 2. ed., 2013

# Criando classes em Java

## Sintaxe básica

Uma classe em Java será declarada com a palavra-chave `class` seguida do nome da classe.

- O nome não pode conter espaços
- Deve começar com uma letra
- Deve ser diferente das palavras reservadas
- Caracteres maiúsculos e minúsculos são diferenciados
- Conteúdo da classe limitado pelas chaves `{ }`

# Criando classes em Java

## Campos de classes em Java

Os campos de classes em Java devem ser declarados dentro do corpo da classe.

- Cada campo deve ser representado por um determinado tipo de dado.
- Em linguagens POO, é possível declarar campos como referências a instâncias de outras classes já existentes.

# Criando classes em Java

## Dados nativos em Java

Java provê tipos primitivos divididos em quatro grandes categorias:

- *Inteiros* (n<sup>os</sup> discretos): **byte** (8 bits), **short** (16 bits), **int** (32 bits) e **long** (64 bits).
- *Floating Point* (n<sup>os</sup> contínuos): **float** (32 bits) e **double** (64 bits)
- *Character*: **char** (16 bits)
- *Boolean*: **boolean**

A classe String é usada para representar cadeias de caracteres.  
(Não são dados nativos, sendo instâncias da classe String)

# Criando classes em Java

## Dados nativos em Java (cont.)

As operações básicas que podem ser feitas com o tipo numérico:

- + Soma
- - Subtração
- / Divisão
- \* Multiplicação
- % Módulo (resto da divisão)

# Criando classes em Java

## Dados nativos em Java (cont.)

Valores numéricos podem ser comparados com operadores que retornam um valor do tipo boolean. Os operadores são:

- < (menor)
- > (maior)
- <= (menor ou igual)
- >= (maior ou igual)
- == (igual)
- != (diferente)

# Criando classes em Java

## Dados nativos em Java (cont.)

Valores booleanos podem ser combinados com três operadores lógicos. Que são:

- && (E lógico)
- || (OU lógico)
- ! (NÃO lógico)

Operação com instâncias da classe String.

- + (concatenação)

Strings não podem ser comparados com os operadores >, <, ==.



# Criando classes em Java

## Declaração de campos de classes:

A declaração é simples, basta declarar o tipo de dado, seguindo dos nomes dos campos que são daquele tipo.

Podemos observar que, para cada dado do modelo existe um campo correspondente na classe.

Exemplo:

```
class AparelhoEletronico {  
  
    String nome, fornecedor;  
  
    int voltagem, numeroSerie;  
  
    String situação;  
  
}
```

# Criando classes em Java

## Declaração de métodos em classes:

A maioria das classes representa modelos que tem dados e operações que manipulam esses dados.

As operações são chamadas de métodos.

Métodos não podem ser criados dentro de outros métodos, nem fora da classe à qual pertencem.

# Exemplo

```
class AparelhoEletronico
{
    String nomeAparelho, fornecedor;
    float voltagem;
    int numeroDeSerie
    String situacão;

    //Operacao criarAparelho, para inicializar os dados do aparelho
    public void criarAparelhoEletronico(String nome, String fornec, float volt, int serie) // argumentos para operação
    {
        nomeAparelho = nome;
        fornecedor = fornec;
        voltagem = volt;
        numeroDeSerie = serie;
        situacao = "desligado";
    }
    // operação para ligar o aparelho
    public void ligar()
    {
        sistuacao = "ligado";
    }
}
```

# Mais um pouco de Java: Estruturas de decisão e controle

- if – else

Estrutura do Comando

```
if (expressão_booleana )  
{ bloco de comandos do if;  
}  
[else]  
{ bloco de comandos do else;  
}
```

- Observações:

- O comando *else* é opcional.
- Na construção de *if*'s aninhados, o *else* refere-se sempre ao *if* mais próximo. Procure usar chaves para delimitar os blocos.
- O operador de comparação **igual a** é representado por `==` e não por `=`.
- Bloco de comandos pode ser um único comando (terminado por ponto-e-vírgula) ou vários comandos (delimitados por `{}`).
- É indispensável o uso de parênteses ( ) na expressão\_booleana.

