

# Técnicas de Programação I



Curso Superior de Tecnologia em Desenvolvimento de  
Software Multiplataforma

Aula 05

Prof. Claudio Benossi

# Correção dos Exercícios

# Correção do Exercício Torneio

- ▶ Crie uma classe de nome Torneio conforme diagrama e criar classe com void main para instanciar objetos:

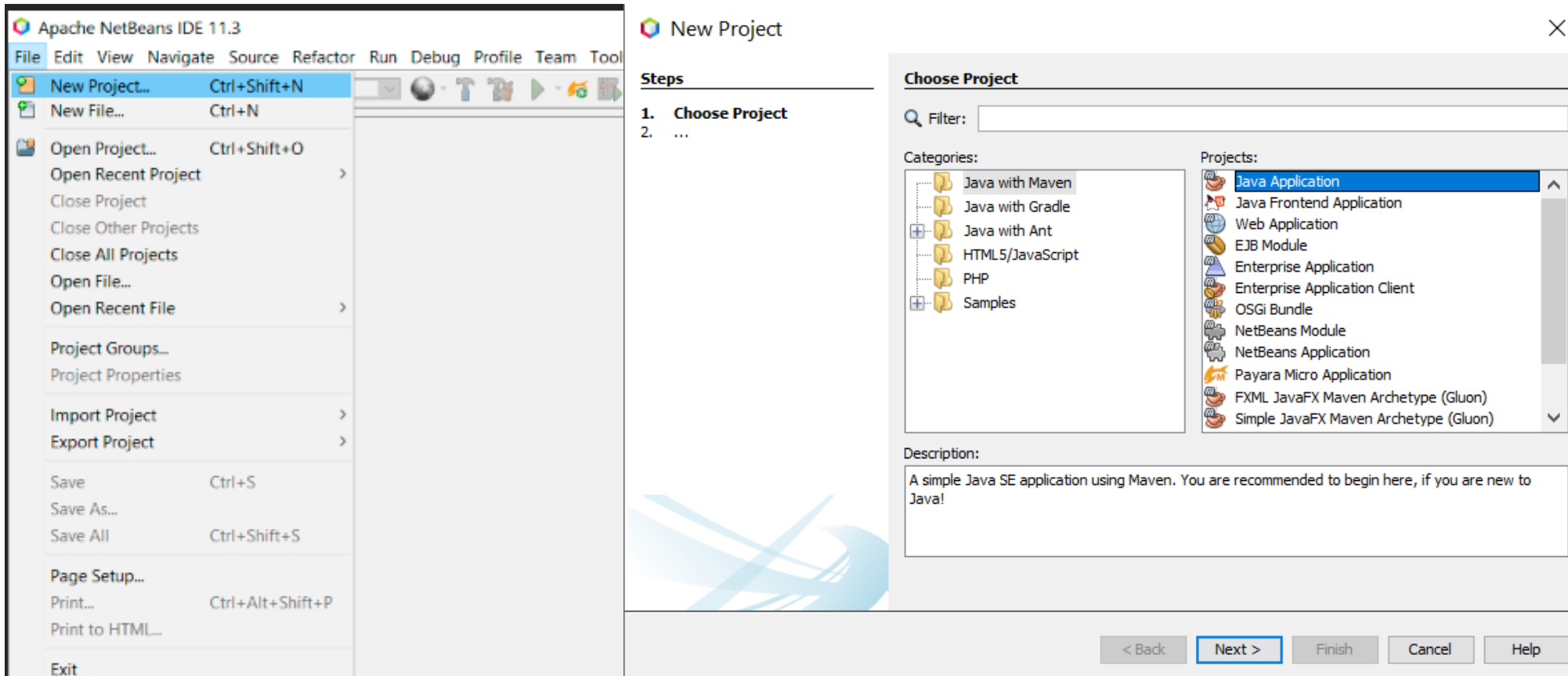
Torneio
- nome: string - idade: int
Torneio(nome: string, idade: int) getNome(): string getIdade(): int setNome(n: string): void setIdade(i: int): void verificaCategoria(): string imprimeDados(): void

- a) Crie os sets e gets para cada um dos atributos;
- b) Crie um método imprimirDados que imprime o estado do objeto inclusive sua categoria;
- c) O método verificarCategoria que deverá retornar qual a categoria do atleta baseado na tabela abaixo:

Categoria	Idade
Infantil	5 a 7
Juvenil	8 a 10
Adolescente	11 a 15
Adulto	16 a 30
Sênior	Acima de 30

# Correção do Exercício Torneio

## Novo Projeto



Apache NetBeans IDE 11.3

File Edit View Navigate Source Refactor Run Debug Profile Team Tool

New Project... Ctrl+Shift+N

New File... Ctrl+N

Open Project... Ctrl+Shift+O

Open Recent Project >

Close Project

Close Other Projects

Close All Projects

Open File...

Open Recent File >

Project Groups...

Project Properties

Import Project >

Export Project >

Save Ctrl+S

Save As...

Save All Ctrl+Shift+S

Page Setup...

Print... Ctrl+Alt+Shift+P

Print to HTML...

Exit

New Project

Steps

1. Choose Project
2. ...

Choose Project

Filter:

Categories:

- Java with Maven
- Java with Gradle
- Java with Ant
- HTML5/JavaScript
- PHP
- Samples

Projects:

- Java Application
- Java Frontend Application
- Web Application
- EJB Module
- Enterprise Application
- Enterprise Application Client
- OSGi Bundle
- NetBeans Module
- NetBeans Application
- Payara Micro Application
- FXML JavaFX Maven Archetype (Gluon)
- Simple JavaFX Maven Archetype (Gluon)

Description:

A simple Java SE application using Maven. You are recommended to begin here, if you are new to Java!

< Back Next > Finish Cancel Help

# Correção do Exercício Torneio

New Java Application

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name:

Project Location:

Project Folder:

Artifact Id:

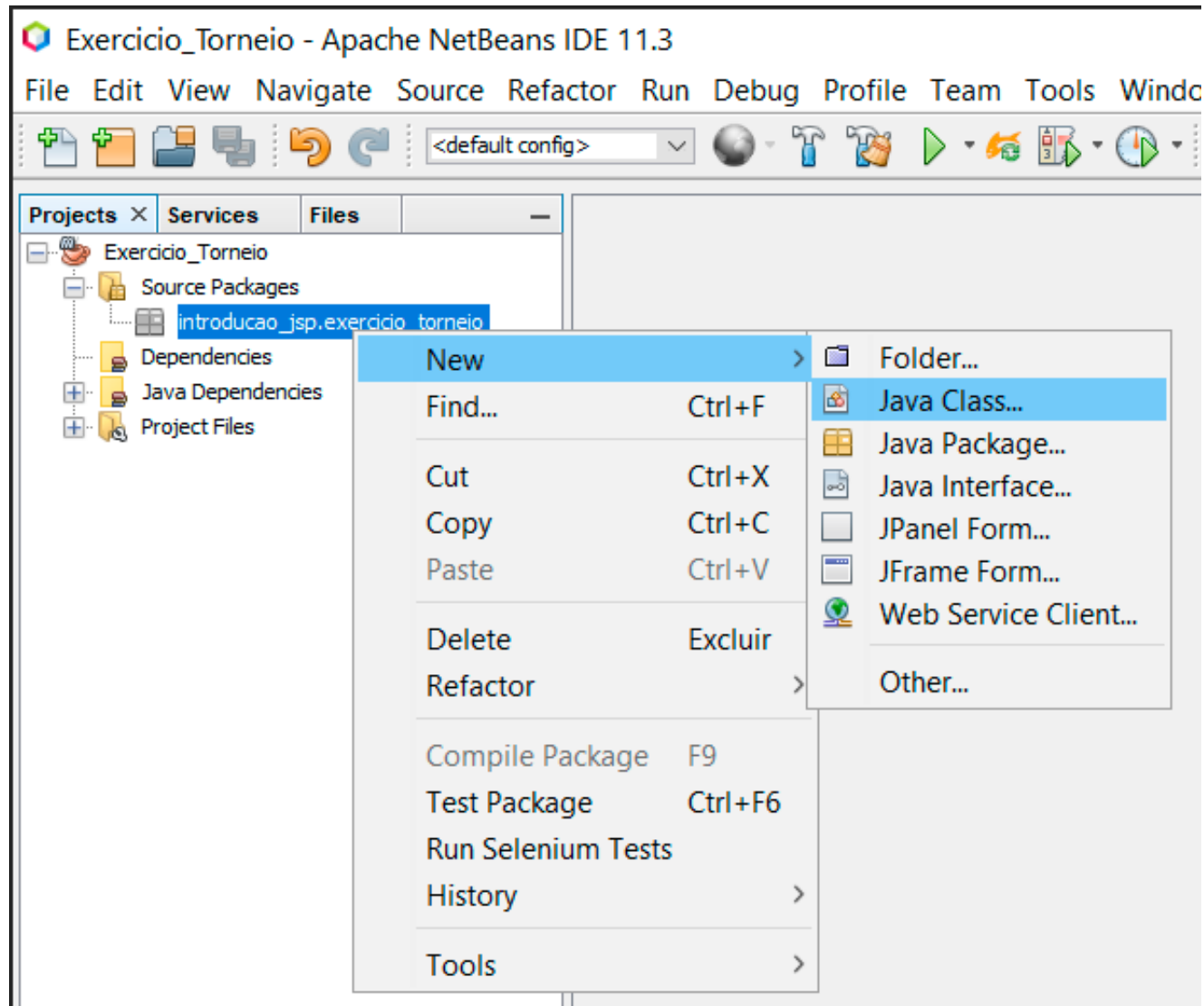
Group Id:

Version:

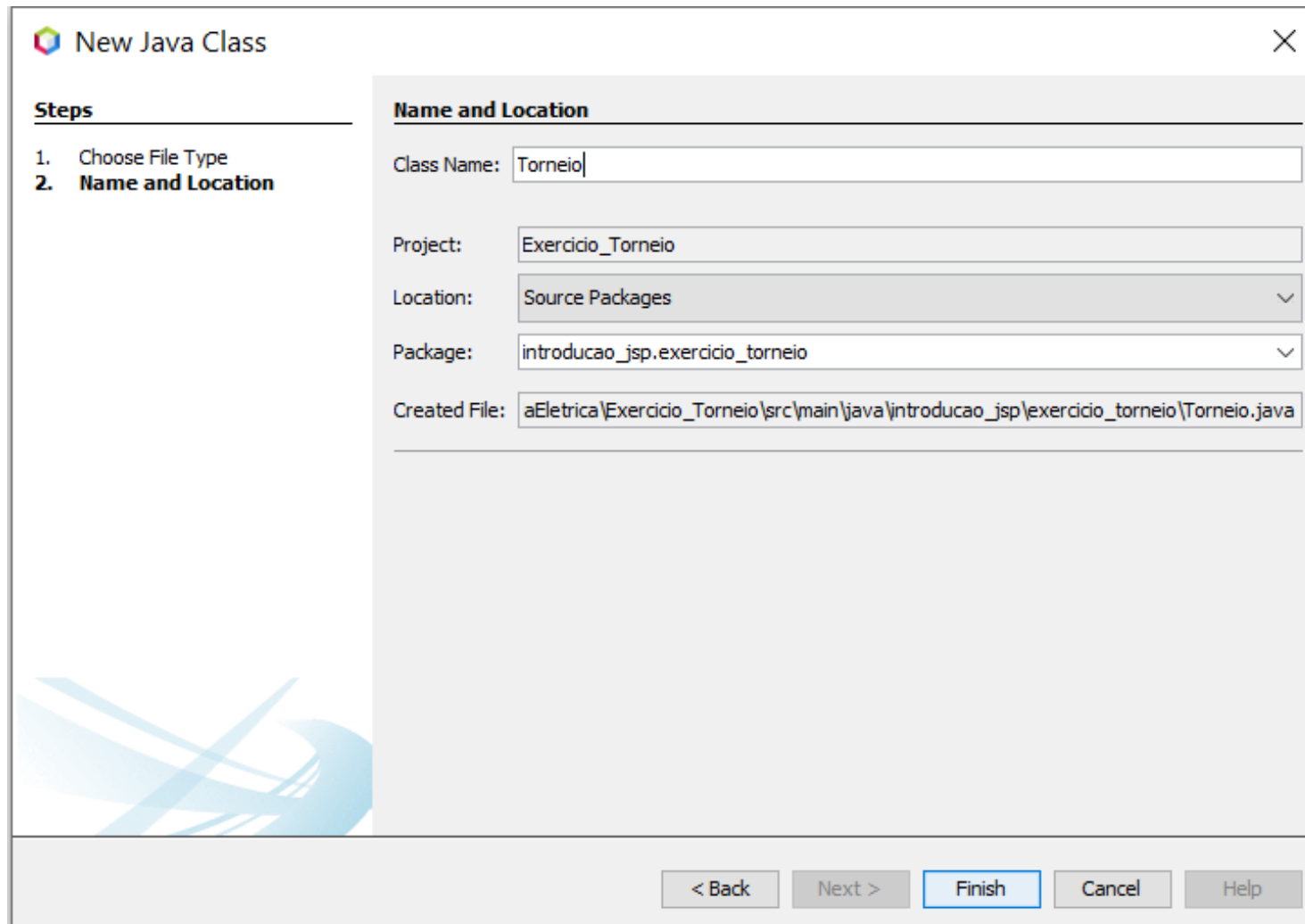
Package:  (Optional)

< Back   Next >   **Finish**   Cancel   Help

# Correção do Exercício Torneio



# Correção do Exercício Torneio



**New Java Class**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

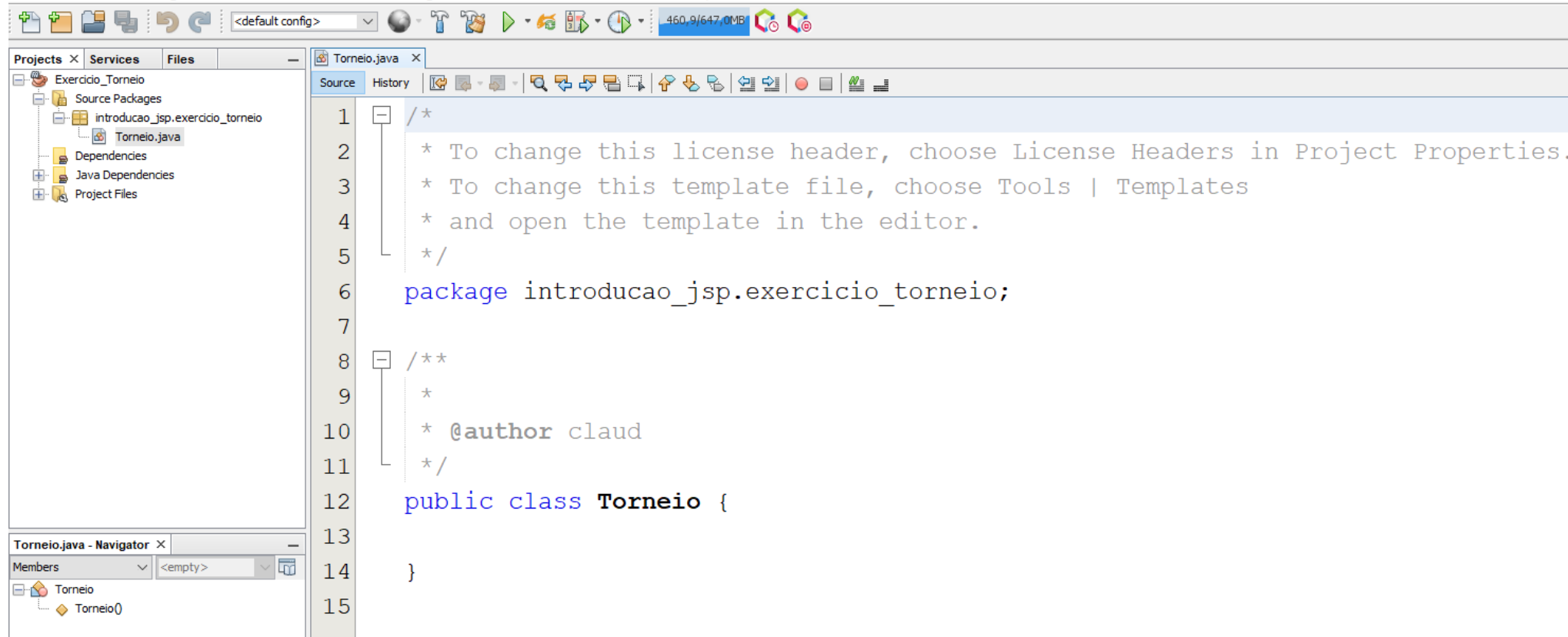
Created File:

< Back   Next >   **Finish**   Cancel   Help

# Correção do Exercício Torneio

Exercicio\_Torneio - Apache NetBeans IDE 11.3

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help



The screenshot displays the Apache NetBeans IDE interface. The main editor window shows the source code of the file `Torneio.java`. The code includes a license header, a package declaration, and the start of a `Torneio` class. The left sidebar shows the project structure with the following components:

- Exercicio\_Torneio
  - Source Packages
    - introducao\_jsp.exercicio\_torneio
      - Torneio.java
  - Dependencies
  - Java Dependencies
  - Project Files

The bottom-left pane, titled "Torneio.java - Navigator", shows the class structure:

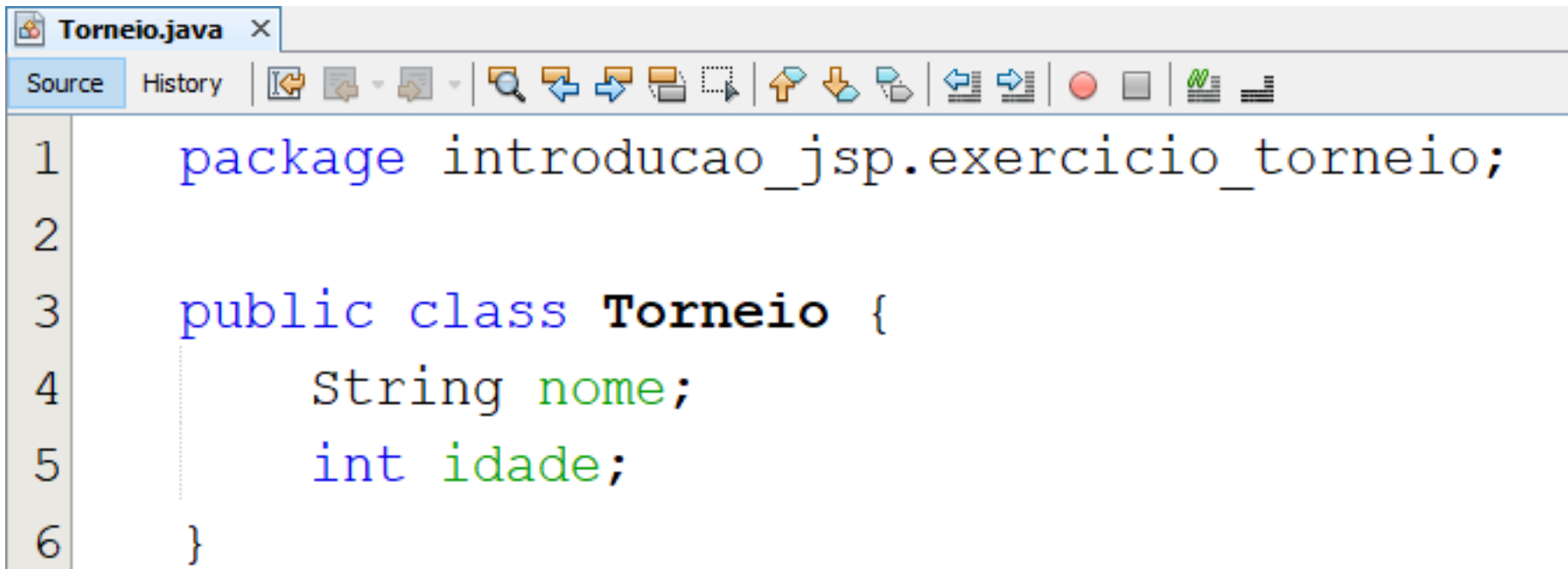
- Members
  - Torneio
    - Torneio()

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package introducao_jsp.exercicio_torneio;
7
8  /**
9   *
10   * @author claud
11   */
12  public class Torneio {
13
14  }
15
```



# Correção do Exercício Torneio

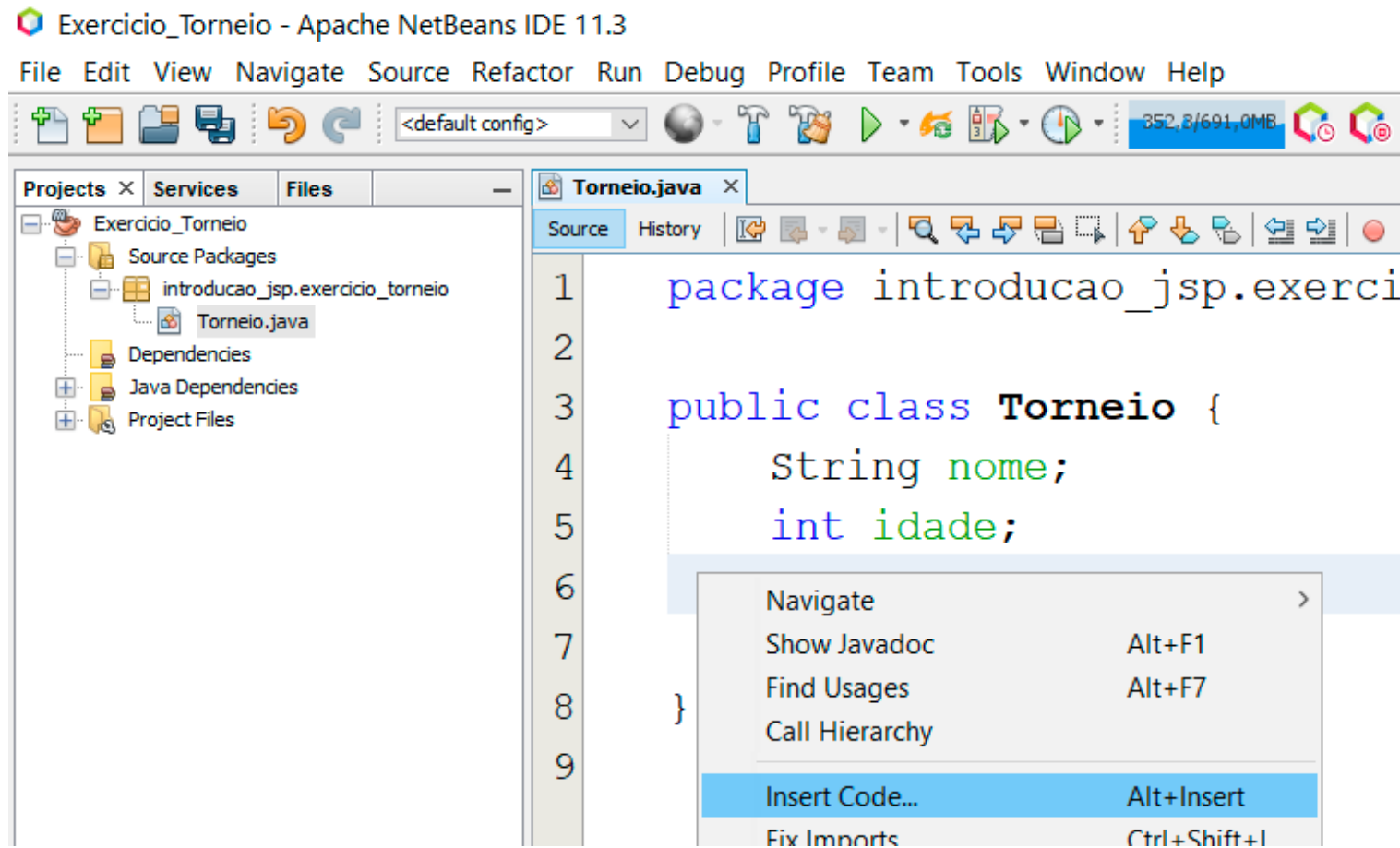
Vamos limpar o código e inserir os atributos a serem utilizados:



```
1 package introducao_jsp.exercicio_torneio;
2
3 public class Torneio {
4     String nome;
5     int idade;
6 }
```

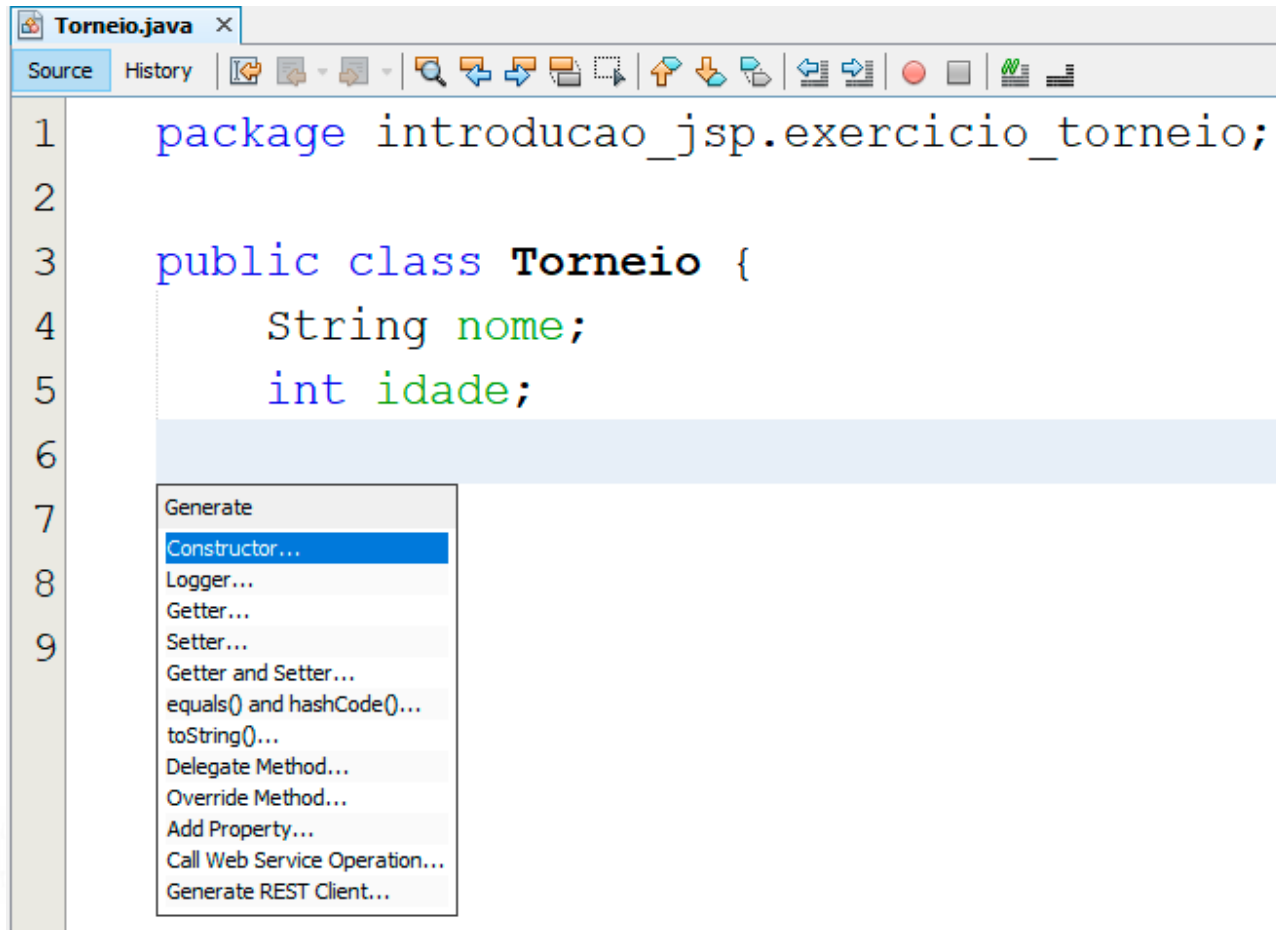
# Correção do Exercício Torneio

Vamos criar o método construtor, para isso selecione o botão direito do mouse e selecione a opção Insert Code ...:



# Correção do Exercício Torneio

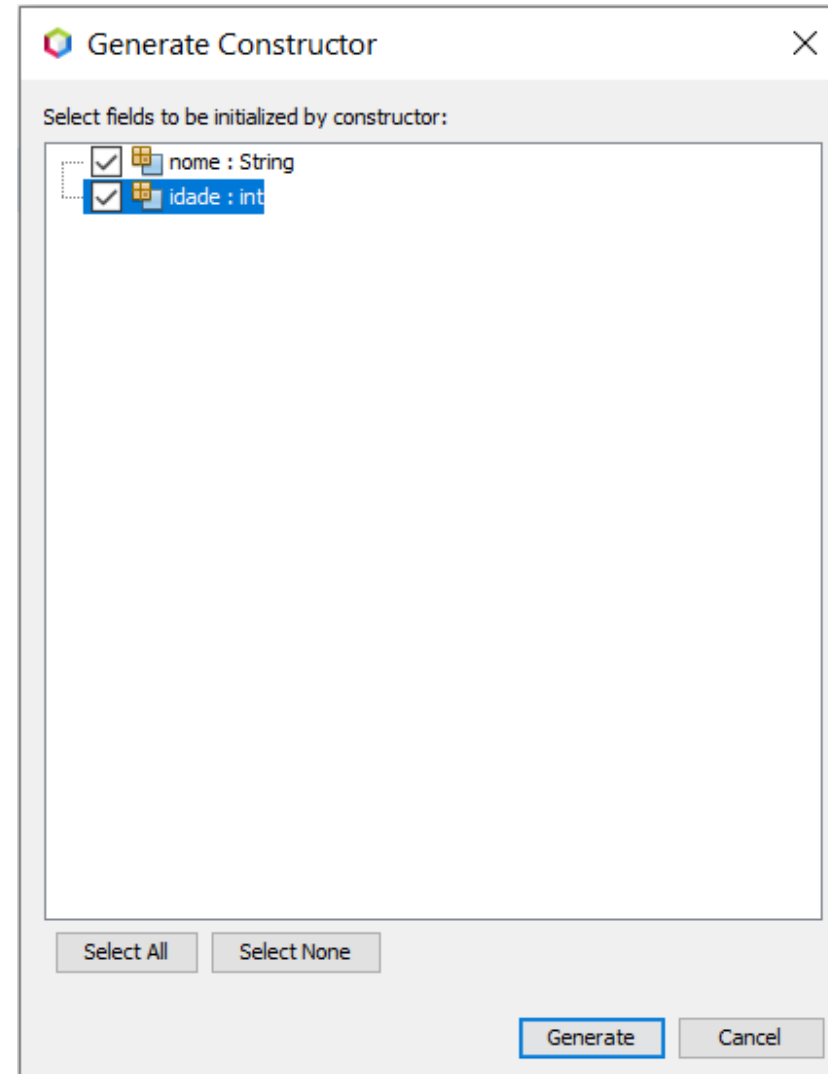
Selecione a opção Constructor:



```
1 package introducao_jsp.exercicio_torneio;
2
3 public class Torneio {
4     String nome;
5     int idade;
6
7     // Generate menu open
8     // Constructor... selected
9 }
```

# Correção do Exercício Torneio

Selecione os campos necessários para criar o objeto, nesse caso o nome e a idade.

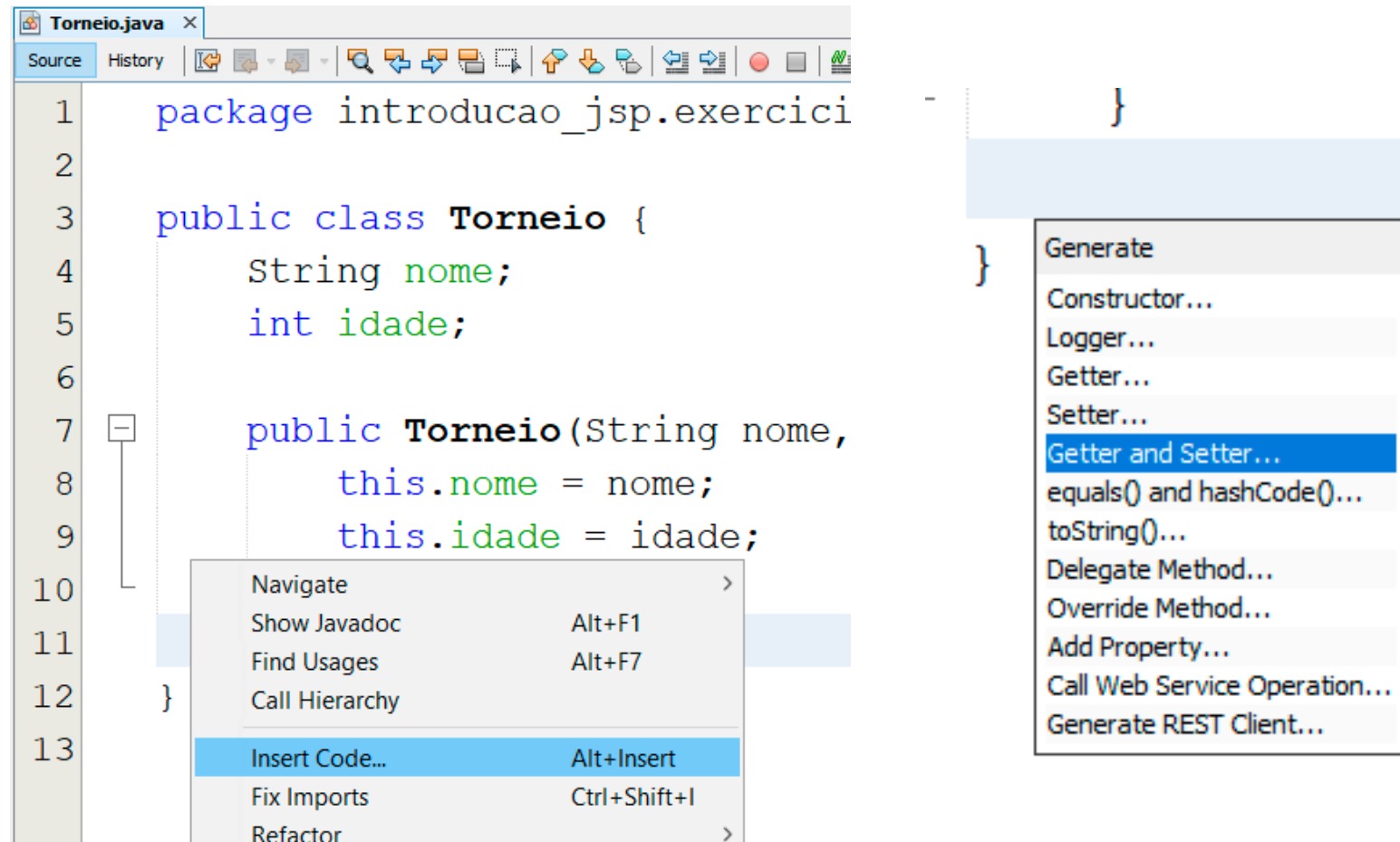


# Correção do Exercício Torneio

```
Torneio.java x
Source History
1 package introducao_jsp.exercicio_torneio;
2
3 public class Torneio {
4     String nome;
5     int idade;
6
7     public Torneio(String nome, int idade) {
8         this.nome = nome;
9         this.idade = idade;
10    }
11
12 }
```

# Correção do Exercício Torneio

Agora vamos criar os métodos de acesso aos atributos, para isso, vamos selecionar o botão direito do mouse e selecionar a opção Insert Code ...:



```

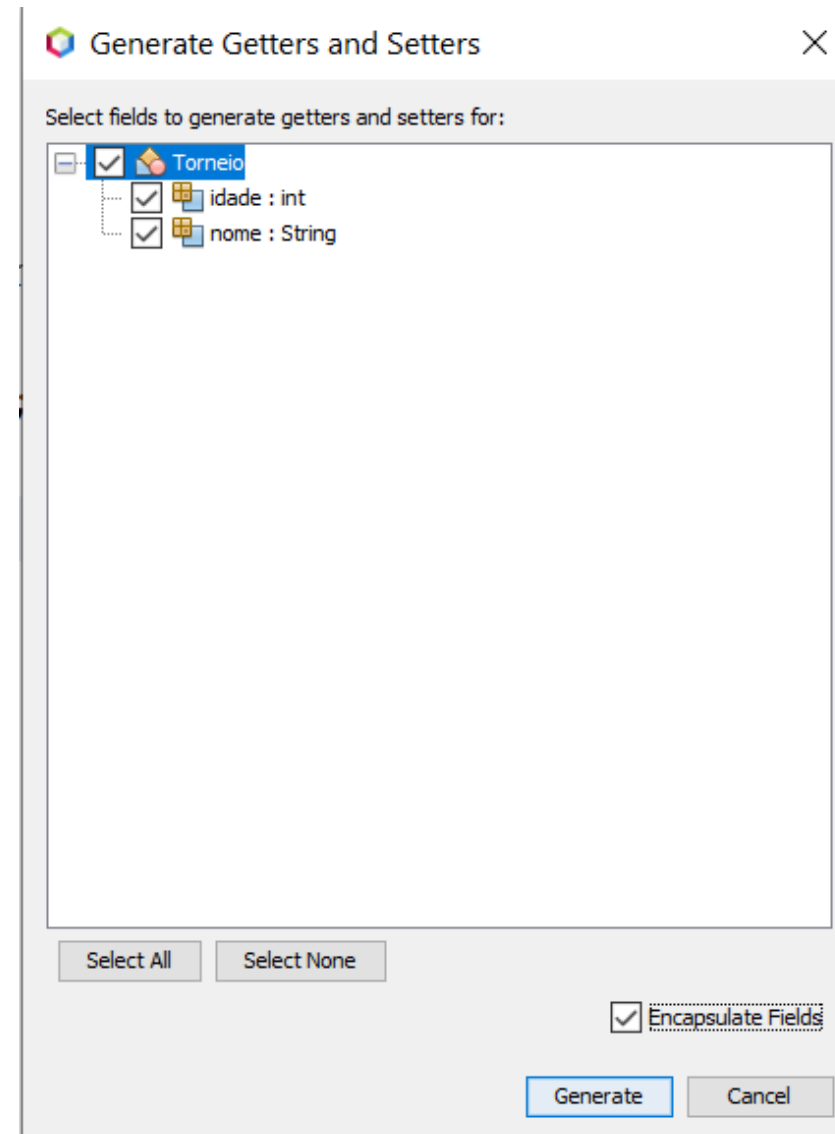
1 package introducao_jsp.exercici
2
3 public class Torneio {
4     String nome;
5     int idade;
6
7     public Torneio(String nome,
8         this.nome = nome;
9         this.idade = idade;
10
11 }
12
13

```

# Correção do Exercício Torneio

Selecione os atributos que terão os métodos de acesso criados e também selecione a opção de encapsulamento.

**Encapsulamento:** Um mecanismo da linguagem de programação para restringir o acesso a alguns componentes dos objetos, escondendo os dados de uma classe e tornando-os disponíveis somente através de métodos.



# Correção do Exercício Torneio

```
Torneio.java x
Source History
7 public Torneio(String nome, int idade) {
8     this.nome = nome;
9     this.idade = idade;
10 }
11
12 public int getIdade() {
13     return idade;
14 }
15
16 public void setIdade(int idade) {
17     this.idade = idade;
18 }
19
20 public String getNome() {
21     return nome;
22 }
23
24 public void setNome(String nome) {
25     this.nome = nome;
26 }
```

}

Métodos Construtor

}

Métodos de acesso



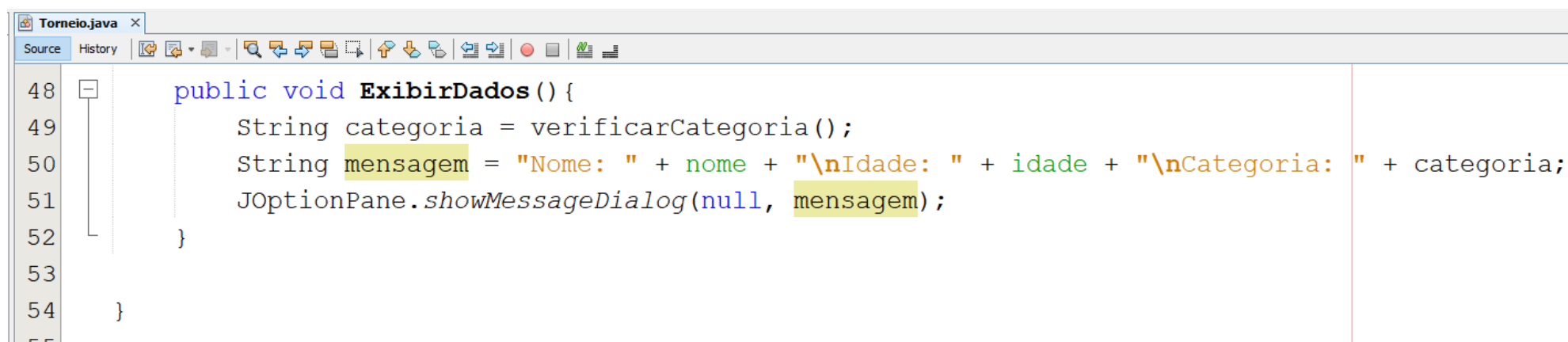
# Correção do Exercício Torneio

Agora vamos criar um método que faça a verificação da categoria com base na idade:

```
Torneio.java x
Source History
27 public String verificarCategoria() {
28     if (idade >= 5 && idade <= 7)
29         return "infantil";
30
31     if (idade >= 8 && idade <= 10)
32         return "juvenil";
33
34     if (idade >= 11 && idade <= 15)
35         return "adolescente";
36
37     if (idade >= 16 && idade <= 30)
38         return "adulto";
39
40     if (idade >= 30)
41         return "senior";
42
43     return "idade invalida";
44 }
```

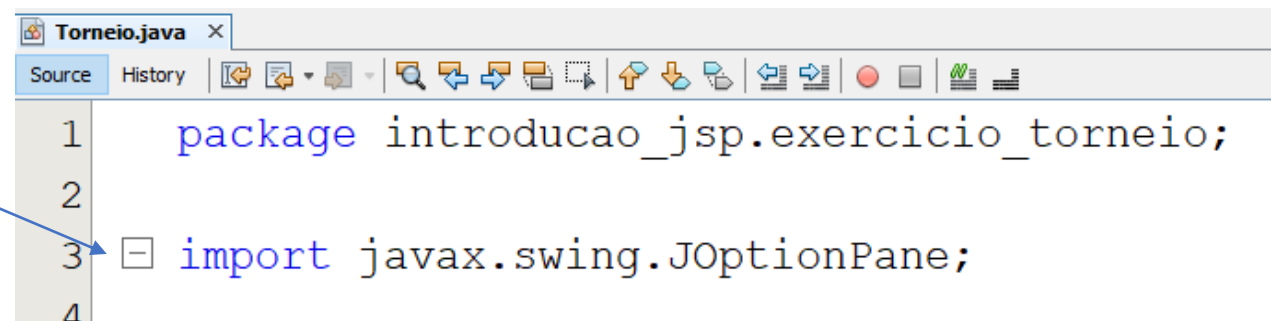
# Correção do Exercício Torneio

Agora vamos criar um método para exibir os dados:



```
48 public void ExibirDados() {  
49     String categoria = verificarCategoria();  
50     String mensagem = "Nome: " + nome + "\nIdade: " + idade + "\nCategoria: " + categoria;  
51     JOptionPane.showMessageDialog(null, mensagem);  
52 }  
53  
54 }
```

Como estamos trabalhando com o objeto JOptionPane é necessário fazer o import da classe.



```
1 package introducao_jsp.exercicio_torneio;  
2  
3 import javax.swing.JOptionPane;  
4
```

# Correção do Exercício Torneio

Agora vamos criar uma nova classe principal com o método main().

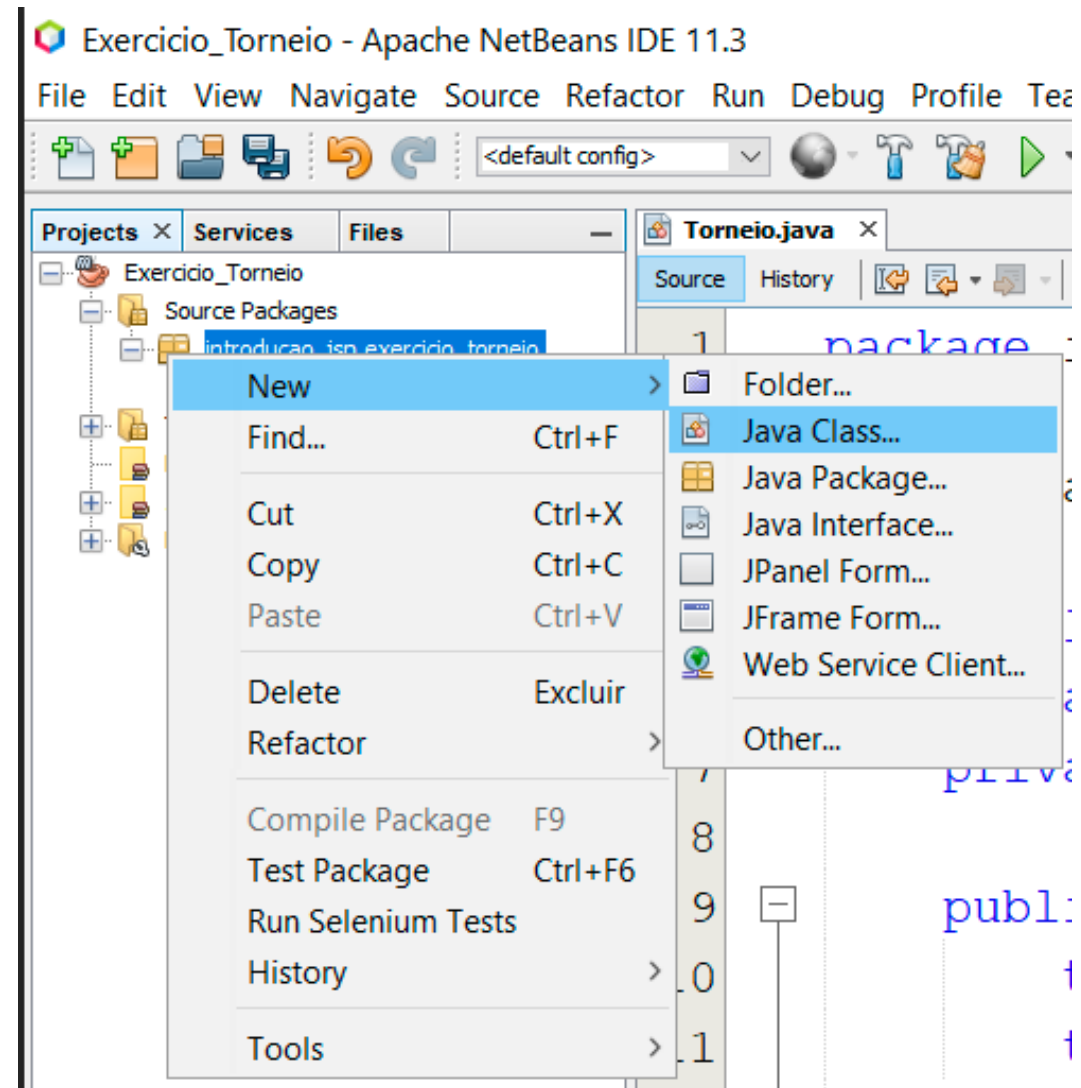
Toda aplicação Java SE precisa de uma classe principal, que normalmente é gerada quando se cria um novo projeto.

A classe principal é uma classe de "Start" da aplicação, ou seja, a porta de entrada da aplicação.


Você pode ter mais de 20 classes no seu projeto, mas apenas uma será a classe principal, a partir da qual você acessará as demais classes, o que define uma classe como principal é a introdução ***public static void main(String[] args).***

# Correção do Exercício Torneio

Selecione a opção *Source Package* e clique com o botão direito do mouse e selecione a opção *New – Java Class*



# Correção do Exercício Torneio

 New Java Class ✕

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

< Back

Next >

Finish

Cancel

Help

# Correção do Exercício Torneio

