

Técnicas de Programação I



Curso Superior de Tecnologia em Desenvolvimento de
Software Multiplataforma

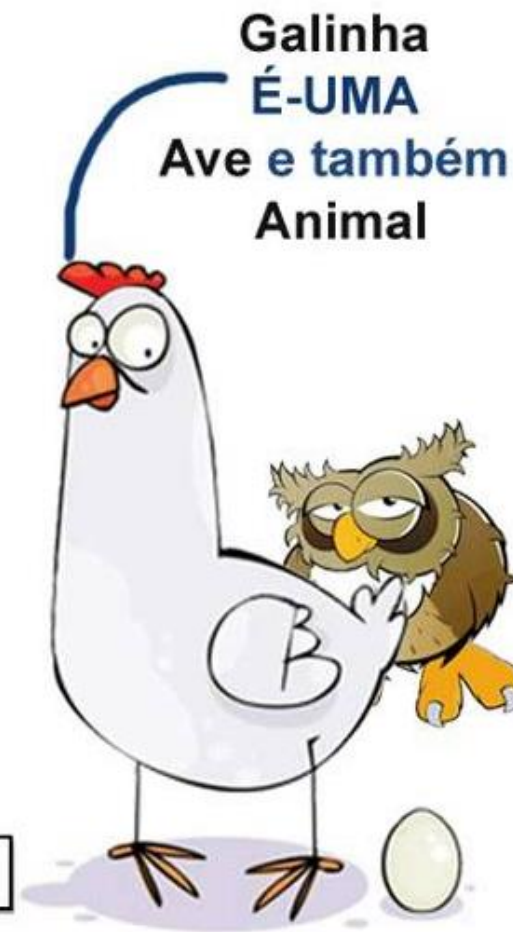
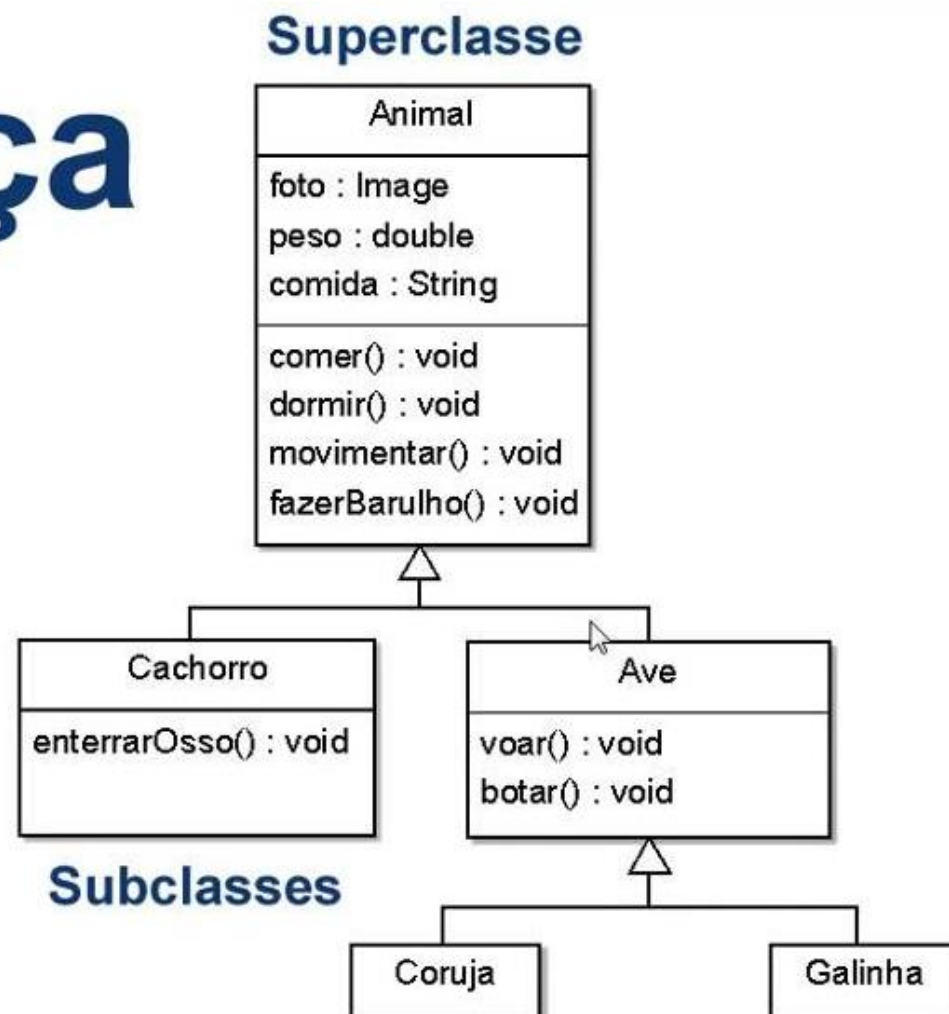
Aula 06

Prof. Claudio Benossi

Herança e Classes Abstratas

Na aula anterior ...

Herança



Na aula anterior ...

- ✓ 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.
- ✓ 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**.



Na aula anterior ...

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:

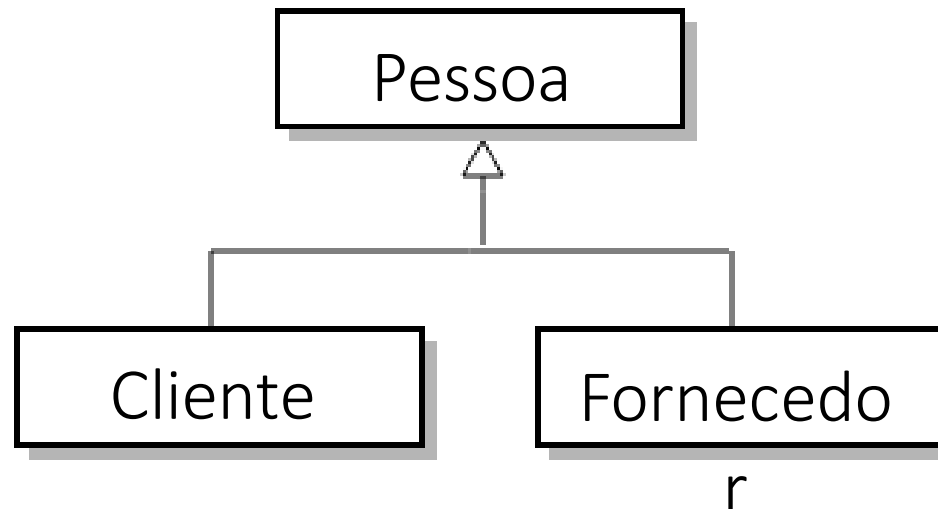
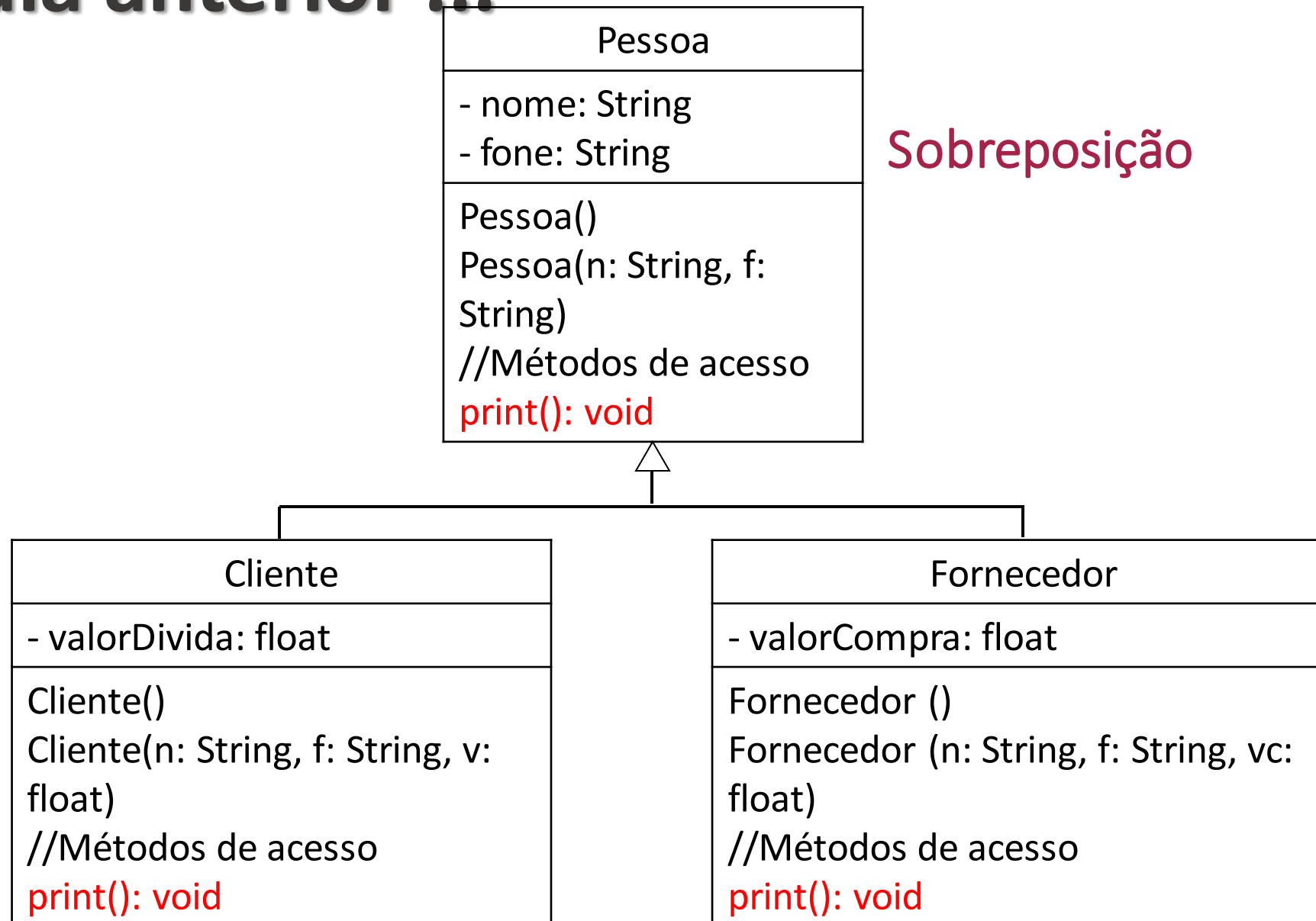


Diagrama UML
simplificado (não mostra
os métodos e atributos)

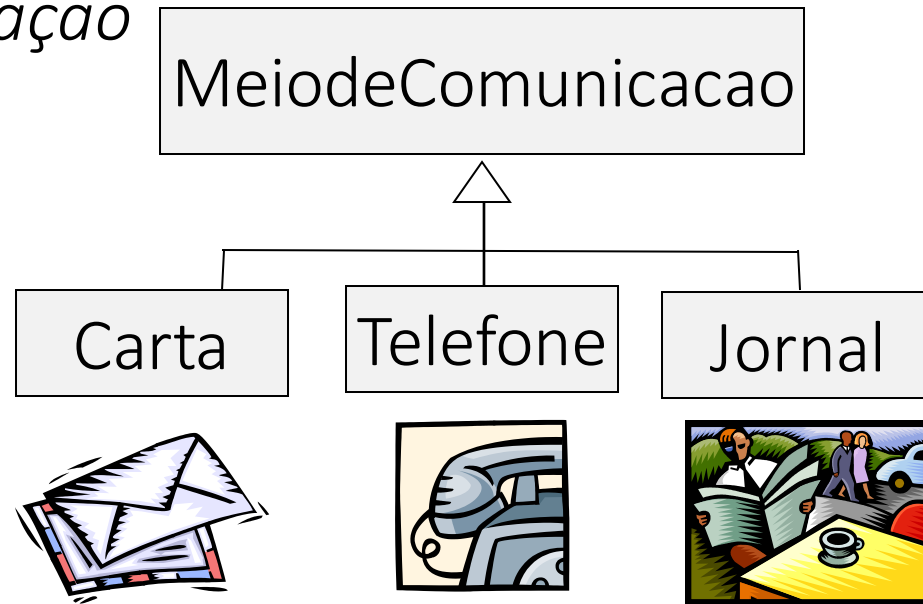
Na aula anterior ...