# Mais um pouco de Java: Estruturas de decisão e controle

- switch

Estrutura do  
Comando

```
switch(variávelDeControle)
{ case constante1:
    bloco1;
    break;
  case constante2:
    bloco2;
    break;
  [default:
    bloco3;
    break;]
}
```

- Observações:

- Usual para selecionar alguma ação de um número de alternativas.
- A variável de controle só pode ser inteira ou char.
- O case define o ponto de entrada da execução. Se você quiser que só um bloco de declarações seja executado use **break**.
- A opção **default** é opcional.

# Mais um pouco de Java: Contadores

Variáveis que recebem um valor inicial e são incrementadas a cada interação de uma repetição.

`contador = contador + 1;`

- **Exemplos:**

- ++

incrementa o valor em 1

`A = 5; A++;` => A vale 6

Pode ser colocado antes ou depois da variável

`A = 5;`

`B = A++;` => B vale 5 e A vale 6

`B = ++A;` => B e A valem 6

- +=

soma o valor à variável

`A = 5; A += 3` => A vale 8

- --

decrementa o valor de 1

`A = 5; A--;` => A vale 4

# Mais um pouco de Java: Contadores

- -=            subtrai o valor à variável  
A = 5;    A -= 3    => A vale 2
- \*=            multiplica a variável pelo valor  
A = 5;    A \*= 2;    => A vale 10
- /=            divide a variável pelo valor  
A = 10;    A /= 2    => A vale 5

# Mais um pouco de Java: Estruturas de Repetição

- while

Estrutura do  
Comando

```
while (expressão_booleana)
{ Bloco de comandos;
}
```

## Observações:

- É indispensável o uso de parênteses ( ) na expressão\_booleana.
- O laço permanece em execução enquanto a expressão\_booleana for verdadeira.
- Um erro comum é não atualizar as variáveis de controle do laço, o que acarreta um *loop* infinito.
- Outro erro freqüente é colocar o ponto e vírgula após o while. A fase de atualização das variáveis de controle ficam fora do laço, gerando um *loop* infinito.
- Use sempre chaves para delimitar o Bloco de comandos.
- Break força a saída do laço



# Mais um pouco de Java: Estruturas de repetição

- do - while

Estrutura do  
Comando

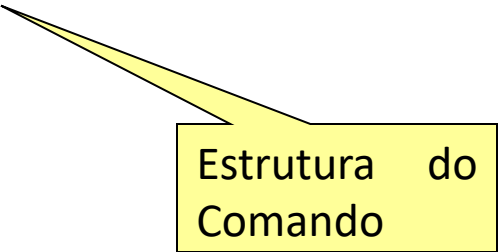
do  
{ Bloco de comandos;  
} while (expressão\_booleana);

- Observações:

- É semelhante ao comando while, sendo que a condição de parada do laço é testada após o bloco de comandos.
- Pelo menos uma vez o bloco de comandos será executado.
- Observe as mesmas considerações do comando while.

# Mais um pouco de Java: Estruturas de repetição

- **for**            **for(inicialização; terminação; iteração)**  
                  **{ bloco de comandos;**  
                  **}**



Estrutura do  
Comando

- **Observações:**
  - Num *loop* for é explicita as quatro partes de uma iteração.
  - Um erro comum é colocar o ponto e vírgula após o for, ficando o bloco de comandos fora do laço. O resultado é um *loop* que nada realiza.

# Mais um exemplo

```
class DataSimples {  
    byte dia,mes;  
    short ano;  
    void inicializaDataSimples(byte d, byte m, short a){  
        if (dataEValida(d,m,a)){  
            dia=d; mes=m; ano=a;}  
        else {  
            dia=0;mes=0;ano=0;}  
        }  
    boolean dataEValida(byte d, byte m, short a){  
        if ((d >=1) && (d <=31) && (m>=1) && (m<=12))  
            return true;  
        else  
            return false;  
        }  
    }  
}
```

# Construtor

- Construtor é um tipo especial de método de classe chamados automaticamente quando instâncias são criadas através da palavra chave **new**
- Construtores são úteis para:
  - inicializar os atributos de uma classe
  - realizar rotinas complexas de inicialização
  - realizar inicializações segundo parâmetros passados no momento da criação do objeto

# Declarando um construtor com Java

- Na declaração de um construtor deve-se considerar que:
  - Construtores devem ter **exatamente** o mesmo nome da classe
  - Construtores não possuem tipo de retorno
  - Construtores são, normalmente, públicos