```
Torneio.java x Principal.java x
Source History
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package introducao_jsp.exercicio_torneio;
7
8  /**
9   *
10   * @author claud
11   */
12  public class Principal {
13
14  }
15
```

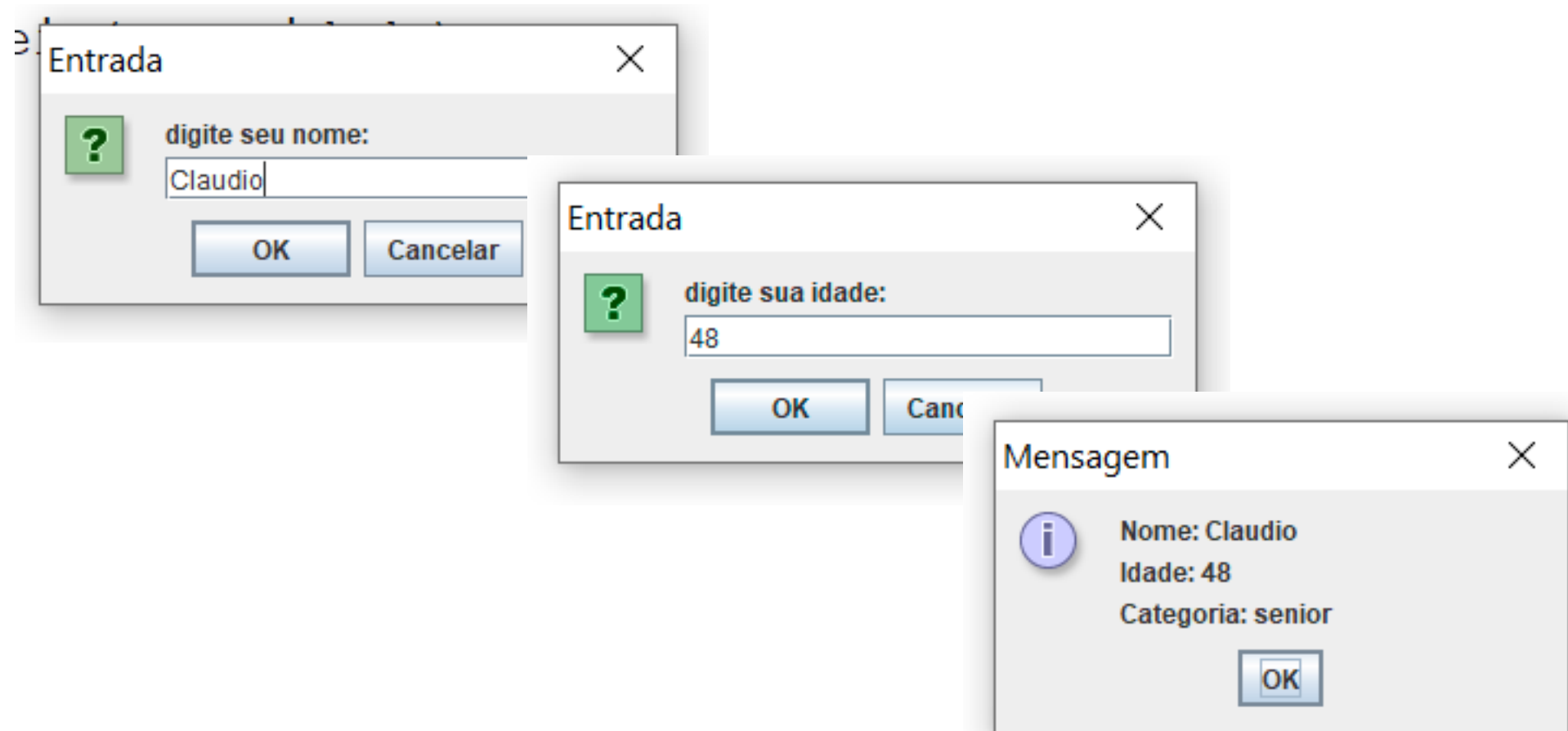
# Correção do Exercício Torneio

Agora vamos limpar o código e criar o método main, solicitar ao usuário que informe seu nome e idade e instanciar o objeto Torneio com os dados informados pelo usuário.

```
Torneio.java x Principal.java x
Source History
1 package introducao_jsp.exercicio_torneio;
2
3 import javax.swing.JOptionPane;
4
5 public class Principal {
6     public static void main (String [] args){
7         String nome = JOptionPane.showInputDialog("digite seu nome: ");
8         int idade = Integer.parseInt(JOptionPane.showInputDialog("digite sua idade: "));
9
10        Torneio t01 = new Torneio(nome,idade);
11        t01.ExibirDados();
12    }
13 }
```

# Correção do Exercício Torneio

Testando



The image displays three overlapping Windows-style dialog boxes, illustrating the execution of a tournament exercise. The first dialog, titled "Entrada", prompts the user to "digite seu nome:" (enter your name) and shows the input "Claudio". The second dialog, also titled "Entrada", prompts the user to "digite sua idade:" (enter your age) and shows the input "48". The third dialog, titled "Mensagem" (Message), displays the collected information: "Nome: Claudio", "Idade: 48", and "Categoria: senior".

**Entrada**

digite seu nome:

Claudio

OK Cancelar

**Entrada**

digite sua idade:

48

OK Cancelar

**Mensagem**

Nome: Claudio  
Idade: 48  
Categoria: senior

OK



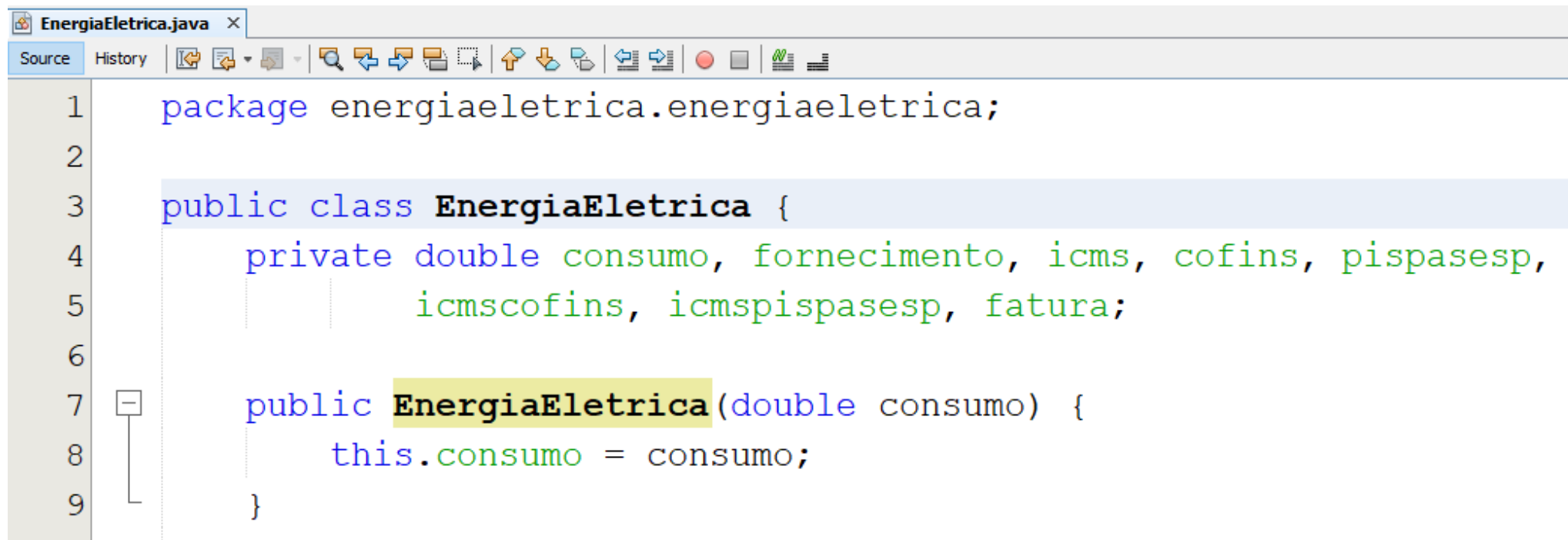
# Correção do Exercício Consumo da Energia Elétrica

Vamos Criar um Novo Projeto como “**EnergiaEletrica**”, depois vamos criar a nossa classe EnergiaEletrica e estabelecer os atributos a serem utilizados nessa aplicação:

```
EnergiaEletrica.java x
Source History
1 package energiaelettrica.energieletrica;
2
3 public class EnergiaEletrica {
4     private double consumo, fornecimento, icms, cofins, pispasesp, icmscofins, icmspispasesp, fatura;
5
6 }
```

# Correção do Exercício Consumo da Energia Elétrica

Vamos criar o método construtor dessa classe, lembrando que é necessário passar o consumo, os demais atributos serão calculados:



```
1 package energiaeletrica.energiaeletrica;
2
3 public class EnergiaEletrica {
4     private double consumo, fornecimento, icms, cofins, pispasesp,
5         icmscofins, icmspispasesp, fatura;
6
7     public EnergiaEletrica(double consumo) {
8         this.consumo = consumo;
9     }
```

# Correção do Exercício Consumo da Energia Elétrica

Agora vamos criar os métodos de acesso aos atributos dos atributos:

```
EnergiaEletrica.java
Source History
13 public double getConsumo() {
14     return consumo;
15 }
16
17 public void setConsumo(double consumo) {
18     this.consumo = consumo;
19 }
20
21 public double getFornecimento() {
22     return fornecimento;
23 }
24
25 public void setFornecimento(double fornecimento) {
26     this.fornecimento = fornecimento;
27 }
28
29 public double getIcms() {
30     return icms;
31 }
32
33 public void setIcms(double icms) {
34     this.icms = icms;
35 }
```

# Correção do Exercício Consumo da Energia Elétrica

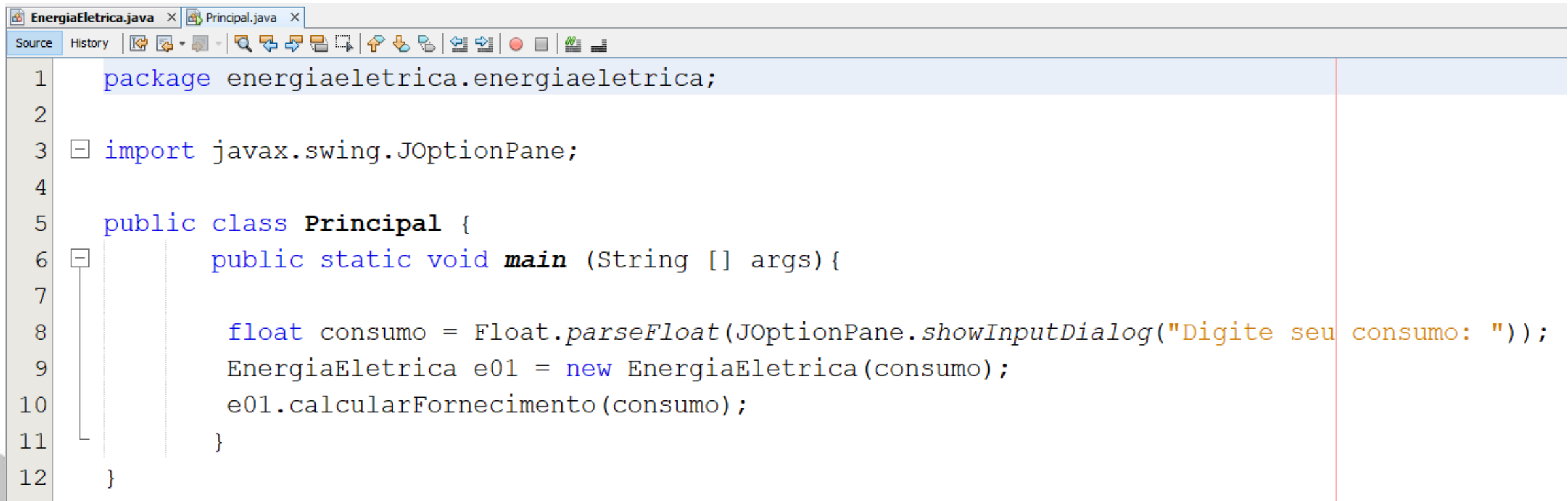
Vamos criar um método para calcular o fornecimento, com base na descrição do exercício:

```
EnergiaEletrica.java
Source History
77 public void calcularFornecimento(double consumo) {
78     this.setFornecimento(consumo * 0.28172);
79
80     if (consumo <= 200) {
81         this.setIcms(fornecimento * 0.136363);
82         this.setCofins(fornecimento * 0.0614722);
83         this.setPispasesp(fornecimento * 0.013346);
84         this.setIcmscofins(fornecimento * 0.0614722 * 0.0136363);
85         this.setIcmspispasesp(fornecimento * 0.013346 * 0.0136363);
86     }
87     else {
88         this.setIcms(fornecimento * 0.333333);
89         this.setCofins(fornecimento * 0.0730751);
90         this.setPispasesp(fornecimento * 0.0158651);
91         this.setIcmscofins(fornecimento * 0.0730751 * 0.333333);
92         this.setIcmspispasesp(fornecimento * 0.0158651 * 0.333333);
93     }
94
95     this.setFatura(fornecimento + icms + cofins + pispasesp + icmscofins + icmspispasesp);
96
97     String mensagem = "fornecimento: " + getFornecimento() + "\nIcms: " + getIcms() + "\nCofins: " + g
98     JOptionPane.showMessageDialog(null, mensagem);
99 }
100 }
```

```
77 public void calcularFornecimento(double consumo) {
78     this.setFornecimento(consumo * 0.28172);
79     if (consumo <= 200) {
80         this.setIcms(fornecimento * 0.136363);
81         this.setCofins(fornecimento * 0.0614722);
82         this.setPispasesp(fornecimento * 0.013346);
83         this.setIcmscofins(fornecimento * 0.0614722 * 0.0136363);
84         this.setIcmspispasesp(fornecimento * 0.013346 * 0.0136363);
85     }
86     else {
87         this.setIcms(fornecimento * 0.333333);
88         this.setCofins(fornecimento * 0.0730751);
89         this.setPispasesp(fornecimento * 0.0158651);
90         this.setIcmscofins(fornecimento * 0.0730751 * 0.333333);
91         this.setIcmspispasesp(fornecimento * 0.0158651 * 0.333333);
92     }
93     this.setFatura(fornecimento + icms + cofins + pispasesp + icmscofins + icmspispasesp);
94     String mensagem = "fornecimento: " + getFornecimento() + "\nIcms: " + getIcms() +
95         "\nCofins: " + getCofins() + "\nPis/Pasesp: " + getPispasesp() +
96         "\nIcms para Cofins: " + getIcmscofins() + "\nIcms para Pis/Pasesp: " +
97         getIcmspispasesp() + "\nFatura: " + getFatura();
98     JOptionPane.showMessageDialog(null, mensagem);
99 }
```

# Correção do Exercício Consumo da Energia Elétrica

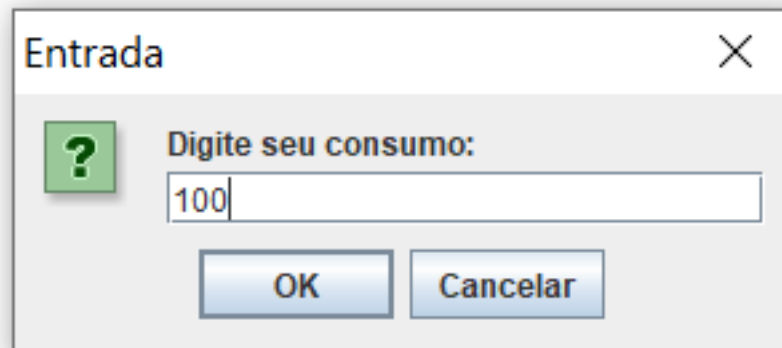
Vamos criar agora nossa classe principal:



```
1 package energiaeletrica.energieletrica;
2
3 import javax.swing.JOptionPane;
4
5 public class Principal {
6     public static void main (String [] args){
7
8         float consumo = Float.parseFloat(JOptionPane.showInputDialog("Digite seu consumo: "));
9         EnergiaEletrica e01 = new EnergiaEletrica(consumo);
10        e01.calcularFornecimento(consumo);
11    }
12 }
```

# Correção do Exercício Consumo da Energia Elétrica

Testando:



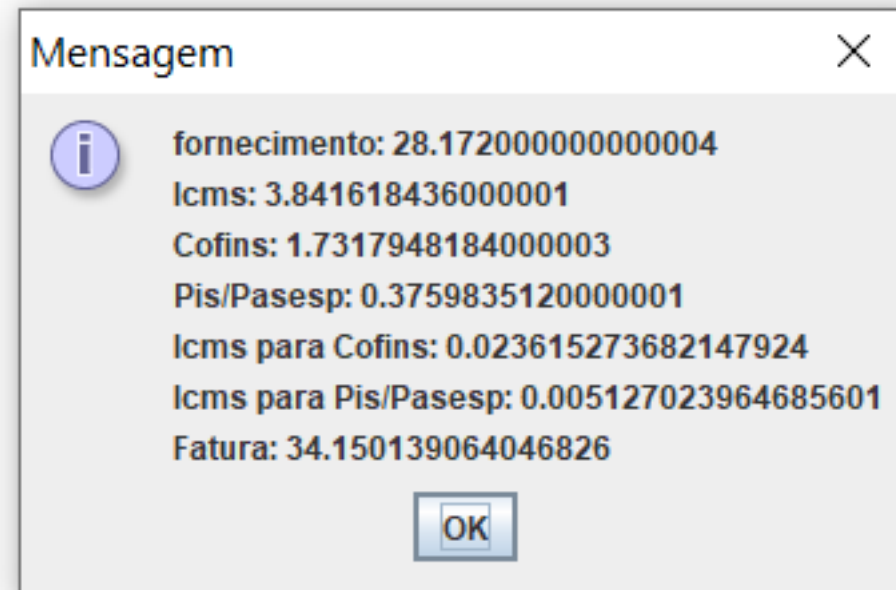
Entrada

?

Digite seu consumo:

100

OK Cancelar



Mensagem

i

fornecimento: 28.172000000000004  
Icms: 3.841618436000001  
Cofins: 1.7317948184000003  
Pis/Pasesp: 0.3759835120000001  
Icms para Cofins: 0.023615273682147924  
Icms para Pis/Pasesp: 0.005127023964685601  
Fatura: 34.150139064046826

OK



# Herança

Como o próprio nome sugere, na orientação a objetos o termo herança **se refere a algo herdado**.

Em Java, a herança ocorre quando uma **classe** passa a herdar **características** (atributos e métodos) **definidas em uma outra classe**, especificada como sendo sua ancestral ou superclasse.







# Herança

A técnica da herança **possibilita** o **compartilhamento** ou **reaproveitamento** de recursos definidos anteriormente em uma outra classe.

A classe **fornecedora** dos recursos recebe o nome de **superclasse** e a **receptora** dos recursos de **subclasse**.



Uma classe derivada **herda a estrutura de atributos** e métodos de sua classe “base”, mas pode seletivamente:

- adicionar novos métodos
- estender a estrutura de dados
- redefinir a implementação de métodos já existentes

► Exemplo:

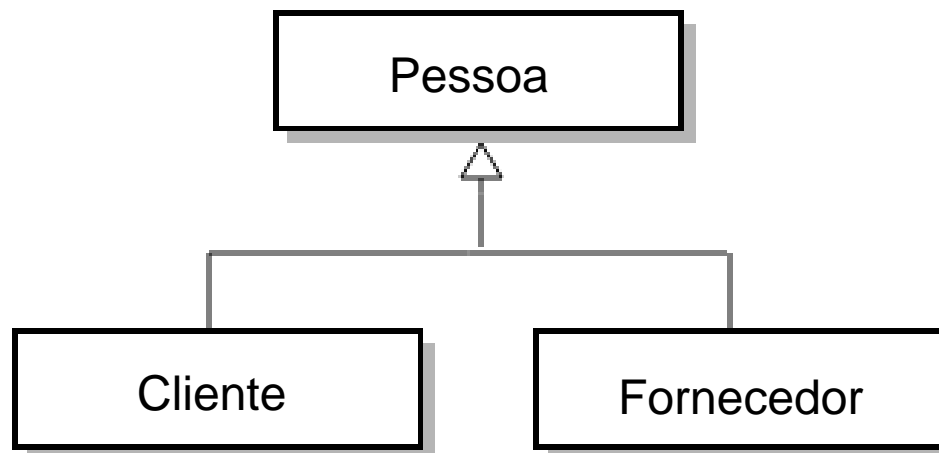
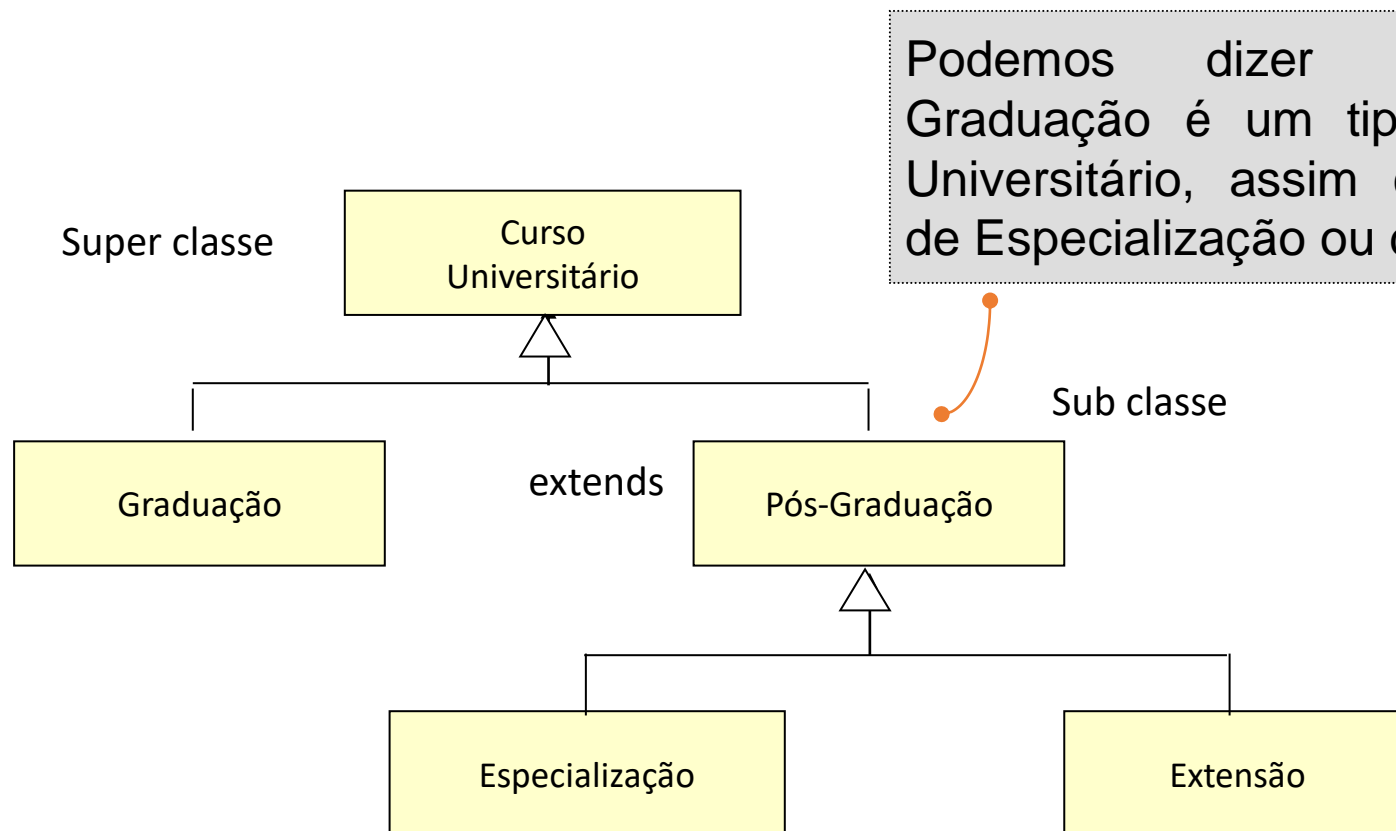


Diagrama UML simplificado (não mostra os métodos e atributos)





# Herança em Java

Para dizer que uma classe implementa herança, utilizamos a palavra-chave `extends` logo após a declaração da mesma:

```
public class Filho extends Pai{ }
```



# Herança em Java

Se uma classe não contém a declaração `extends`, automaticamente “herda” as características da classe **Object**

Alguns métodos da classe **Object** são:

- `toString( ) : String`
- `equals( Object ) : boolean`

# Herança modificador de acesso

- ▶ **protected**: podem ser acessados pelos membros da subclasse
- ▶ **private**: só podem ser acessados pela própria classe

```
public class Pai {  
    public int a;  
    protected int b;  
    private int c;  
  
    public Pai() {  
        a=10;  
        b=24;  
        c=66;  
    }  
}
```

Pai.java

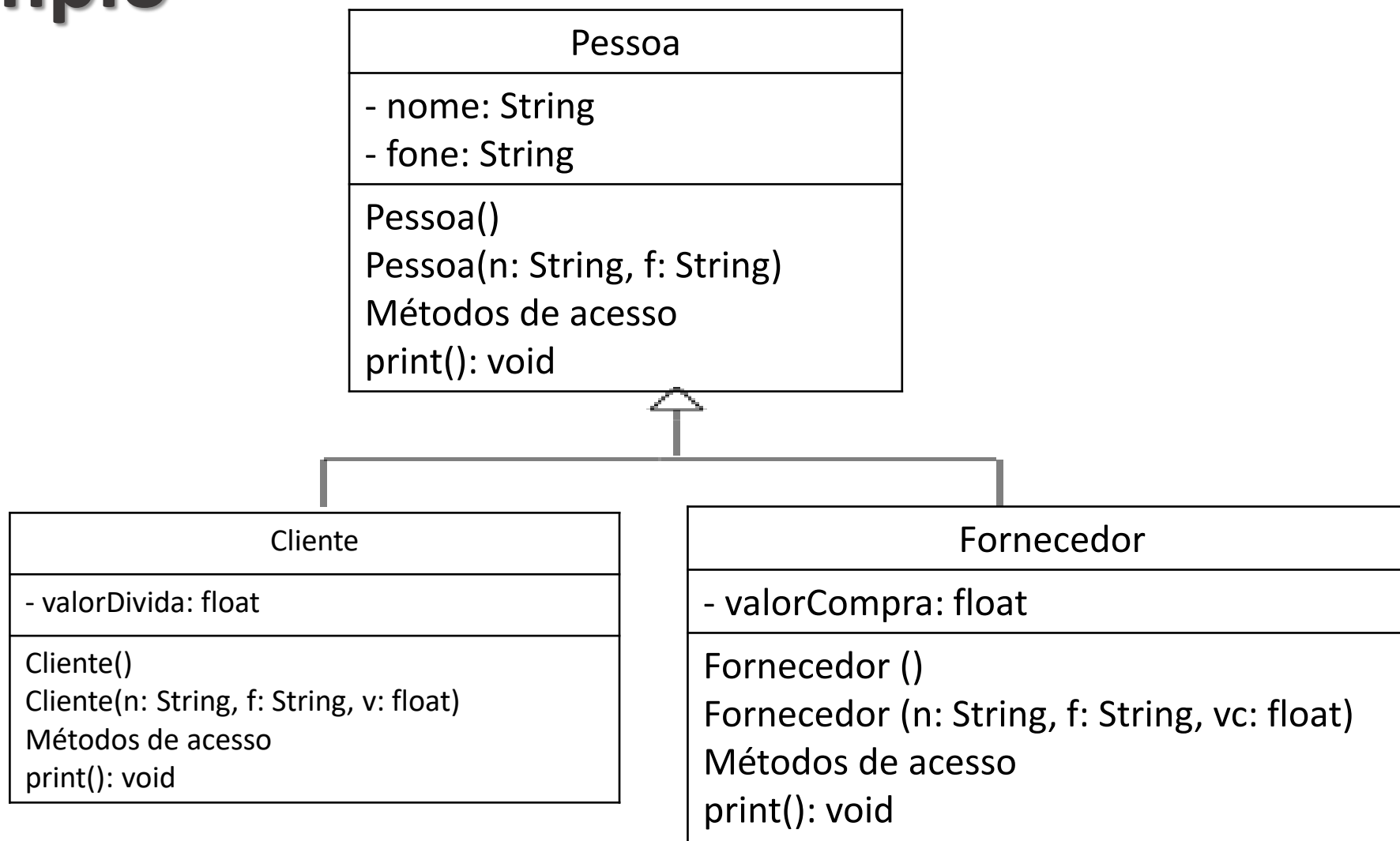
```
public class Filho extends Pai{  
  
    public void imprimeFilho() {  
        System.out.println("Valor de a: " + a);  
        System.out.println("Valor de b: " + b);  
        System.out.println("Valor de c: " + c);  
    }  
}
```

Filho.java