Na aula anterior ...

Nos ajuda a lidar com a complexidade.

Generalização



Especialização

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

Na aula anterior ...

Uma classe abstrata é uma classe que:

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

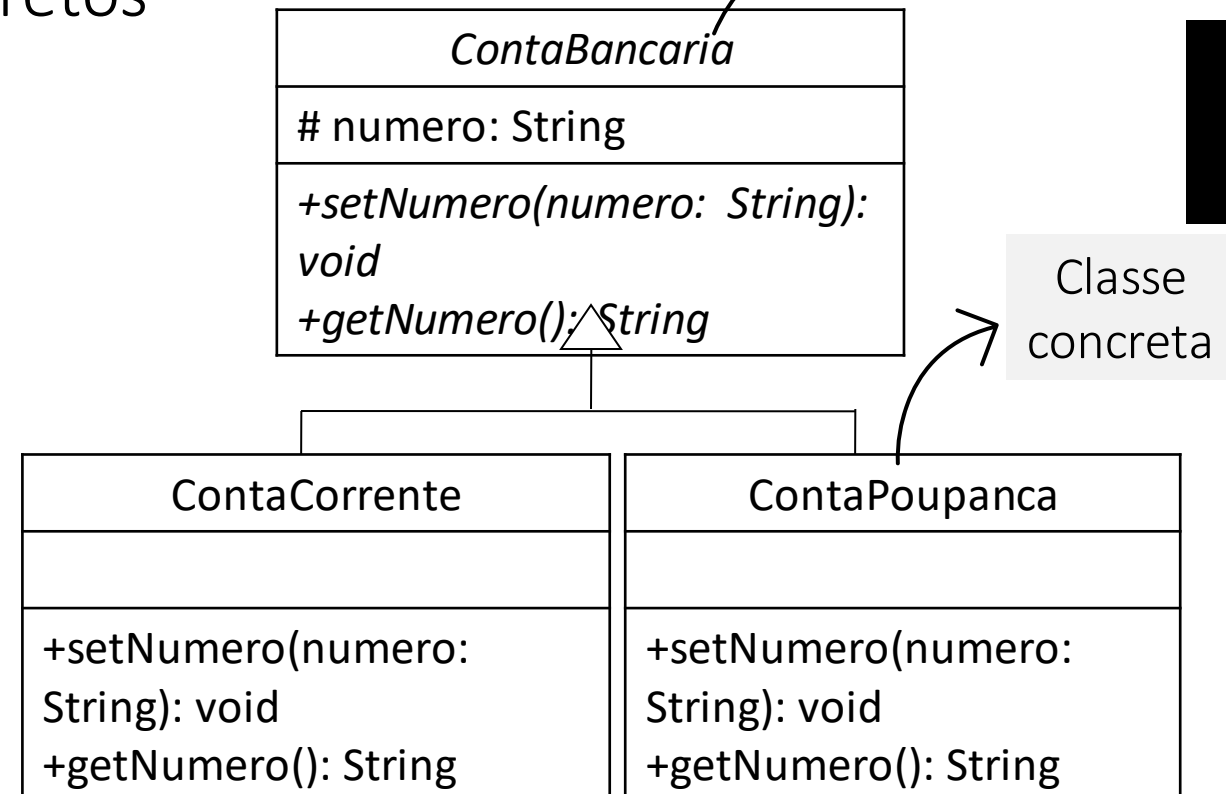
Você pode ter métodos concretos em uma classe abstrata

Classe abstrata

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



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



Na aula anterior ...

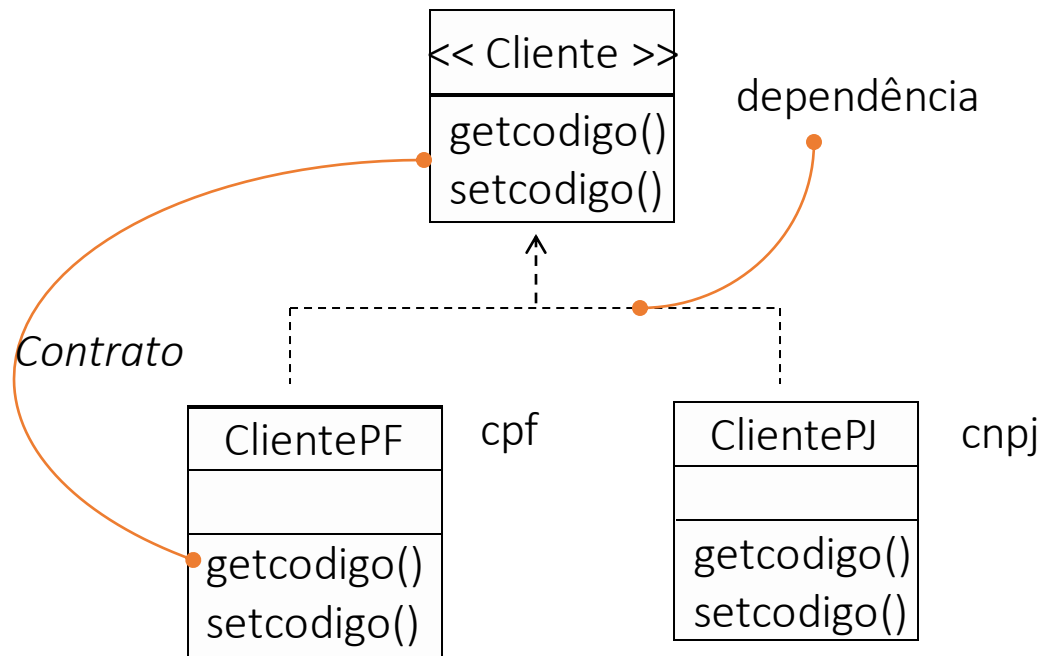
- ✓ São estruturas similares às classes abstratas, que definem a especificação de funcionalidades, sem a implementação das mesmas.
- ✓ Uma interface Java pode ser definida como uma “**classe abstrata pura**”, pois não pode possuir atributos (exceto constantes) nem definições de métodos, nem construtor.



Na aula anterior ...

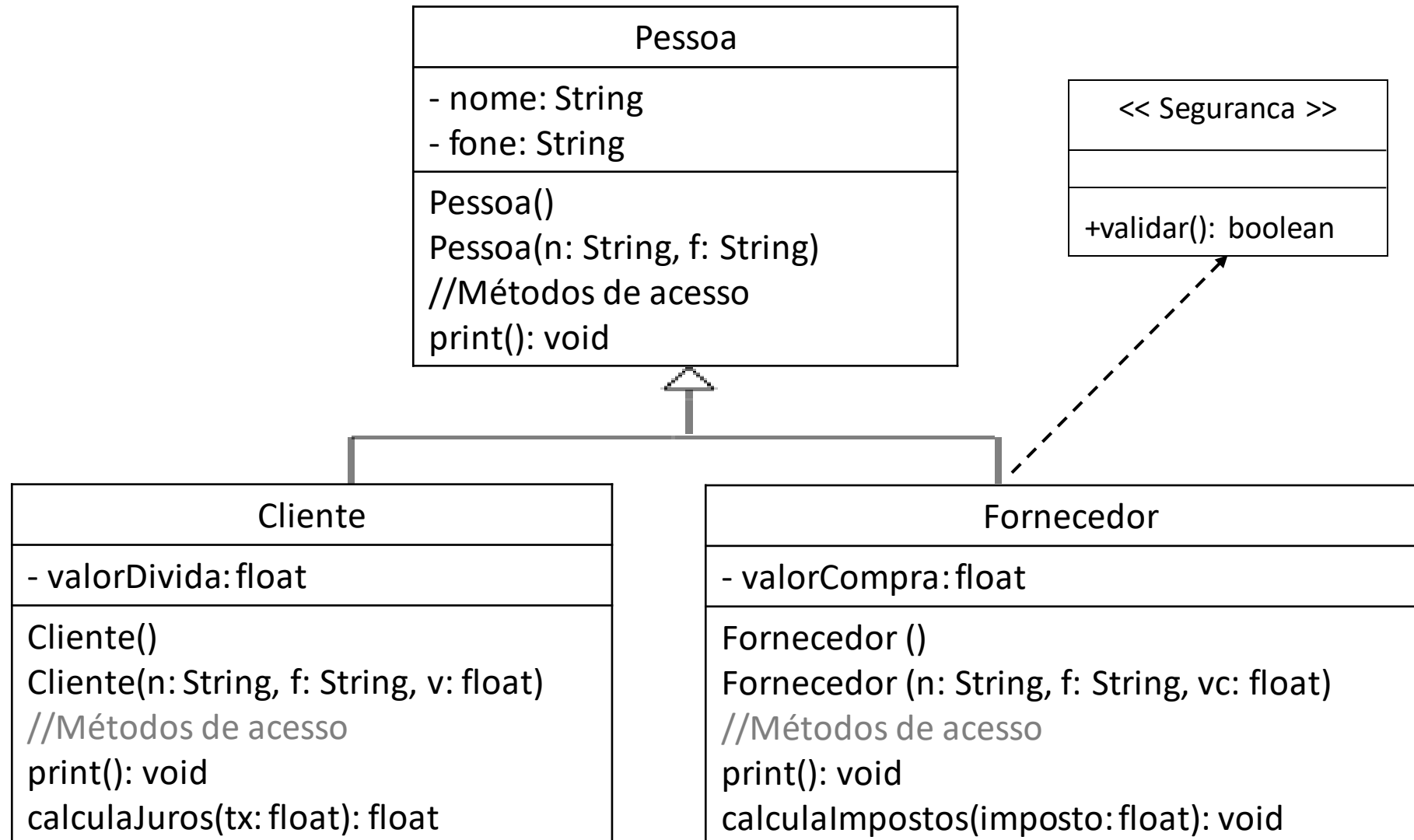
Uma interface estabelece uma espécie de contrato que é obedecido por uma classe.