# Voltando ao nosso exemplo

```
class AparelhoEletronico
```

```
{
```

```
    String nomeAparelho, fornecedor;
```

```
    float voltagem;
```

```
    int numeroDeSerie
```

```
    String situacão;
```

```
//Operacao criarAparelho, para inicializar os dados do aparelho
```

```
public AparelhoEletronico(String nome, String fornec, float volt, int serie) // argumentos para operação
```

```
{    nomeAparelho = nome;
```

```
    fornecedor = fornec;
```

```
    voltagem = volt;
```

```
    numeroDeSerie = serie;
```

```
    situacao = "desligado";
```

```
}
```

```
// operação para ligar o aparelho
```

```
public void ligar()
```

```
{
```

```
    sistuacao = "ligado";
```

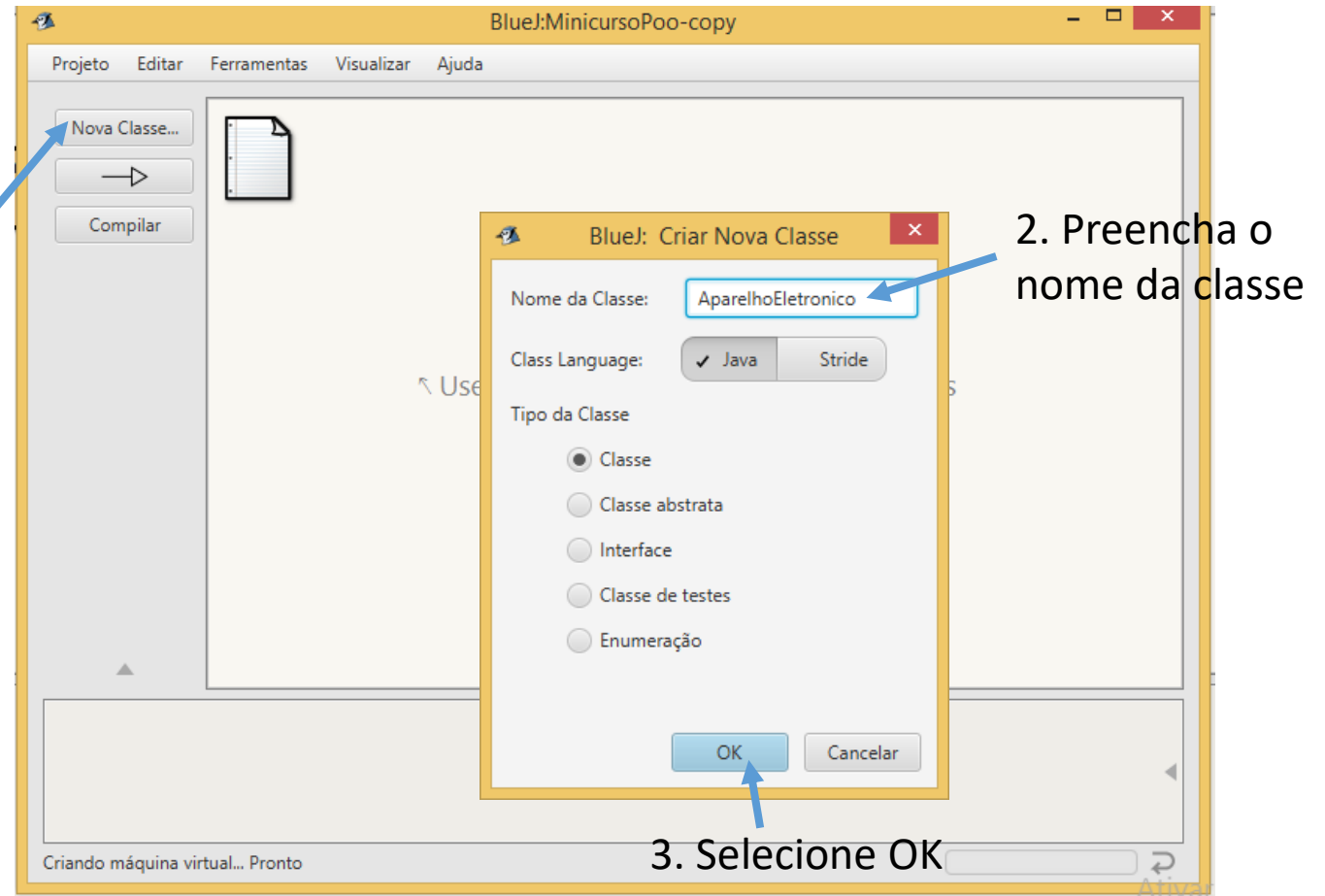
```
}
```

```
}
```

# Prática

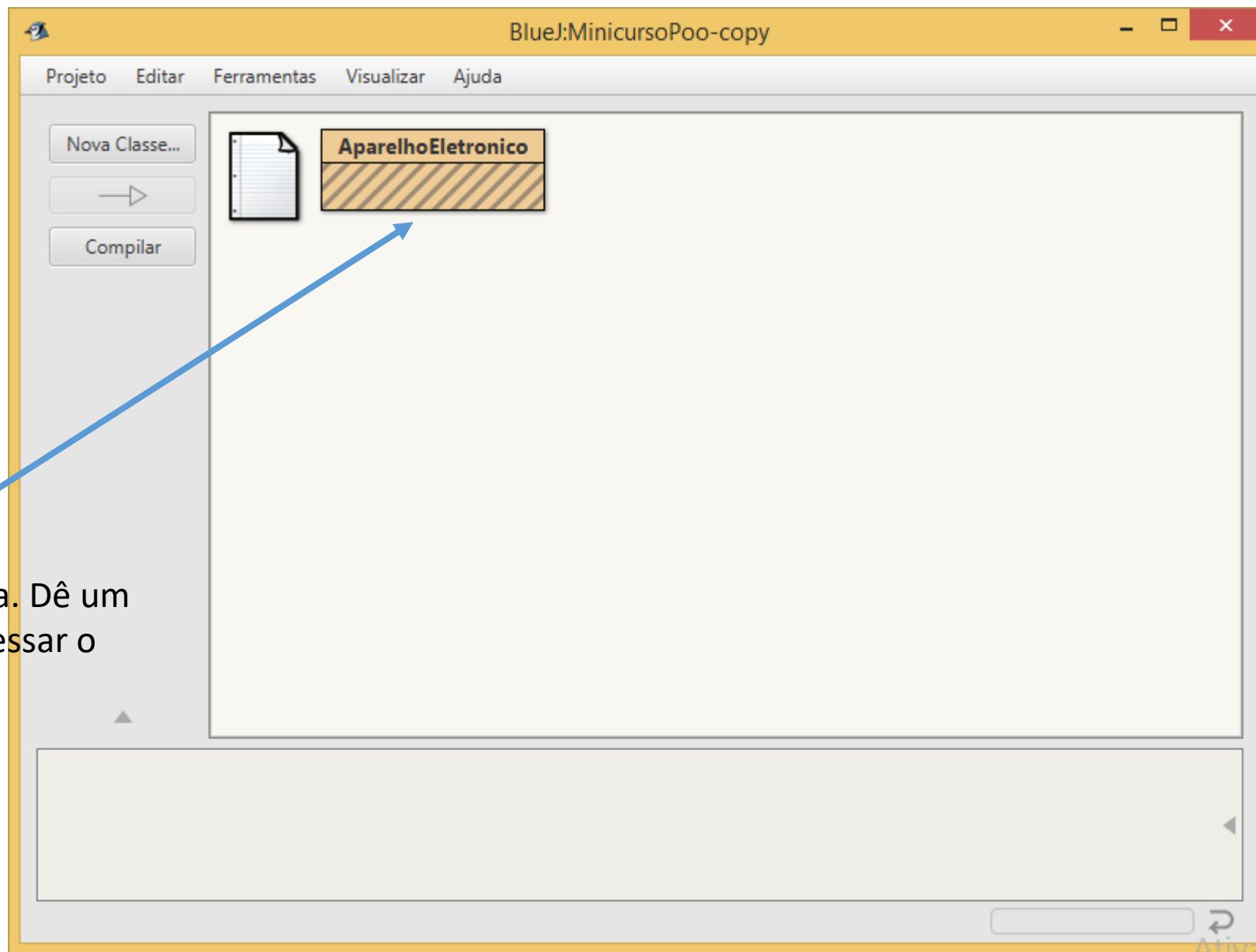
- Vamos utilizar a IDE BlueJ
- [www.bluej.org](http://www.bluej.org)