```
public class TestaFilho {  
  
    public static void main(String[] args) {  
        Filho f=new Filho();  
        f.imprimeFilho();  
    }  
}
```

TestaFilho.java

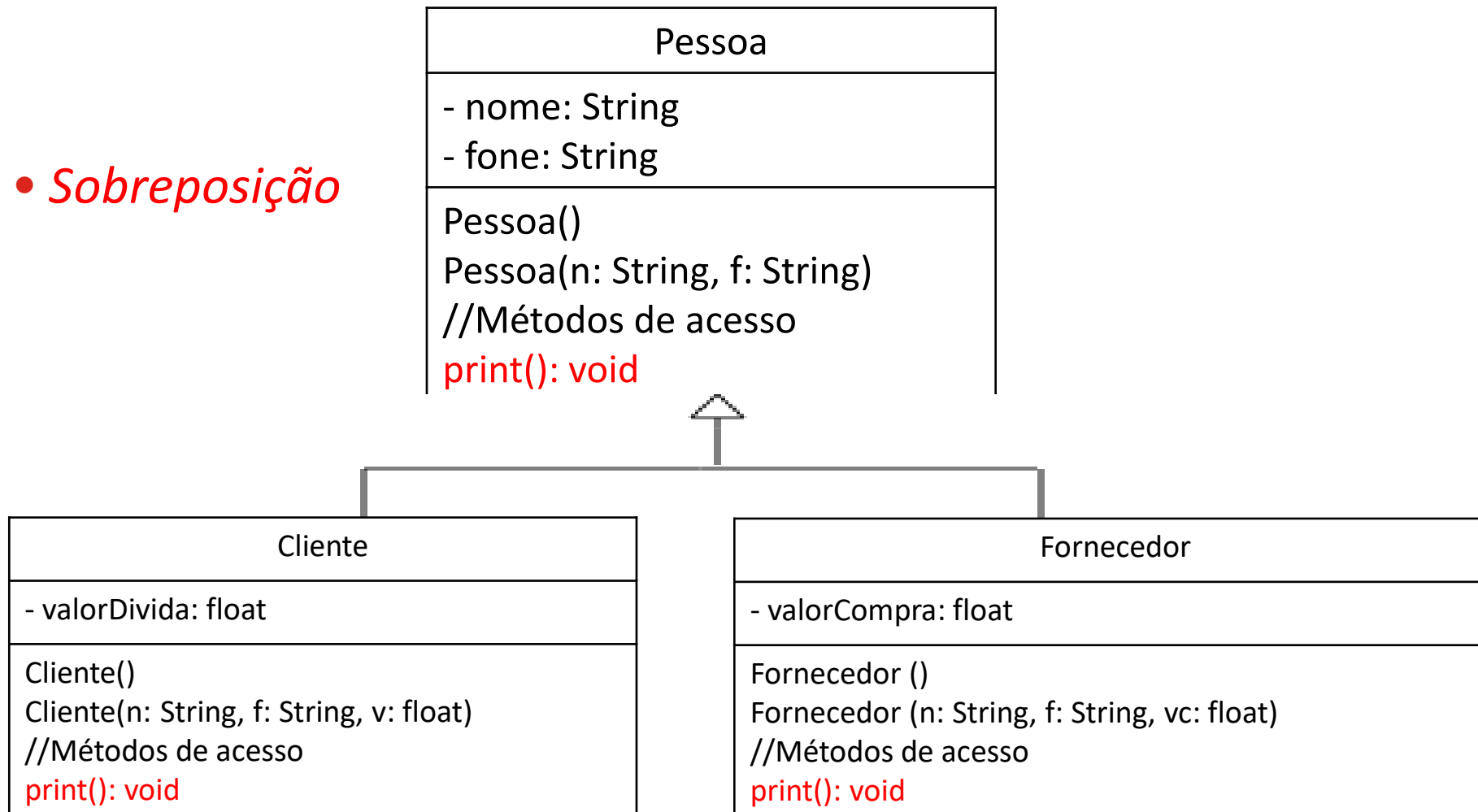
# Exemplo





# Sobreposição (Overriding) e Extensão

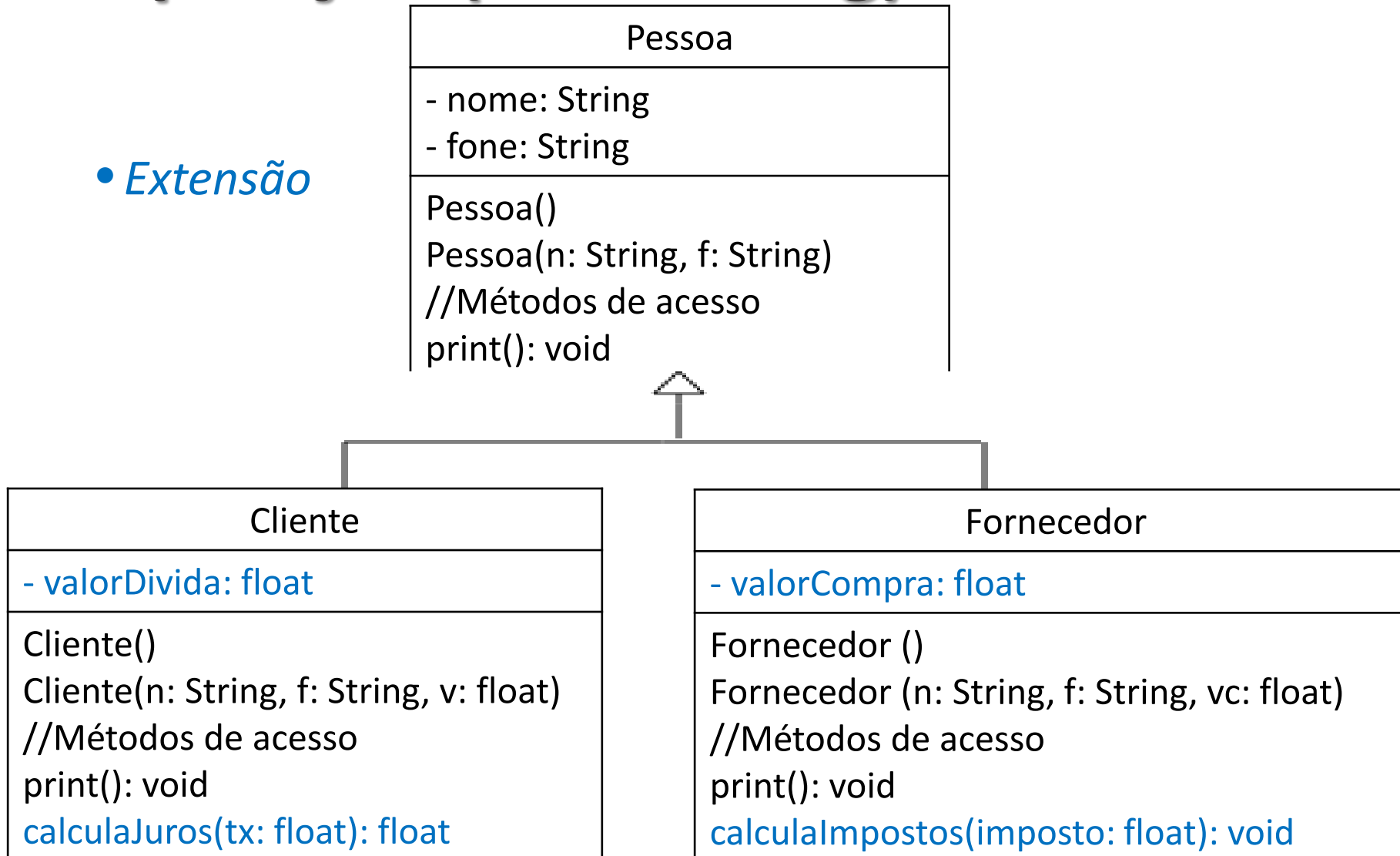
- *Sobreposição*



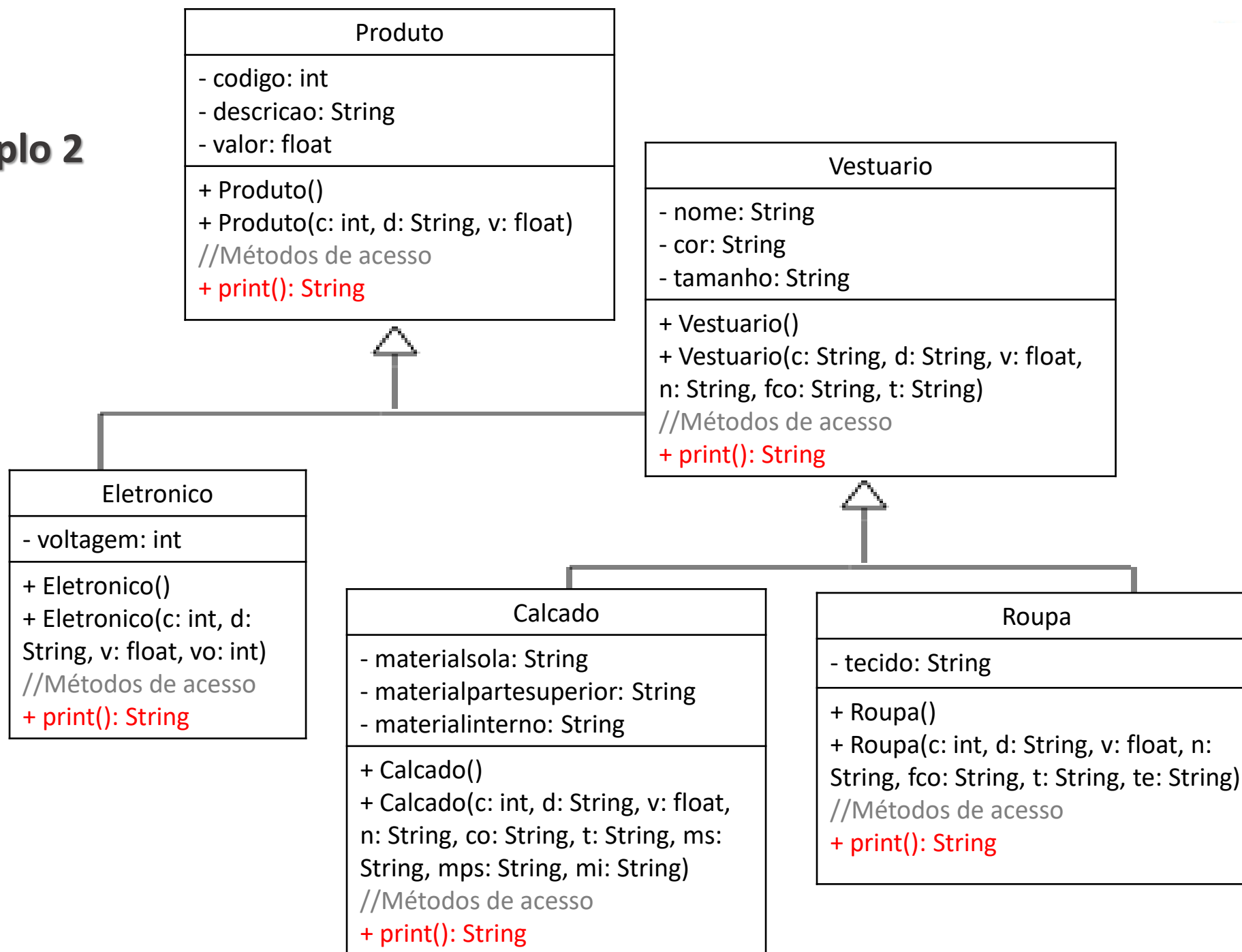


# Sobreposição (Overriding) e Extensão

- *Extensão*



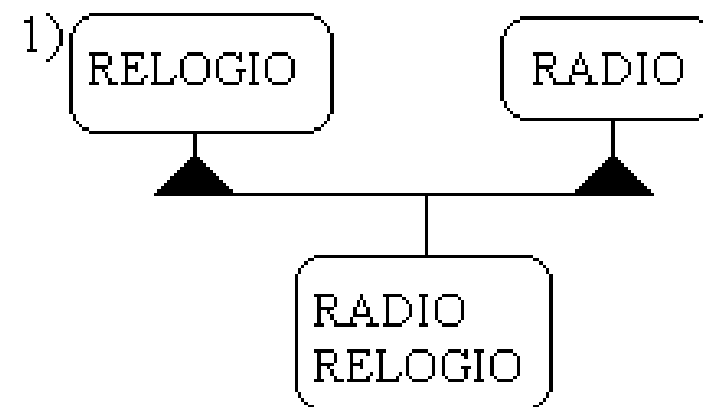
## Exemplo 2



# Herança múltipla

Algumas linguagens OO permitem fazer herança múltipla como o C++, o que significa que uma subclasse pode herdar características de duas ou mais classes. Isso permite que uma classe agrupe atributos e métodos de várias classes.

Java não tem herança múltipla!

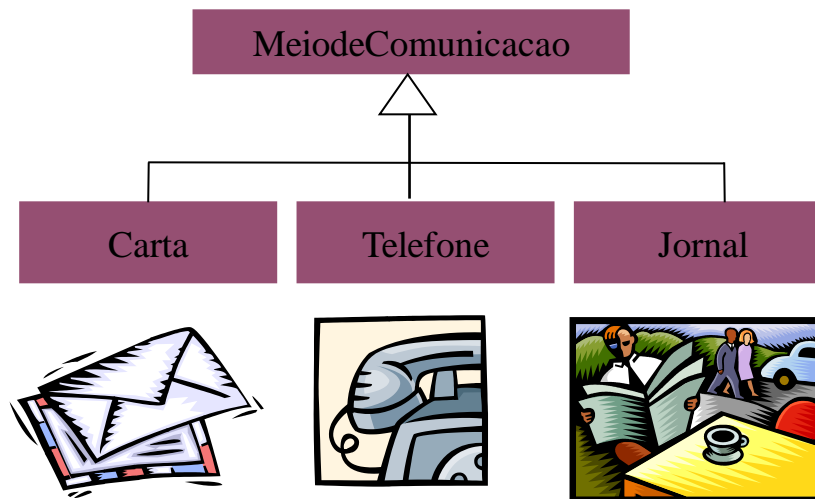


# Abstração de Dados – Classes Abstratas

*Exemplo*

Nos ajuda a lidar com a complexidade.

*Generalização*



*Especialização*

A classe *MeiodeComunicacao* neste caso é abstrata e pode representar um domínio.

# Classes Abstratas

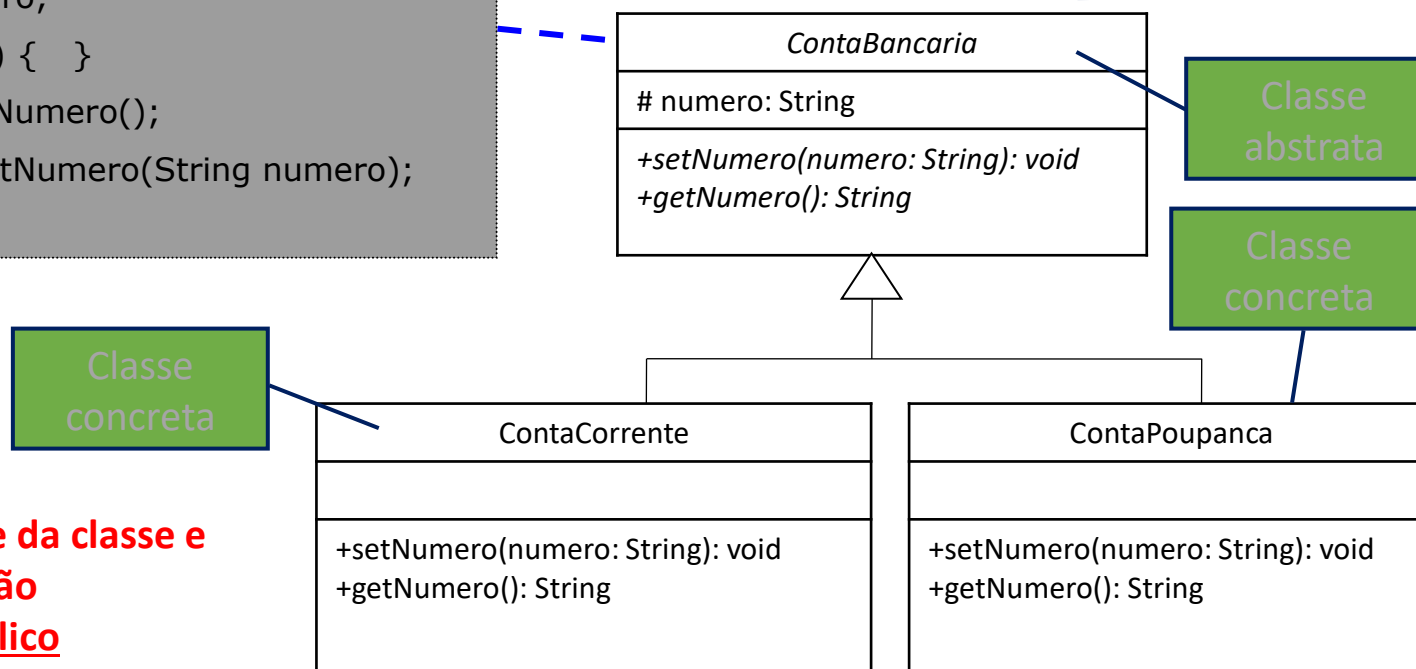
Uma classe abstrata é uma classe que:

- Provê organização
- **Não possui “instâncias”**
- Pode ter métodos abstratos ou concretos

```
public abstract class ContaBancaria {
    protected String numero;
    public ContaBancaria() { }
    public abstract int getNumero();
    public abstract void setNumero(String numero);
}
```

Se você definir um método abstrato sua classe **OBRIGATORIAMENTE DEVE SER ABSTRATA.**

Você pode ter métodos concretos em uma classe abstrata



**No diagrama o nome da classe e métodos abstratos são apresentados em itálico**

# Classes Abstratas

- ▶ São criadas quando não se pretende criar objetos a partir delas
- ▶ São normalmente usadas como superclasses
- ▶ Se caracterizam por serem muito genéricas
- ▶ A assinatura da classe abstrata é da seguinte forma:

```
public abstract class NomeDaClasse{
```

```
...
```

```
}
```

Palavra-chave que identifica a classe como abstrata



# Métodos Abstratos

Os métodos abstratos definidos em uma classe abstrata devem obrigatoriamente implementados em uma classe concreta. Mas se uma classe abstrata herdar outra classe abstrata, a classe que herda não precisa implementar os métodos abstratos.

Indica que todas as classes filhas concretas devem reescrever esse método e ele deve ser um método concreto nas classes filhas;



# Métodos Abstratos

Os métodos que são abstratos têm um comportamento diferente nas classes filhas, por isso não possuem corpo;

A assinatura do método abstrato é da seguinte forma:

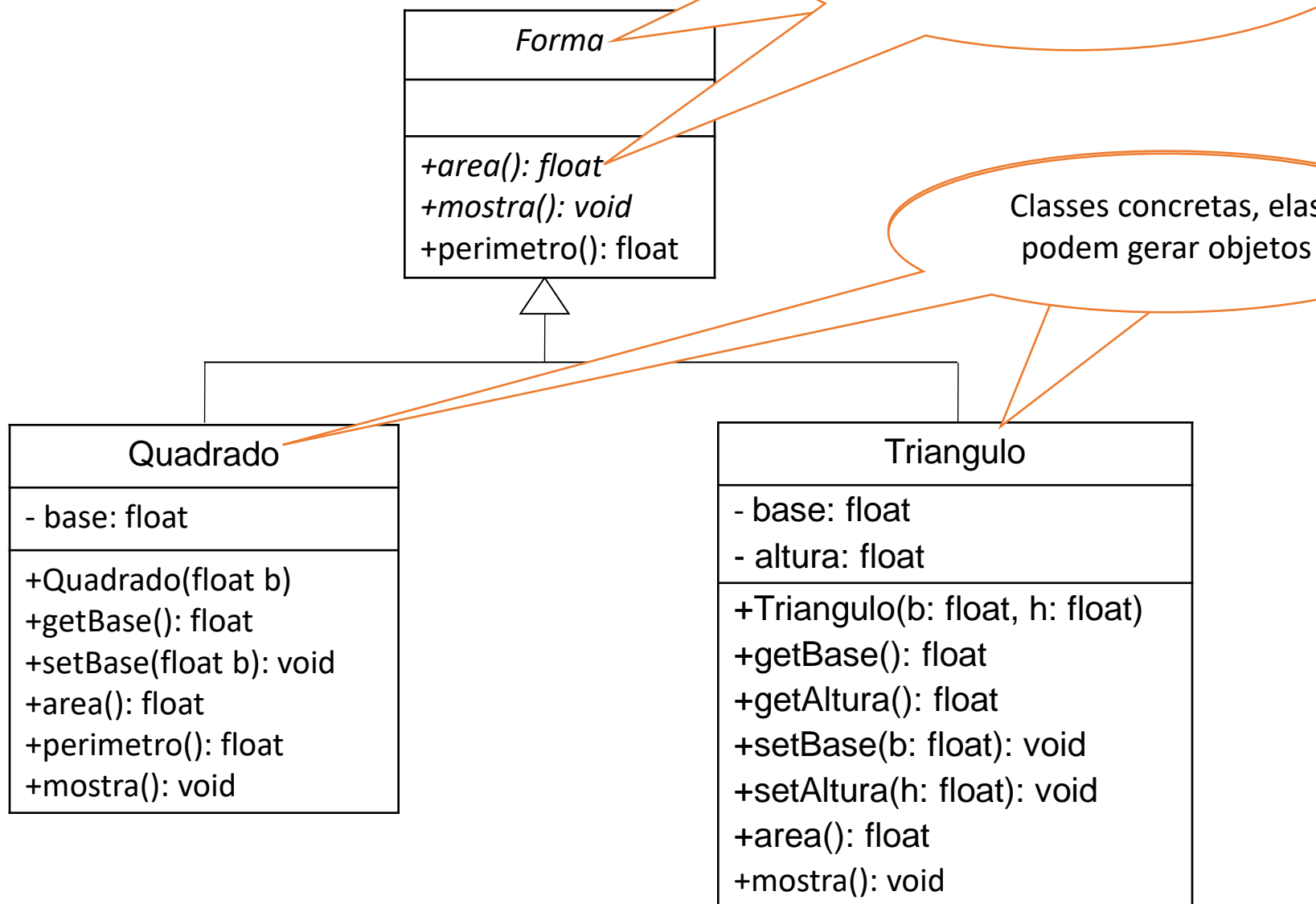
```
public abstract tipoRetorno NomeDoMetodo();
```

Palavra-chave que identifica  
o método como abstrato

Não é definida  
uma forma de  
implementação  
específica.



# Exemplos

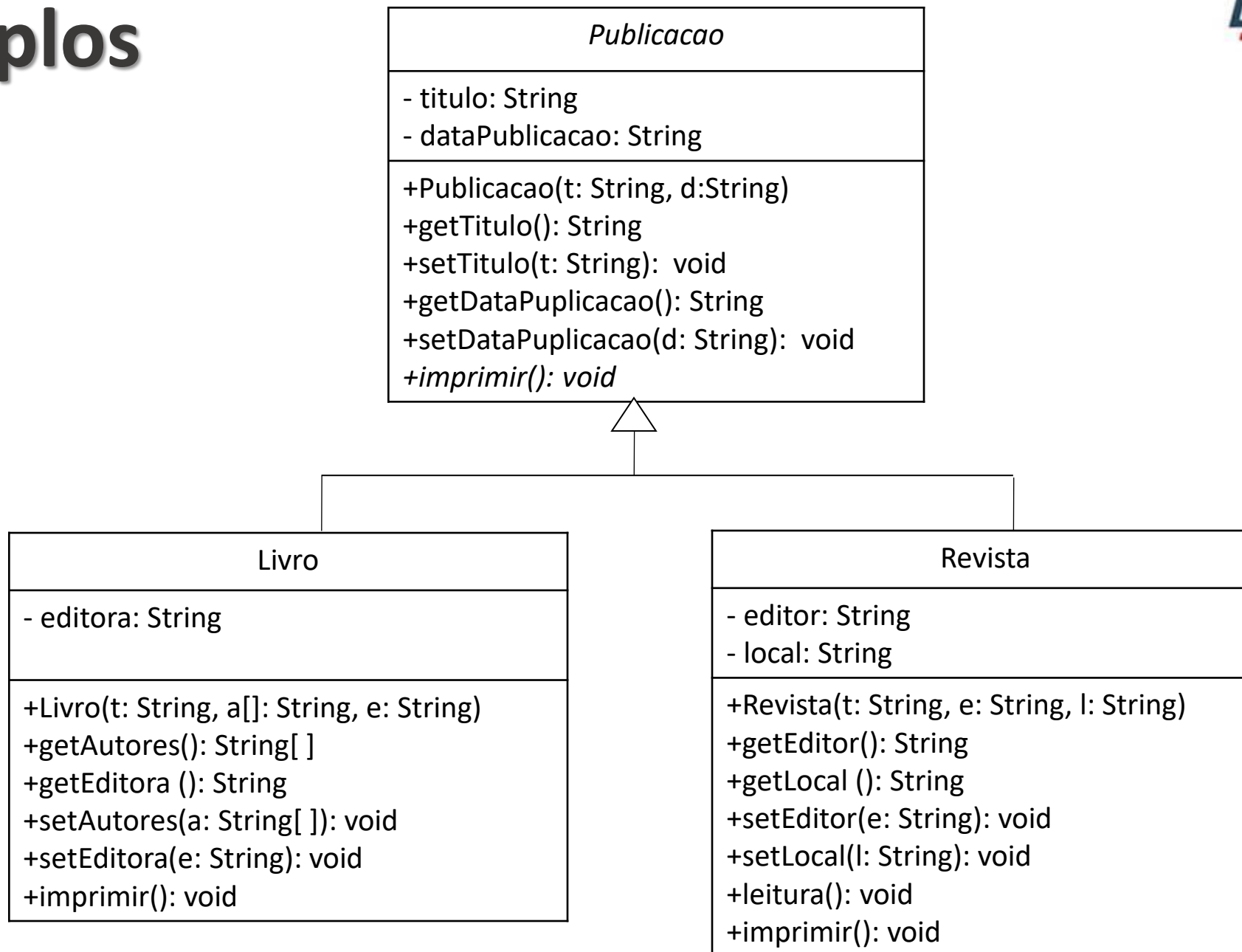


# Exemplos

```
public abstract class Forma {  
    //Atributos  
  
    //Metodos  
    public abstract float area();  
    public abstract void mostra();  
  
    public float perimetro(){ return 0f; }  
}
```

Métodos abstratos, não  
possuem implementação

# Exemplos





# Interface

São estruturas similares às classes abstratas, que definem a especificação de funcionalidades, sem a implementação das mesmas.

Uma interface Java pode ser definida como uma “**classe abstrata pura**”, pois não pode possuir atributos (exceto constantes) nem definições de métodos, nem construtor.

Apenas métodos públicos podem ser declarados nelas, mas não podem ser definidos.

- ✓ Todos os métodos declarados dentro de uma interface são, implicitamente, *public* e *abstract*

Da mesma forma, não é possível definir atributos - apenas constantes públicas.

- ✓ Todos os atributos declarados dentro de uma interface são, implicitamente, *public*, *static* e *final*
- 



# Interface

São proibidas algumas declarações dentro das interfaces, dentre elas:

✓ *private, protected, transient, volatile e synchronized.*





# Interface

A palavra-chave *interface* é utilizada em vez de *class* no cabeçalho da declaração:

- ▶ `public interface nomeInterface { ... }`

Todos os métodos são abstratos. A palavra-chave *abstract* não é necessária

# Classes Abstratas e Interface

A diferença entre uma classe abstrata e uma interface Java é que a interface obrigatoriamente não pode ter um “corpo” associado.

Enquanto uma classe abstrata é “estendida” (palavra chave *extends*) por classes derivadas, uma interface Java é “implementada” (palavra chave *implements*) por outras classes.



# Classes Abstratas e Interface

```
public class nomeClasse implements nomeInterface
```

Ou

```
public class nomeClasse implements nomeInterface1,  
                                   nomeInterface2
```

Ou

```
public class nomeClasse extends nomeHeranca implements  
                                   nomeInterface
```

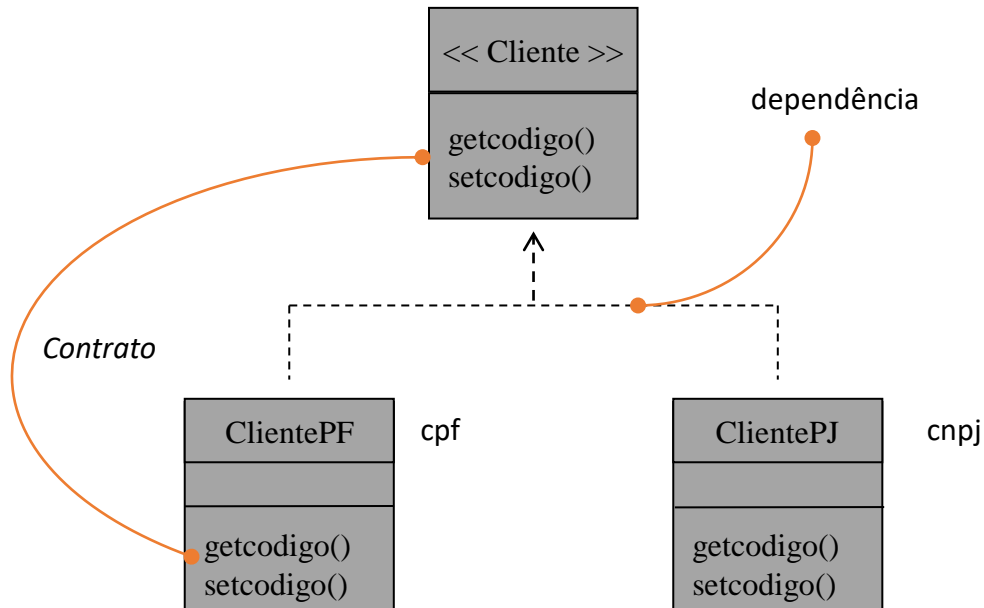
A interface **não faz parte da hierarquia** de classes em Java.

# Implementação de Interfaces

Uma interface estabelece uma espécie de contrato que é obedecido por uma classe.

Quando uma classe implementa uma interface, garante-se que todas as funcionalidades especificadas pela interface serão oferecidas pela classe.

# Implementação de Interfaces



Uma **classe pode implementar várias interfaces**, sendo um mecanismo elegante para se trabalhar com herança múltipla em Java.



# Interfaces - Vantagens

- Implementar similaridades entre classes não-relacionadas, sem forçar relacionamentos na hierarquia de classes.
- Definir métodos que uma ou mais classes devam implementar.

# Interfaces - Vantagens

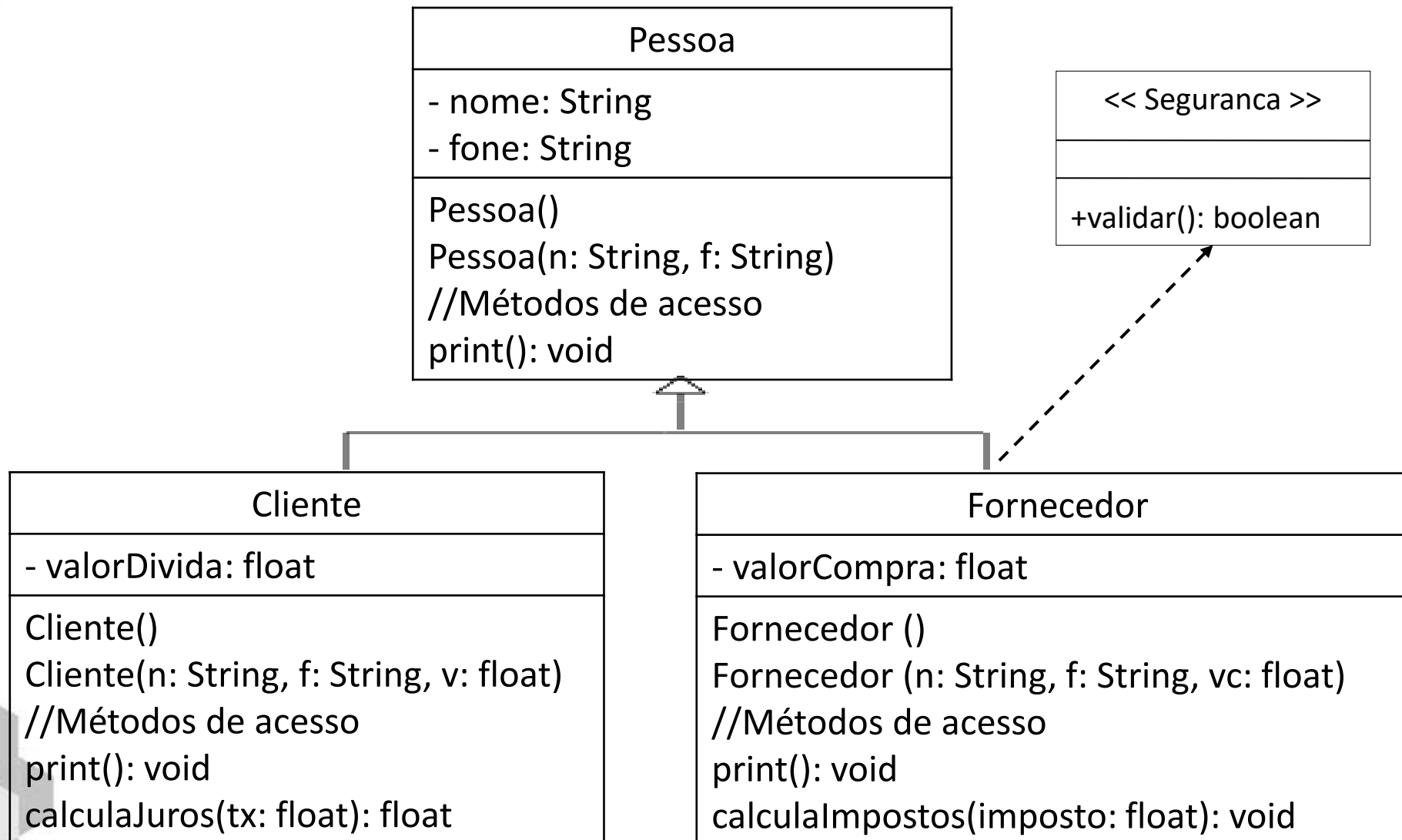
- Revelar apenas as interfaces de um objeto, sem revelar a estrutura de sua classe, ajuda assim esconder a complexidade da arquitetura de componentes;
- Oferece uma forma simplificada de implementar herança múltipla.

A decorative graphic in the top-left corner consisting of a cluster of hexagons in various colors including red, yellow, and grey.

# Interfaces - Vantagens

- Facilitar o entendimento do código – separação entre o que a classe “é” do que ela “faz”.

# Interfaces - Exemplo



# Interfaces - Exemplo