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

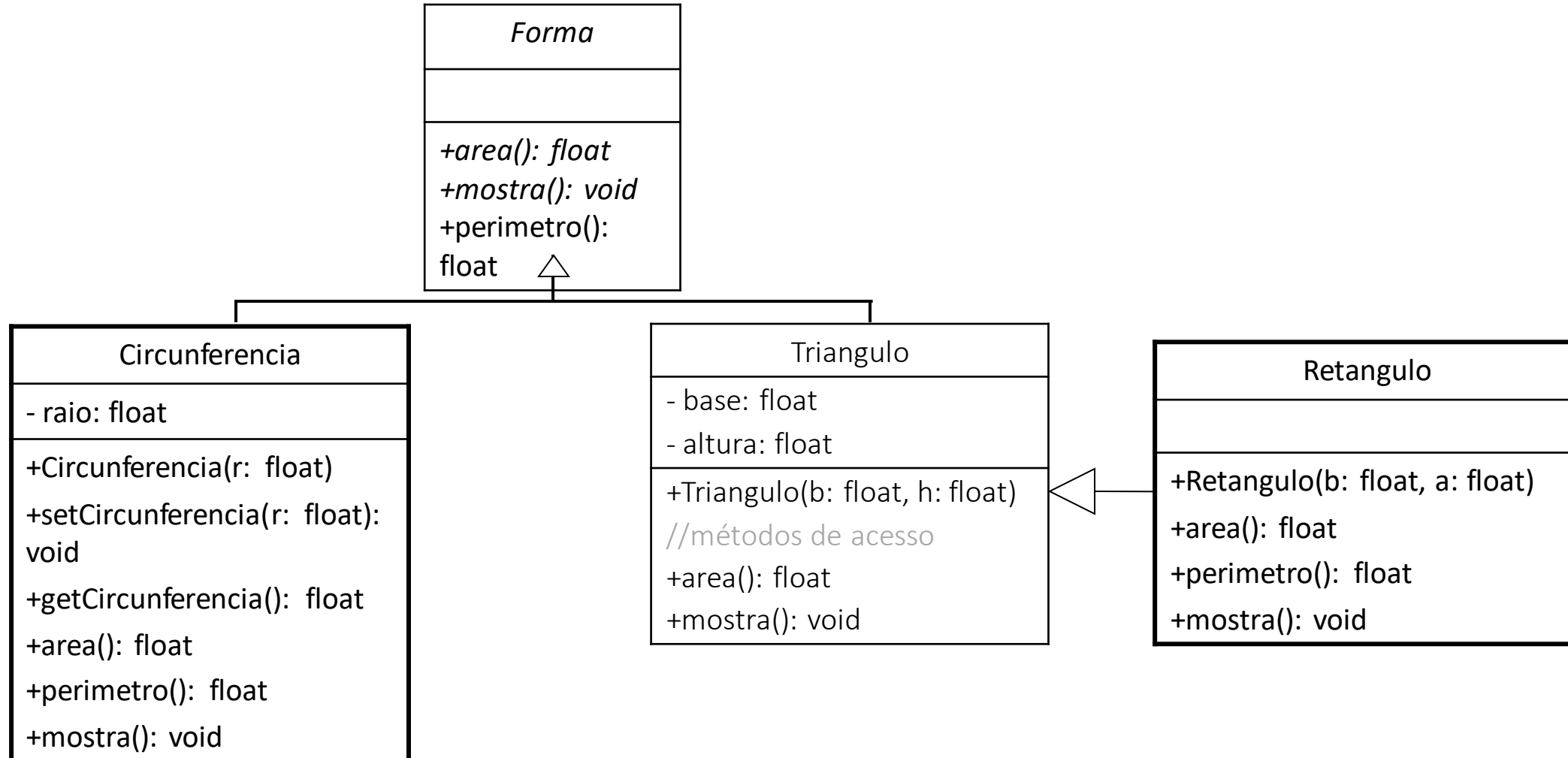


- ✓ Uma **classe pode implementar várias interfaces**, sendo um mecanismo elegante para se trabalhar com herança múltipla em Java.

Na aula anterior ...



Na aula anterior ... Exercício





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:

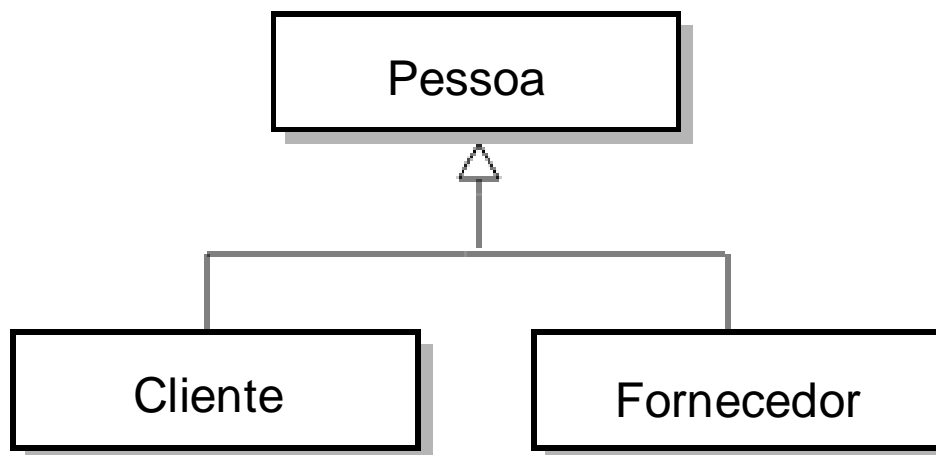
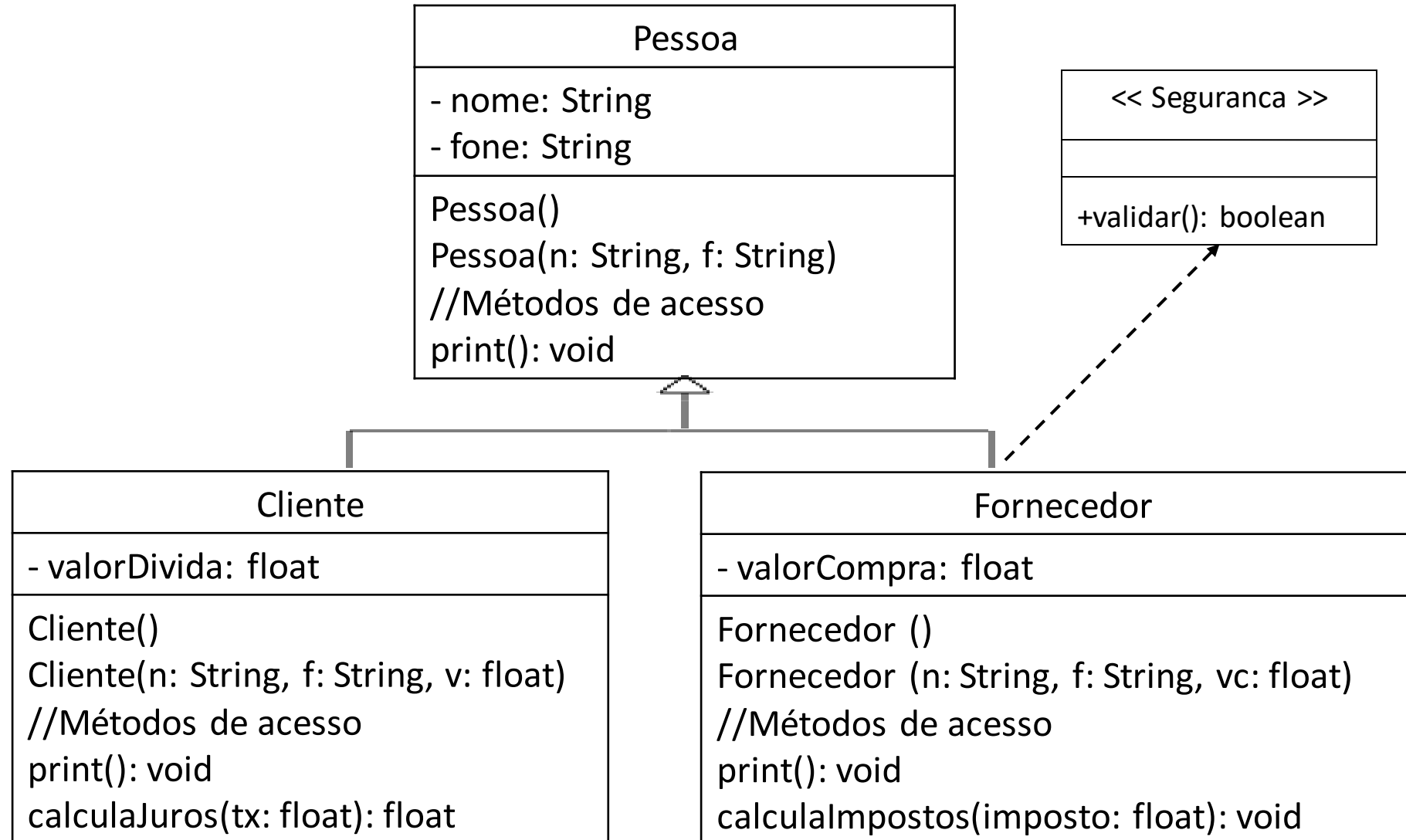


Diagrama UML
simplificado (não
mostra os métodos
e atributos)