1. Selecione para criar uma nova classe



# Prática

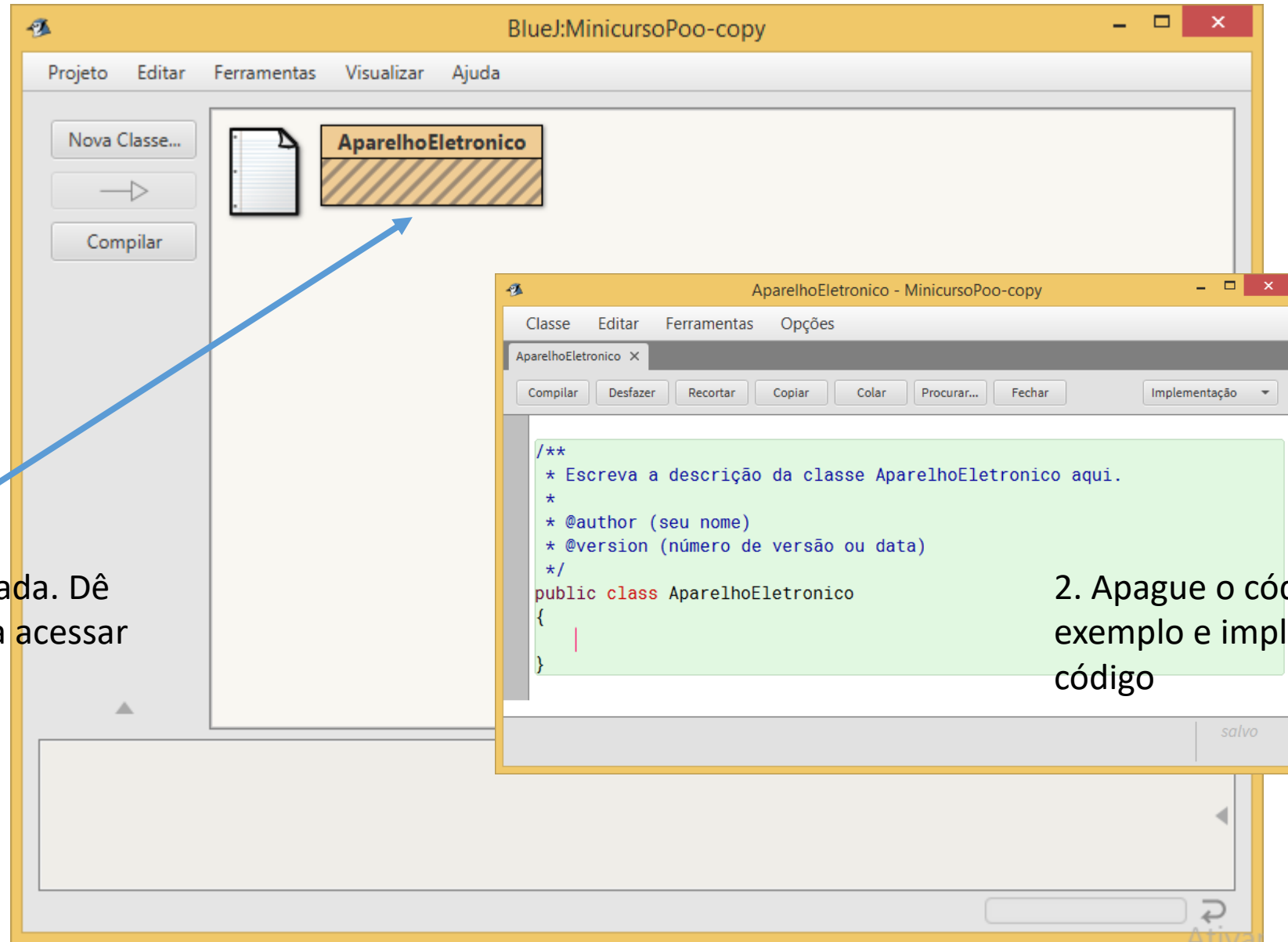
A classe já está criada. Dê um duplo clique para acessar o código