```
public interface Imprimivel {
    //constante
    final char nlin='\n';

    //métodos
    public String formatoString();
    public void formatoSystemOut();
}
```

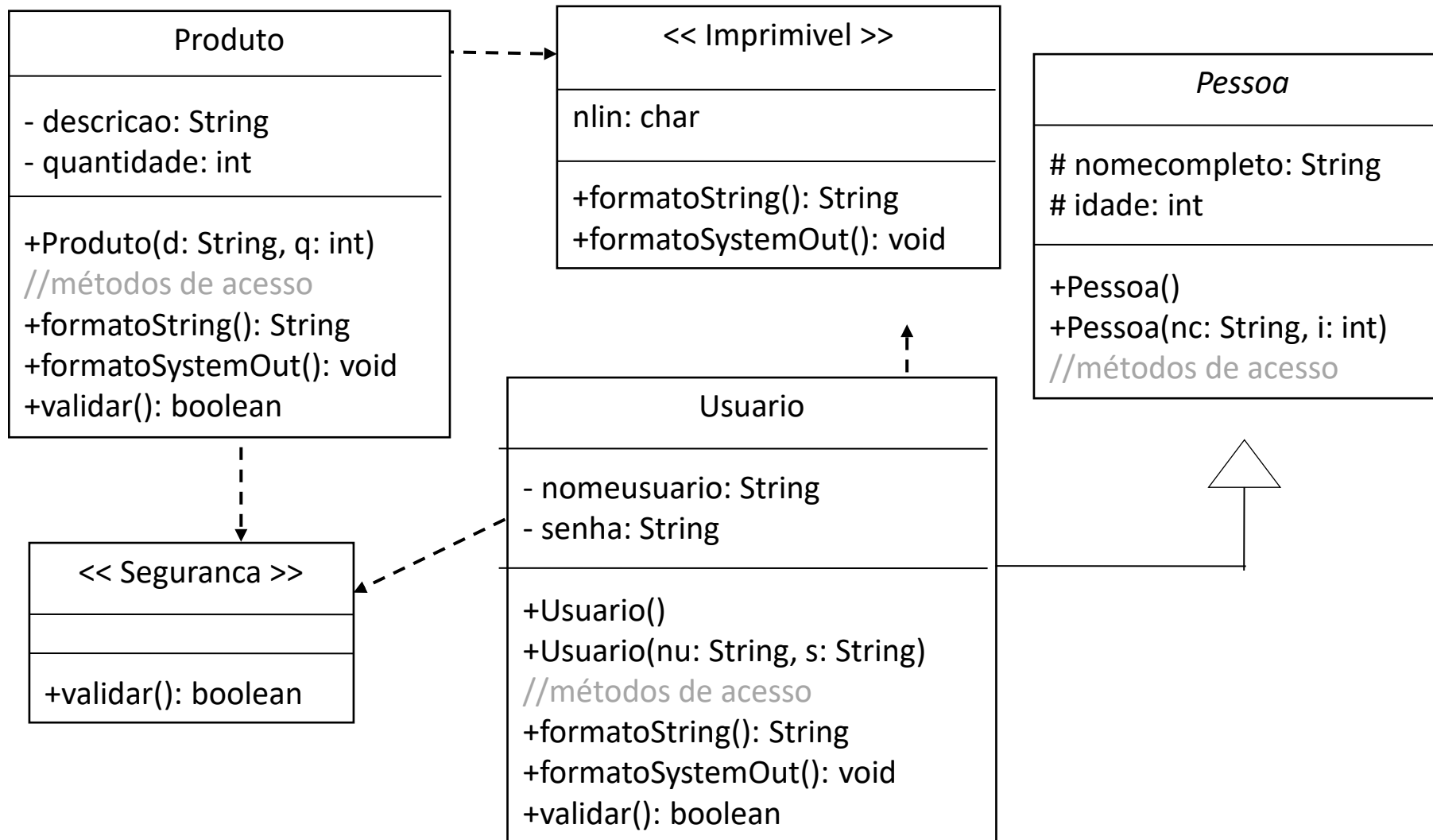
<< Imprimivel >>
nlin: char
+formatoString(): String +formatoSystemOut(): void

```
public interface Seguranca {
    public boolean validar();
}
```

<< Seguranca >>
+validar(): boolean



# Interfaces - Exemplo



# Interfaces - Exemplos

Todas as classes que implementem a interface "Imprimivel" terão a "obrigação" de implementar os métodos `formatoString()` e `formatoSystemOut()`;

Isso faz com que você tenha um padrão, e não importa como os métodos serão implementados, o usuário saberá que esses métodos existirão;



# Interfaces - Exemplos

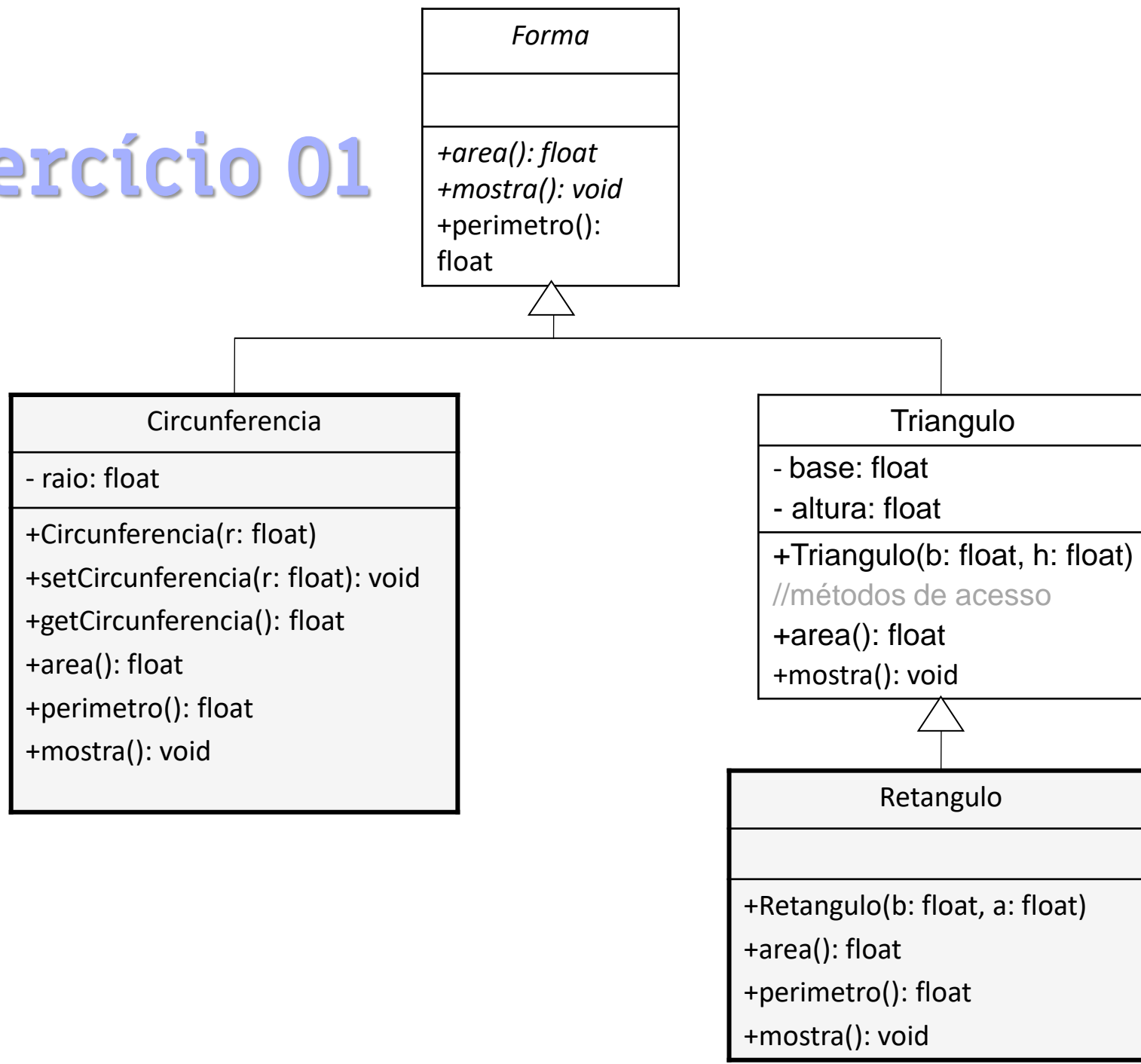
Outras classes como ContaCorrente, Alunos, FichaClinica, etc, podem implementar essa interface e terão os métodos da interface também.

# Interfaces x Classes Abstratas

Use classes abstratas quando você quiser definir um *"template"* para subclasses e você possui alguma implementação (métodos concretos) que todas as subclasses podem utilizar.

Use interfaces quando você quiser definir uma regra que todas as classes que implementem a interface devem seguir, independentemente se pertencem a alguma hierarquia de classes ou não.

# Exercício 01



# Exercício 01

Crie a classe abaixo como subclasse de Forma:

Circunferencia
- raio: float
+Circunferencia(r: float) +setCircunferencia(r: float): void +getCircunferencia(): float +area(): float +perimetro(): float +mostra(): void

O método `area()` deve retornar valor da área da circunferência, sabendo que  $area = \pi * r^2$

O método `perimetro()` deve retornar o valor do perímetro:  $perimetro = 2 * \pi * r$

Em ambos os métodos utilize a constante `Math.PI` da classe `Math`.

O método `mostra` deve exibir os valores de todos os atributos da classe

# Exercício 01

Crie a classe abaixo como subclasse de Triangulo:

Retangulo
+Retangulo(b: float, a: float) +area(): float +perimetro(): float +mostra(): void

O método `area()` deve retornar valor da área da circunferência, sabendo que  $\text{area} = \text{base} * \text{altura}$

O método `perimetro()` deve retornar o valor do perímetro:  $\text{perimetro} = (\text{base} * \text{altura}) * 2$

O método `mostra` deve exibir os valores de todos os atributos da classe



# Exercício 01

Instancie dois objetos na classe java principal, um da classe Circunferencia e outro da classe Retangulo, com os valores dos atributos digitados pelo usuário e utilize o construtor com parâmetros.

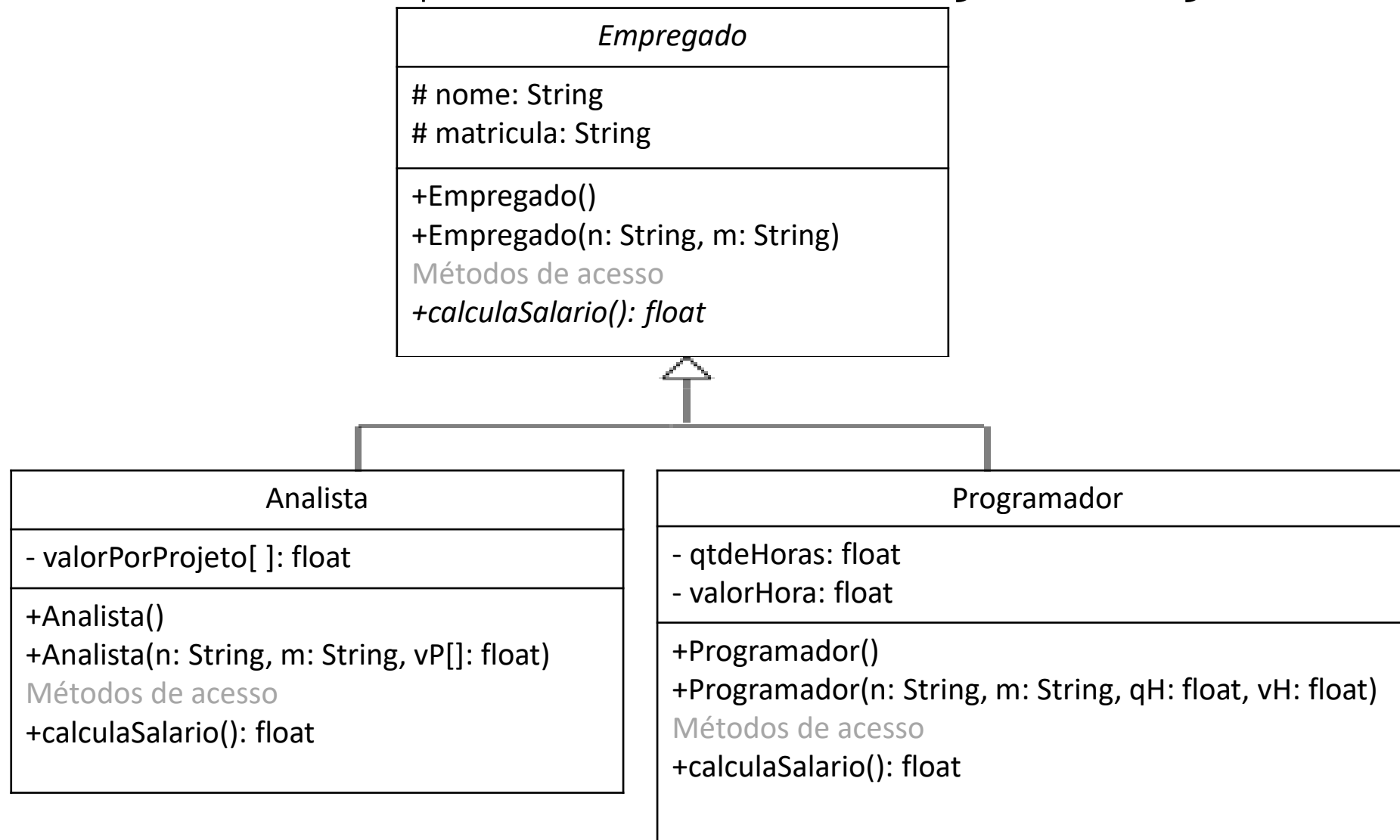
Mostre os dados de cada objeto através do método mostra().





# Exercício 02

Considere e implemente em Java o seguinte diagrama:





## Exercício 02

O método `calculaSalario()` da classe `Analista` calcula e retorna o valor do salário somando os valores dos projetos armazenados no vetor `valorPorProjeto`;

O método `calculaSalario()` da classe `Programador` retorna o valor da seguinte expressão,  $\text{valorHora} * \text{qtdeHoras}$ .



## Exercício 02

Crie, na classe java principal dois objetos, um a partir da classe Analista e outra da classe Programador, solicite os dados para o usuário e utilize o construtor com parâmetros.

Mostre, para ambos, o valor retornado pelo método calculaSalario(), juntamente com o nome e a matricula;

“Coragem é ir de  
falha em falha sem  
perder o  
entusiasmo”



Winston Churchill

# Obrigado!

**Se precisar ...**

Prof. Claudio Benossi

**[Claudio.benossi@fatec.sp.gov.br](mailto:Claudio.benossi@fatec.sp.gov.br)**