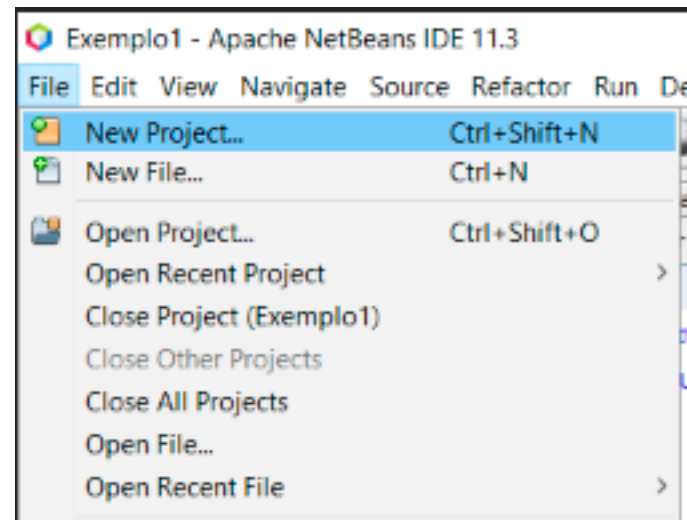
Interfaces - Exemplo



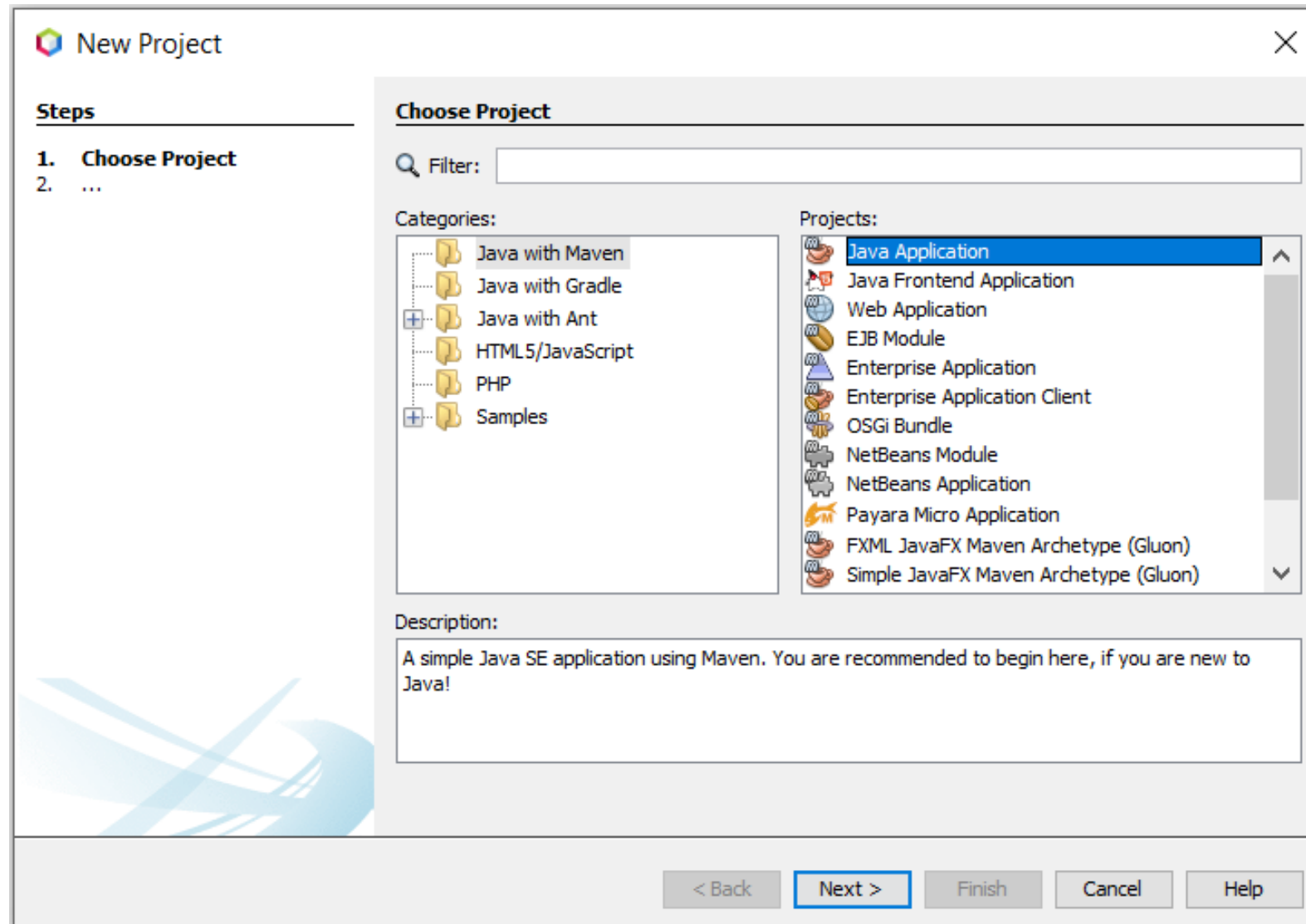
Interfaces - Exemplos

Vamos implementar usando o NetBeans.

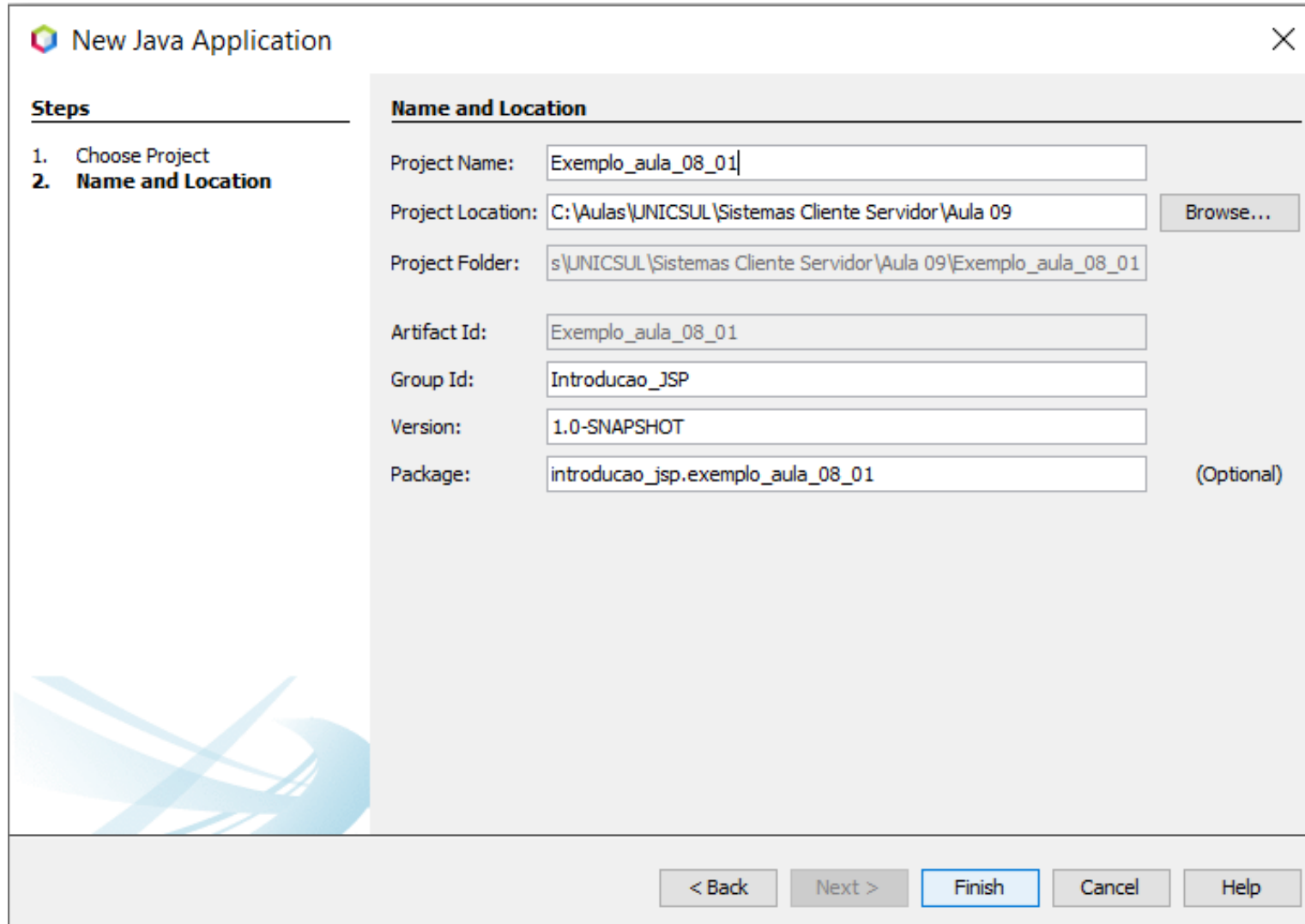
Vamos iniciar um novo projeto:



Interfaces - Exemplos



Interfaces - Exemplos



New Java Application

Steps

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

Name and Location

Project Name:

Project Location:

Project Folder:

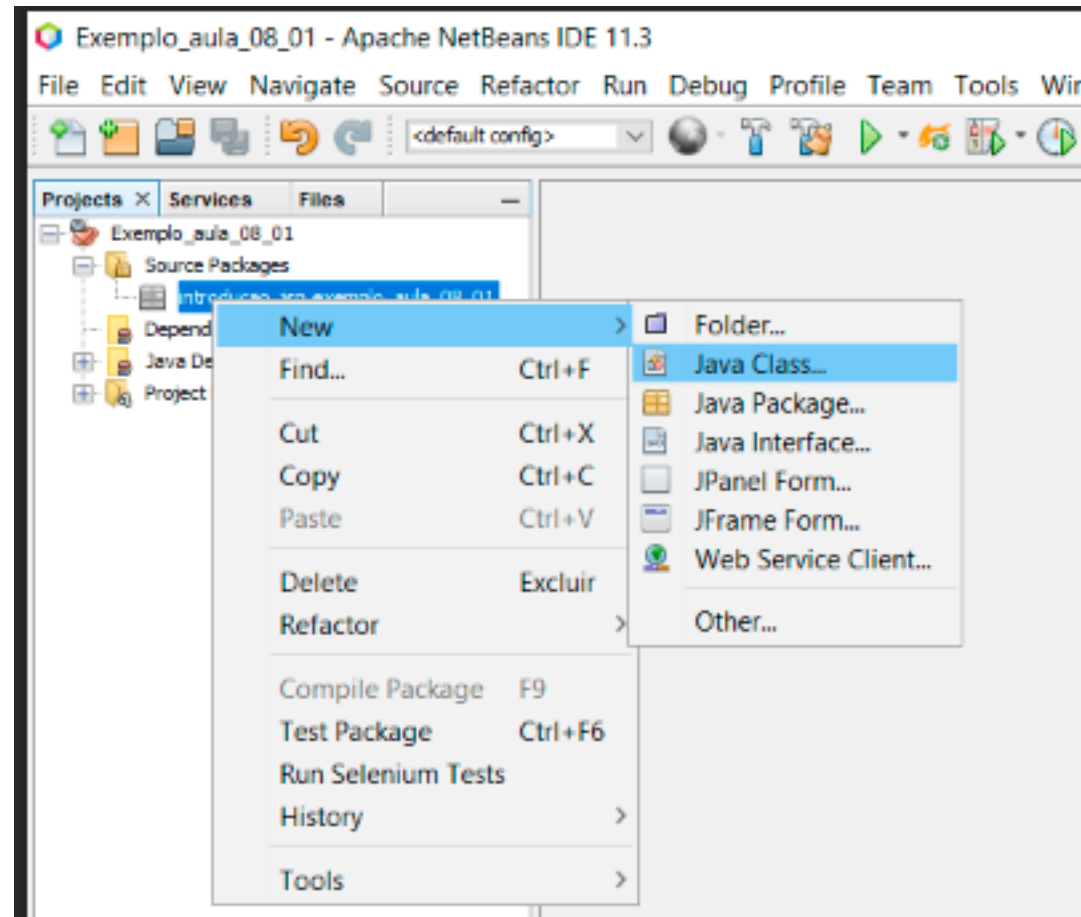
Artifact Id:

Group Id:

Version:

Package: (Optional)

Interfaces - Exemplos



Interfaces - Exemplos

Vamos criar as seguintes classes:

- ▶ Pessoa
- ▶ Fornecedor(extends Pessoa)
- ▶ Cliente(extends Pessoa)
- ▶ TestePessoa(public static void main(String[] args))

```
Pessoa.java x Fornecedor.java x TestePessoa.java x Cliente.java x
Source History

1  import javax.swing.JOptionPane;
2
3  public class Pessoa {
4      private String nome;
5      private String fone;
6
7      Pessoa() {}
8
9      Pessoa(String n, String f){
10         nome=n;
11         fone=f;
12     }
13
14     public String getNome() {
15         return nome;
16     }
17
18     public void setNome(String nome) {
19         this.nome = nome;
20     }
21
22     public String getFone() {
23         return fone;
24     }
25
26     public void setFone(String fone) {
27         this.fone = fone;
28     }
29
30     public void print() {
31         JOptionPane.showMessageDialog(null, "Dados \nNome: "
32                                     + nome + "\nTelefone: " + fone);
33     }
34 }
35 }
```



```

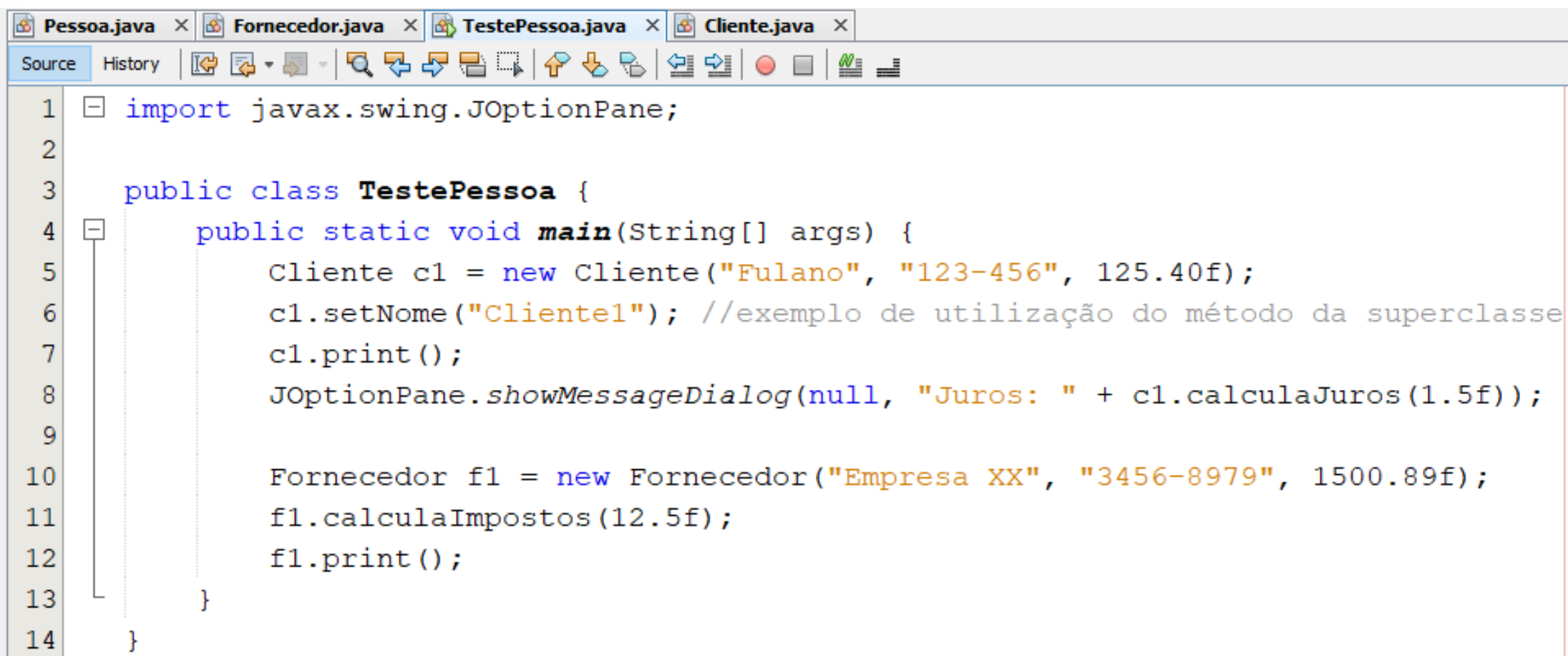
1  import javax.swing.JOptionPane;
2
3  public class Fornecedor extends Pessoa{
4      private float valorCompra;
5
6      Fornecedor() {}
7
8      Fornecedor(String n, String f, float vc){
9          super(n, f);
10         valorCompra=vc;
11     }
12
13     public float getValorCompra() {
14         return valorCompra;
15     }
16
17     public void setValorCompra(float valorCompra) {
18         this.valorCompra = valorCompra;
19     }
20
21     public void print() {
22         super.print();
23         JOptionPane.showMessageDialog(null, "\nValor da Compra: " + valorCompra);
24     }
25
26     public void calculaImpostos(float imposto){
27         valorCompra+=valorCompra*imposto/100;
28     }
29 }

```

```

1  import javax.swing.JOptionPane;
2  public class Cliente extends Pessoa {
3      private float valorDivida;
4      Cliente() {}
5      Cliente(String n, String f, float vd) {
6          super(n, f);
7          valorDivida = vd;
8      }
9      public float getValorDivida() {
10         return valorDivida;
11     }
12     public void setValorDivida(float valorDivida) {
13         this.valorDivida = valorDivida;
14     }
15     public void print() {
16         super.print();
17         JOptionPane.showMessageDialog(null, "\nValor da Divida: " + valorDivida);
18     }
19     public float calculaJuros(float tx) {
20         return valorDivida * tx / 100;
21     }
22 }

```



The image shows a screenshot of an IDE window with four tabs: Pessoa.java, Fornecedor.java, TestePessoa.java (active), and Cliente.java. The TestePessoa.java tab is selected, and the code is displayed in a source editor. The code is a Java program that tests the Cliente and Fornecedor classes. It imports javax.swing.JOptionPane and defines a public class TestePessoa with a main method. The main method creates a Cliente object c1, sets its name to "Cliente1", prints it, and shows a message dialog with the calculated interest. It also creates a Fornecedor object f1, calculates its taxes, and prints it.

```
1 import javax.swing.JOptionPane;
2
3 public class TestePessoa {
4     public static void main(String[] args) {
5         Cliente c1 = new Cliente("Fulano", "123-456", 125.40f);
6         c1.setNome("Cliente1"); //exemplo de utilização do método da superclasse
7         c1.print();
8         JOptionPane.showMessageDialog(null, "Juros: " + c1.calculaJuros(1.5f));
9
10        Fornecedor f1 = new Fornecedor("Empresa XX", "3456-8979", 1500.89f);
11        f1.calculaImpostos(12.5f);
12        f1.print();
13    }
14 }
```

Interfaces - Exemplos

Vamos testar a aplicação!

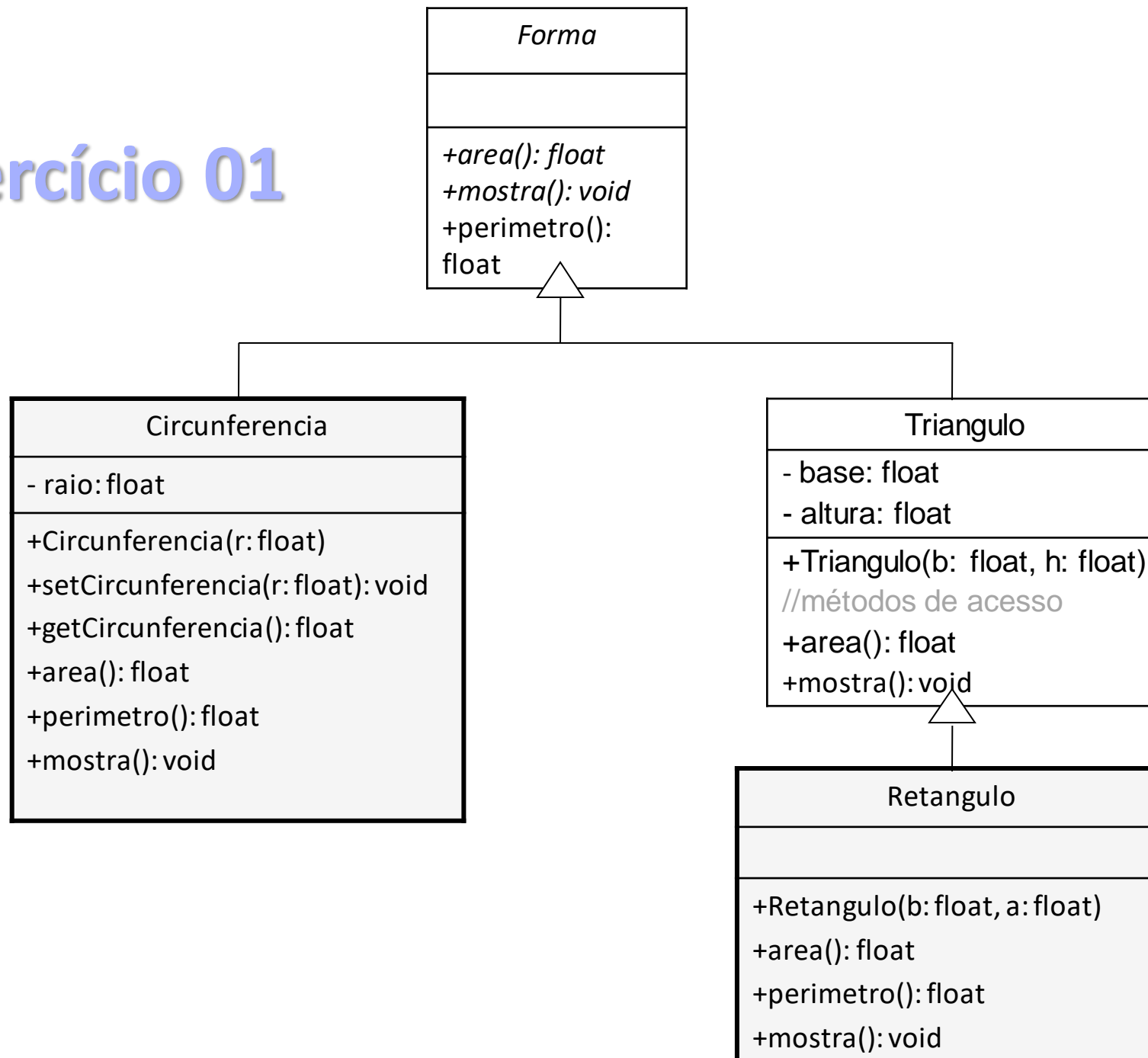


Interfaces x Classes Abstratas

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.

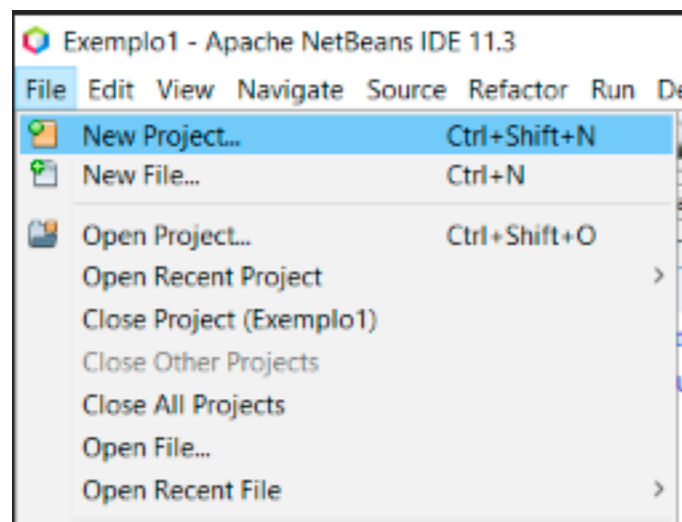
Mostre os dados de cada objeto através do método mostra().