# Prática

1. A classe já está criada. Dê um duplo clique para acessar o código

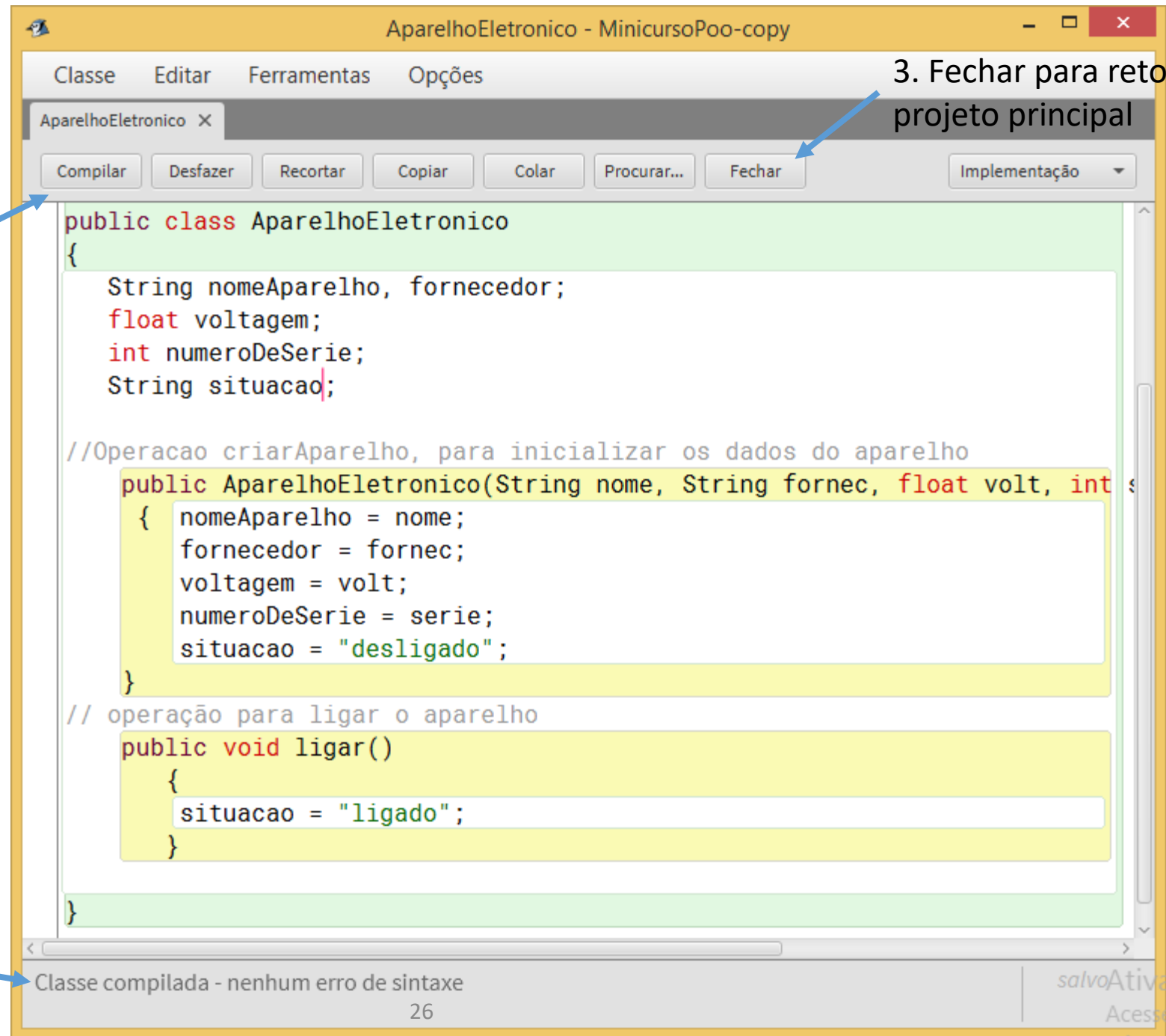


# Prática

1. Compilar

2. Mensagem de erro ou de sucesso

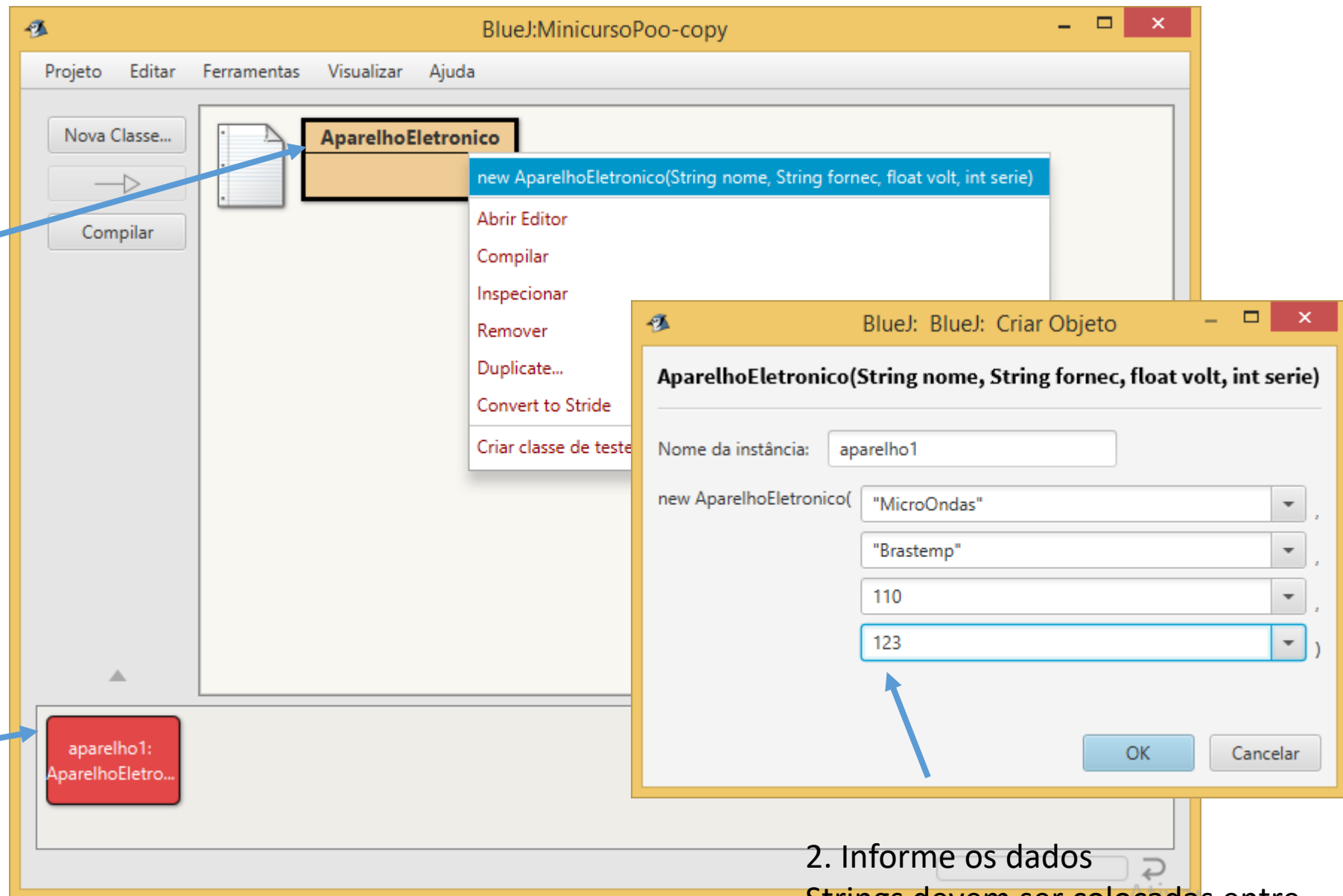
3. Fechar para retornar ao projeto principal



# Prática

1. Com o botão direito do mouse selecione new AparelhoEletronico

3. O objeto criado fica representado nesta caixa vermelha. Com o botão direito do mouse execute os métodos criados.



2. Informe os dados  
Strings devem ser colocadas entre aspas duplas, char entre aspas simples e float com f no final (ex. 0.5f)

# Exercícios

Construa uma classe que represente os alunos de uma universidade, conforme modelo abaixo:

Aluno
<ul style="list-style-type: none"><li>- matricula : int</li><li>- nome : String</li><li>- anoIngresso : int</li><li>- curso : String</li><li>- matriculado : boolean</li><li>- disciplinasMatriculadas : int</li></ul>
<ul style="list-style-type: none"><li>+ criar() : void</li><li>+ matricular(disciplinas : int) : void</li><li>+ imprimir() : void</li><li>+ calcularPermanencia(int ano : int) : int</li><li>+ calcularMensalidade() : float</li></ul>

# Exercícios para casa

Baixar a Ide BlueJ

Resolver a lista de exercícios nr. 1