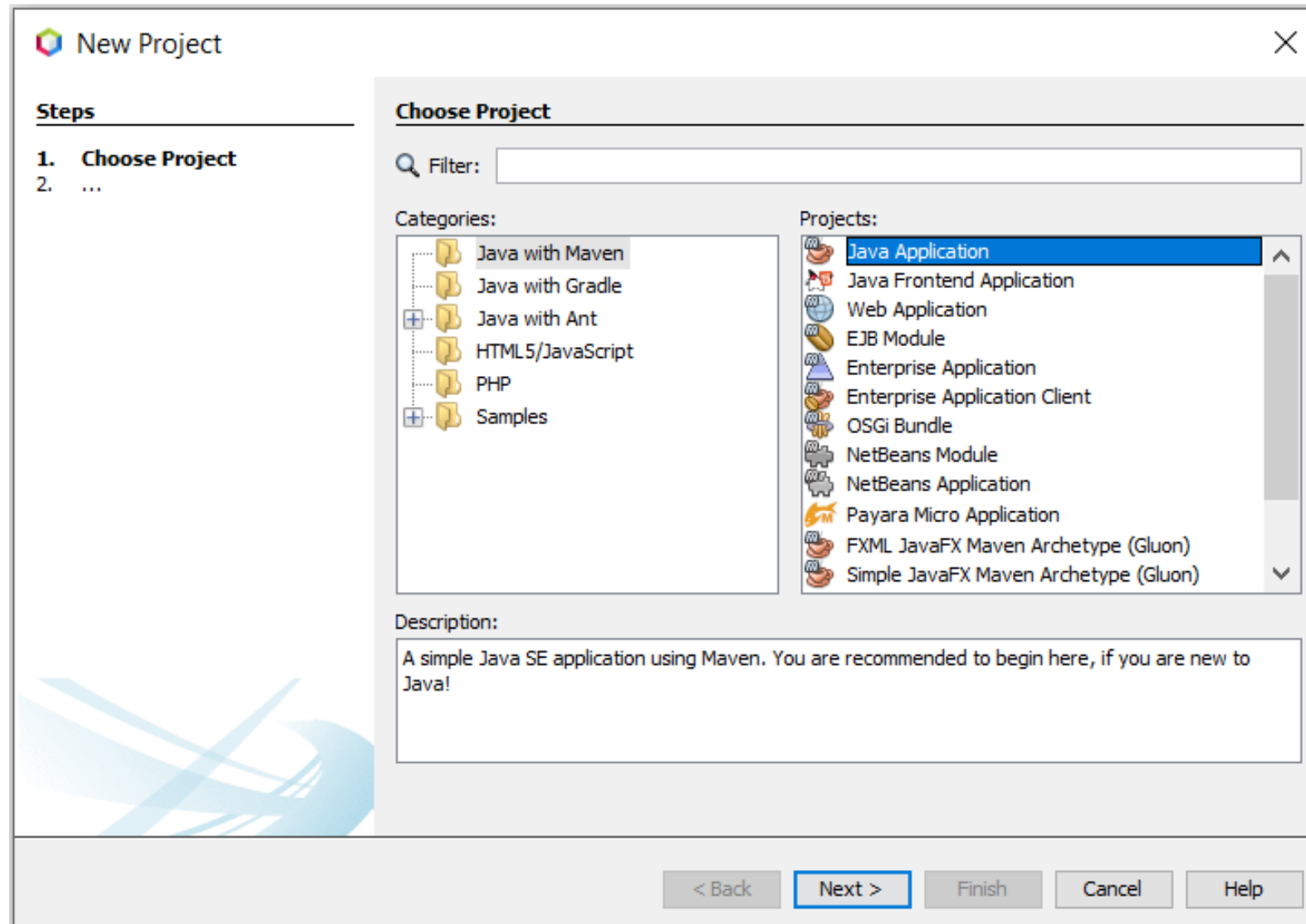
Exercício 01

Vamos implementar usando o NetBeans.

Vamos iniciar um novo projeto:



Exercício 01



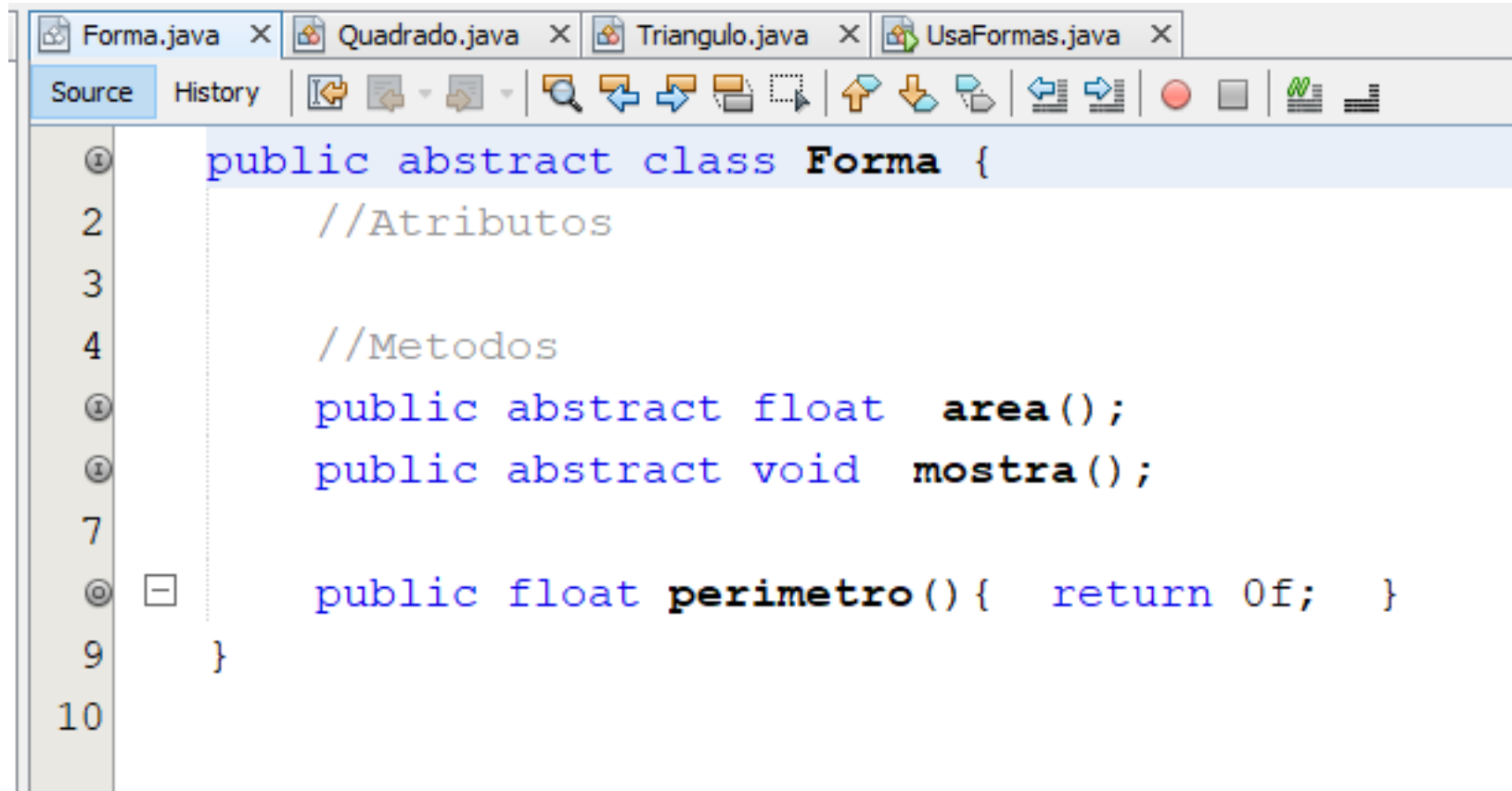


Exercício 01

Vamos criar as seguintes Classes:

- ▶ Forma (Classe Abstrata)
- ▶ Quadrado (extends Forma)
- ▶ Triangulo (extends Forma)
- ▶ UsaFormas (public static void main(String[] args))

Exercício 01



```
Forma.java x Quadrado.java x Triangulo.java x UsaFormas.java x
Source History
1 public abstract class Forma {
2     //Atributos
3
4     //Metodos
5     public abstract float area();
6     public abstract void mostra();
7
8     public float perimetro() { return 0f; }
9 }
10
```

```
1 public class Quadrado extends Forma {
2     //Atributos
3     private float base;
4
5     //Construtor
6     public Quadrado(float b){ base = b; }
7
8     //Metodos de acesso
9     public float getBase() { return base; }
10    public void setBase(float b) { base = b; }
11
12    //sobreposição do método da classe Pessoa
13    public float perimetro(){
14        return base * 4;
15    }
16
17    //Implementação dos métodos abstratos da classe Forma
18    public float area(){
19        return base * base;
20    }
21    public void mostra(){
22        System.out.println("Base: " + base + "\nPerimetro: " + perimetro() + "\nArea: " + area());
23    }
24 }
```



```
1 public class Triangulo extends Forma {
2     //Atributos
3     private float base, altura;
4
5     //Construtor
6     public Triangulo(float b, float h){
7         base = b;
8         altura = h;
9     }
10
11     //Metodos de acesso
12     public float getBase() { return base; }
13     public float getAltura() { return altura; }
14     public void setBase(float b) { base = b; }
15     public void setAltura(float h) { altura = h; }
16
17     //Implementação dos métodos abstratos da classe Forma
18     public float area(){
19         return (base * altura)/2;
20     }
21     public void mostra(){
22         System.out.println("\nBase: " + base + "\nAltura: " + altura + "\nArea: " + area());
23     }
24 }
```

```
Forma.java x Quadrado.java x Triangulo.java x UsaFormas.java x
Source History
1 import java.util.*;
2 public class UsaFormas{
3     public static void main(String args[]){
4         float b,a;
5         Quadrado q;
6         Triangulo t;
7
8         Scanner scan = new Scanner(System.in);
9
10        System.out.print("Digite a base do quadrado: ");
11        b = scan.nextFloat(); //para String use o nextLine()
12        q = new Quadrado(b);
13        //na chamada do metodo abaixo e passado um objeto da classe Quadrado
14        q.mostra();
15
16        System.out.print("Digite a base do triangulo: ");
17        b = scan.nextFloat();
18        System.out.print("Digite a altura do triangulo: ");
19        a = scan.nextFloat();
20        t = new Triangulo(b,a);
21        //na chamada do metodo abaixo e passado um objeto da classe Triangulo
22        t.mostra();
23    }
24 }
```

Exercício 01

Vamos testar nossa aplicação!



Estrutura de Repetição

Desafio

Faça um programa que solicite ao usuário um número, calcule e mostre a tabuada desse número.

```
import java.util.Scanner;
public class Tabuada {
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner sc = new Scanner(System.in);
        System.out.println("Entre com um número inteiro: ");
        int x = sc.nextInt();
        System.out.println(x + " x 0 = " + x * 0);
        System.out.println(x + " x 1 = " + x * 1);
        System.out.println(x + " x 2 = " + x * 2);
        System.out.println(x + " x 3 = " + x * 3);
        System.out.println(x + " x 4 = " + x * 4);
        System.out.println(x + " x 5 = " + x * 5);
        System.out.println(x + " x 6 = " + x * 6);
        System.out.println(x + " x 7 = " + x * 7);
        System.out.println(x + " x 8 = " + x * 8);
        System.out.println(x + " x 9 = " + x * 9);
        System.out.println(x + " x 10 = " + x * 10);
    }
}
```

Estruturas de repetição

Também conhecidas como laços (loop).

Utilizadas para executar repetidamente uma instrução ou um bloco de instruções enquanto uma determinada condição for verdadeira.

As estruturas de repetição em Java são:

- **for**
- **while**
- **do ... while**



Estruturas de repetição

Para determinarmos qual é a estrutura mais adequada, devemos saber:

- ✔ o número de vezes que o trecho programa vai ser executado: laços contados
- ✔ ou a condição para que ela aconteça: laços condicionais

Laços contados: um **contador** irá auxiliar no laço. Neste laço, a repetição da estrutura repete-se até que o contador atinja o limite estipulado na condição.

Laços condicionais: o valor é desconhecido e devemos utilizar uma variável com valor predefinido em uma condição dentro do laço para finalizarmos a repetição.



Estrutura de repetição

Independente do tipo de laço, todos são constituídos de três partes:

- ✓ Inicialização(ões) da(s) variável(is) de controle
- ✓ Condição(ões)
- ✓ Atualização da(s) variável(is) de controle



Estrutura de Repetição

Quando tivermos que repetir um trecho de um programa por determinado número de vezes, precisaremos do auxílio de uma estrutura de repetição.

As estruturas de repetição se dividem em **ENQUANTO**, **REPITA**, **PARA** e para determinarmos qual é a estrutura mais adequada para um determinado programa, devemos saber qual o número de vezes que o trecho programa vai ser executado (**laços contados**) ou a condição para que ela aconteça (**laços condicionais**).

Estrutura de Repetição

Na linguagem Java os comandos que implementam estas estruturas de repetição são: **while**, **do-while** e **for**.

Estrutura de Repetição

Estrutura de repetição while

Comando da linguagem Java.

{ iniciar a variável de controle }

while (condição) {

{instruções}

{atualizar a variável de controle}

};

Obs: as chaves { } são necessárias para uma estrutura de bloco (quando desejamos repetir mais de um comando). Se formos repetir um comando simples poderíamos omitir estas chaves.

Estrutura de Repetição

```
import java.util.Scanner;
```

```
public class RepeticaoWhile {
    public static void main(String[] args) {
        Scanner leitura = new Scanner(System.in);
        int A = 1, soma = 0, cont = 0;
        while (A > 0) {
            A = leitura.nextInt();
            if (A > 0) {
                soma = soma + A;
                cont++;
            }
        }
        System.out.println("A média é: " + (soma/cont));
    }
}
```

Condição de execução

Variável
Contador

Variável
Acumulador

Estrutura de Repetição

Estrutura de repetição do-while

Comando da linguagem Java.

```
{ iniciar a variável de controle }  
do {  
    {instruções}  
    {atualizar a variável de controle}  
} while (condição);
```

Obs: as chaves { } são necessárias para uma estrutura de bloco (quando desejamos repetir mais de um comando). Se formos repetir um comando simples poderíamos omitir estas chaves.

Exemplos de Estrutura de Repetição

```
import java.util.Scanner;
```

```
public class RepeticaoDoWhile {  
    public static void main(String[] args) {  
        Scanner leitura = new Scanner(System.in);  
        float A, soma = 0;  
        String resp;  
        int cont = 0;  
        do {  
            System.out.println("Informe um número");  
            A = leitura.nextFloat();  
            if (A > 0) {  
                soma = soma + A;  
                cont++;  
            }  
            System.out.println("Deseja continuar (S ou N)?");  
            resp = leitura.next();  
        }  
        while (resp.equalsIgnoreCase("s"));  
        System.out.println("A média é: " + (soma/cont));  
    }  
}
```

Lê o que foi
digitado como valor
float

Variável
Contador

Lê a resposta que
define se a
repetição
continua

Variável
Acumulador

Condição de execução

Estrutura de Repetição

while **vs** do-while

while vs do-while

while (em PT: enquanto) **verifica antes** a condição para executar o bloco. Já o do-while executa o bloco e **verifica depois** a condição para repetir o laço.

GEEK2CODE_

```
while (!naBorda()) {  
    corra();  
}
```

```
do {  
    corra();  
} while (!naBorda());
```



*Ambos iniciando a corrida na borda

Estrutura de Repetição

Estrutura de repetição for

Essa estrutura precisa de uma variável para controlar a contagem do ciclo, que ocorre na própria estrutura.

Observe que há uma economia de instruções, pois a própria estrutura se encarrega de iniciar, testar a condição e atualizar a variável que controla o laço.

```
for (inicialização; condição; atualização) {  
    {instruções}  
}
```


Estrutura de Repetição

Permite a leitura de entrada de dados no prompt

```
import java.util.Scanner;
```

```
public class RepeticaoFor {  
    public static void main(String[] args) {
```

```
        Scanner leitura = new Scanner(System.in);  
        int soma = 0, cont, A;  
        for (cont=0; cont<2; cont++) {
```

```
            A = leitura.nextInt();
```

```
            soma += A;
```

```
        }
```

```
        System.out.println("A média é: " +(soma/cont));
```

```
    }
```

```
}
```

Condição de execução

Variável Contador

Variável Acumulador

Lê o que foi digitado como valor inteiro

Estrutura de Repetição

Estrutura de repetição for

```
for(inicialização; condição; atualização) {  
    {instruções}  
}
```

O laço contado **for** funciona da seguinte maneira:

- no início da execução do laço a inicialização é executada.
- A seguir, a condição é testada.
- Se o resultado do teste for falso as instruções não são executadas e a execução do algoritmo prossegue pelo primeiro comando seguinte ao comando for.
- Por outro lado, se o valor do teste for verdadeiro, então as instruções são executadas e ao final das instruções é feita a atualização da variável de controle.

Estrutura de Repetição

Estrutura de repetição for

Obs: as chaves { } são necessárias para uma estrutura de bloco (quando desejamos repetir mais de um comando).

Se formos repetir um comando simples poderíamos omitir estas chaves.



Desafio

Adivinhe meu número: Crie um jogo onde o computador escolhe um número inteiro aleatório entre 0 e 20.

- ✔ Leia a entrada do usuário para tentar acertar o número;
- ✔ Se errar informar ao usuário se o número é maior ou menor;
- ✔ Repetir até o usuário acertar.

Observação: Para gerar um número aleatório utilize a classe Random:

```
Random aleatorio = new Random();  
    int num = aleatorio.nextInt(20);
```

Desafio

Desafio 2: Adivinhe o número

```
import java.util.Scanner;
import java.util.Random;
class Main {
    public static void main(String[] args) {
        //instância um objeto da classe Random usando o construtor básico
        Random gerador = new Random();
        int num = gerador.nextInt(20);
        int x, i = 0;
        Scanner sc = new Scanner(System.in);
        do{
            i++;
            System.out.println("Digite um número entre 0 e 20:");
            x = sc.nextInt();
            if(x == num){
                System.out.println("Parabéns, você acertou em " + i + " tentativas" );
            } else if (x < num) {
                System.out.println("O número pensado é maior");
            } else {
                System.out.println("O número pensado é menor");
            }
        } while (x != num);
    }
}
```



Exercícios - Estrutura de Repetição

01-) Desenvolva um programa na linguagem Java que leia a quantidade de valores que serão processados e depois leia os valores e calcule a média dos mesmos.

Utilize a estrutura de repetição PARA.



Exercícios - Estrutura de Repetição

01-) Algoritmo:

algoritmo Exercio_01

inteiro: i, n

real: valor, soma

inicio

escrever ("Digite a quantidade de valores a serem processados:")

ler (n)

soma <- 0

para (i = 1; i <= n; i++)

 escrever ("Digite um valor: ")

 ler (valor)

 soma <- soma + valor

fim para

escrever ("A média dos valores digitados é: " + soma / n)

fim

ExemploFor.java

SourceHistory

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

```
import javax.swing.JOptionPane;

public class ExemploFor {

    public static void main(String arg[]){
        int i, n;
        double valor, soma;

        n = Integer.parseInt(JOptionPane.showInputDialog(null,
            "Digite a quantidade de valores que serão processados:",
            "Dado", JOptionPane.INFORMATION_MESSAGE));
        soma = 0;

        for(i=1; i<=n; i++){
            valor = Double.parseDouble(JOptionPane.showInputDialog(null,
                "Digite um valor:",
                "Dado", JOptionPane.INFORMATION_MESSAGE));
            soma = soma + valor;
        }
        JOptionPane.showMessageDialog(null, "A média dos valores digitados é: " + soma / n, "Resultado",
            JOptionPane.INFORMATION_MESSAGE);
    }
}
```

60

Exercícios - Estrutura de Repetição

02-) Desenvolva um programa na linguagem Java que leia um grupo de valores (não sabemos quantos são) para calcular e visualizar a média desses valores e, também, determinar e visualizar o maior deles.

Utilize uma estrutura de repetição ENQUANTO ou REPITA.

Exercícios - Estrutura de Repetição

02-) Algoritmo:

algoritmo Exercicio_02

inteiro: i, n

real: valor, soma

inicio

escrever ("Digite a quantidade de valores a serem processados:")

ler (n)

soma <- 0

i <- 1

enquanto (i <= n)

 escrever ("Digite um valor: ")

 ler (valor)

 soma <- soma + valor

 i <- i + 1

fim para

escrever ("A média dos valores digitados é: " + soma / n)

fim

```
1  import javax.swing.JOptionPane;
2
3  public class ExemploFor {
4
5      public static void main(String arg[]){
6          int i, n;
7          double valor, soma;
8
9          n = Integer.parseInt(JOptionPane.showInputDialog(null,
10              "Digite a quantidade de valores que serão processados:",
11              "Dado", JOptionPane.INFORMATION_MESSAGE));
12          soma = 0;
13          i = 1;
14          while(i<=n){
15              valor = Double.parseDouble(JOptionPane.showInputDialog(null,
16                  "Digite um valor:",
17                  "Dado", JOptionPane.INFORMATION_MESSAGE));
18              soma = soma + valor;
19              i = i + 1;
20          }
21          JOptionPane.showMessageDialog(null, "A média dos valores digitados é: " + soma / n, "Resultado",
22              JOptionPane.INFORMATION_MESSAGE);
23      }
24  }
```



Estrutura de Repetição

Vamos desenvolver uma aplicação para desenvolver a tabuada de acordo com o valor apresentado pelo usuário:

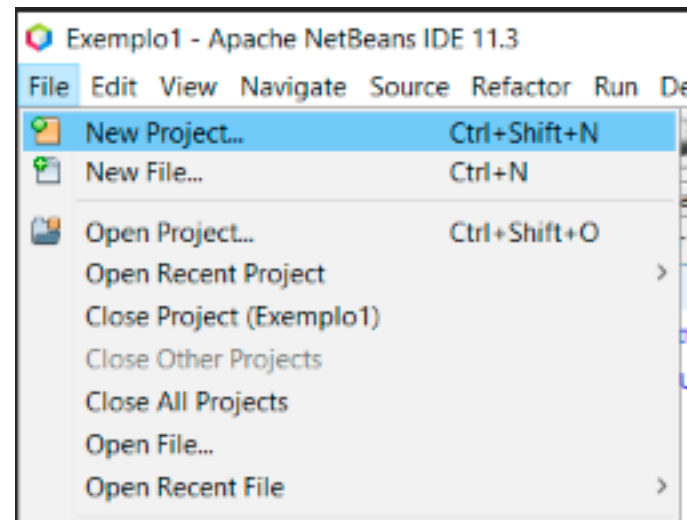
Tabuada
- numero: int - operador: char
+ Tabuada(numero:int, operador: char) + getNumero(): int + getOperador(): char + setNumero(numero: int): void + setOperador(operador: char): void + geraTabuada(): String



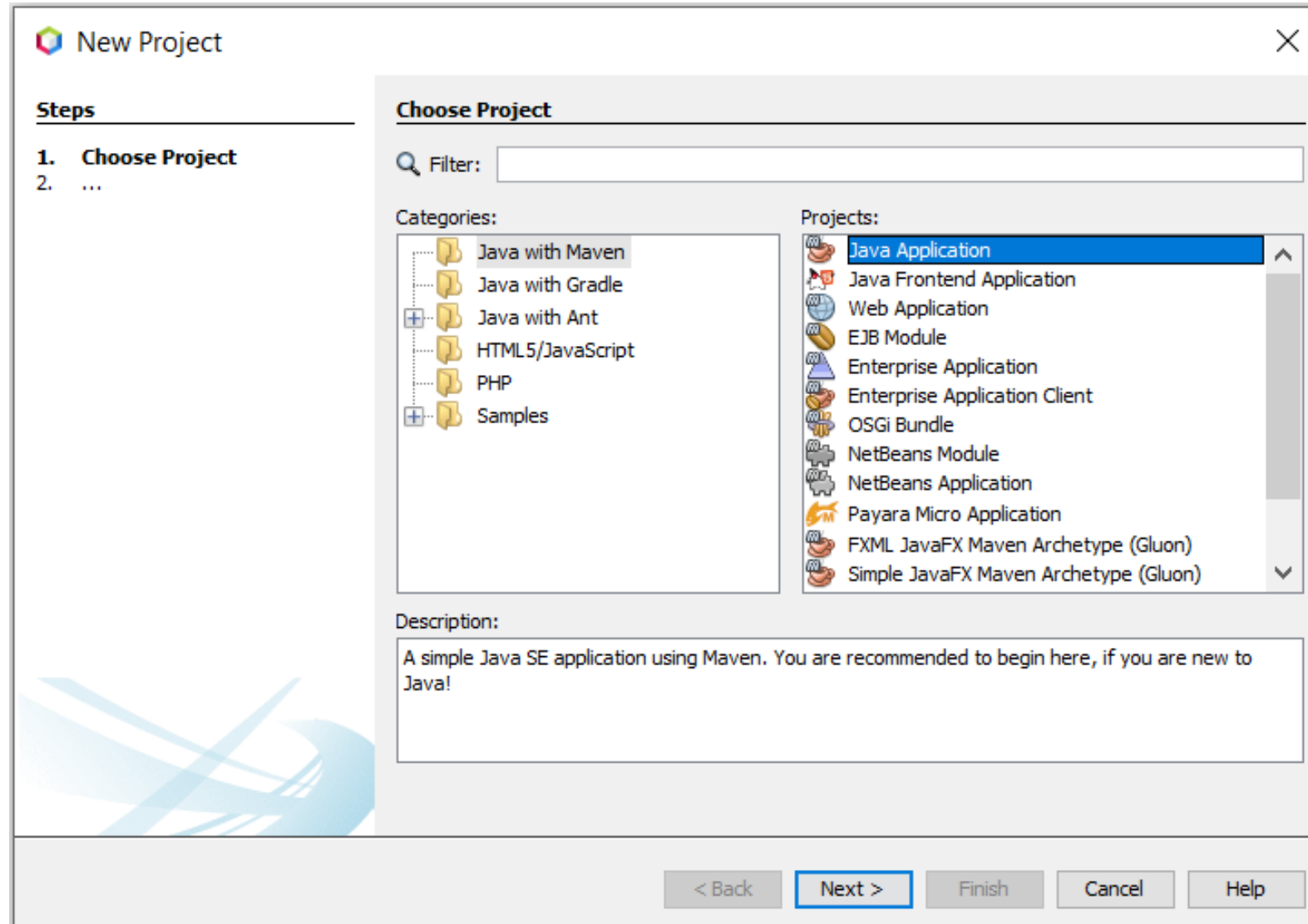
Estrutura de Repetição

Vamos implementar usando o NetBeans.

Vamos iniciar um novo projeto:



Estrutura de Repetição

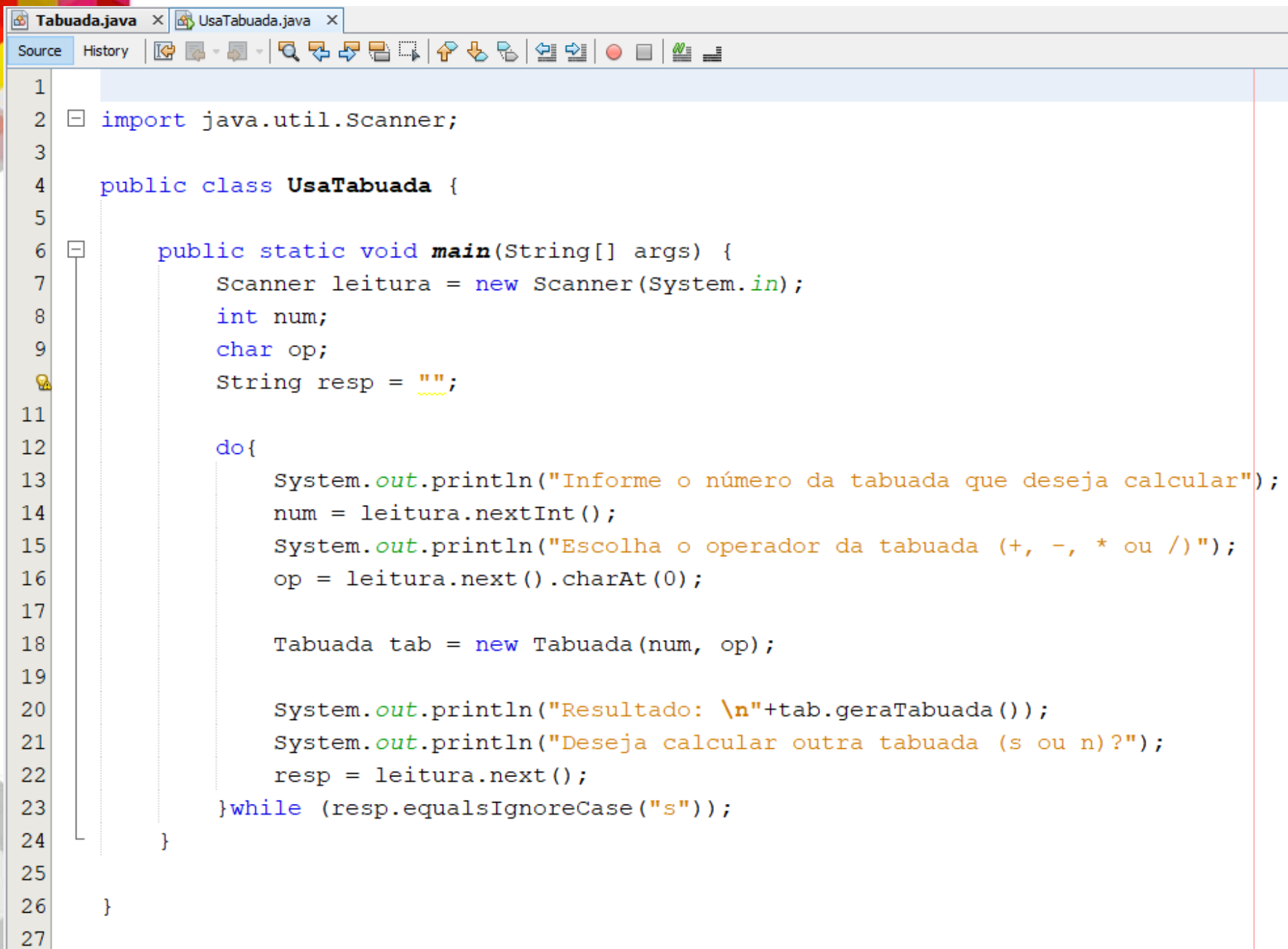


Estrutura de Repetição

Vamos criar as seguintes Classes:

- ▶ Tabuada
- ▶ UsaTabuada(`public static void main(String[] args)`)

```
1 public class Tabuada {
2     private int numero;
3     private char operador;
4
5     public Tabuada(int numero, char operador) {
6         this.numero = numero;
7         this.operador = operador;
8     }
9
10    public int getNumero() { return numero; }
11    public char getOperador() { return operador; }
12    public void setNumero(int numero) { this.numero = numero; }
13    public void setOperador(char operador) { this.operador = operador; }
14
15    public String geraTabuada() {
16        String resposta="";
17        switch(operador){
18            case '+':
19                for(int i = 1; i <= 10; i++){
20                    resposta += numero + " + " + operador + " " + i + " = " + (numero+i) + "\n";
21                }
22                break;
23            case '-':
24                for(int i = 1; i <= 10; i++){
25                    resposta += numero + " - " + operador + " " + i + " = " + (numero-i) + "\n";
26                }
27                break;
28            case '*':
29                for(int i = 1; i <= 10; i++){
30                    resposta += numero + " * " + operador + " " + i + " = " + (numero*i) + "\n";
31                }
32                break;
33            case '/':
34                for(int i = 1; i <= 10; i++){
35                    resposta += numero + " / " + operador + " " + i + " = " + (numero/i) + "\n";
36                }
37                break;
38        }
39
40        return resposta;
41    }
42 }
```

The image shows a screenshot of an IDE with two tabs: 'Tabuada.java' and 'UsaTabuada.java'. The 'UsaTabuada.java' tab is active, displaying the following Java code:

```
1
2 import java.util.Scanner;
3
4 public class UsaTabuada {
5
6     public static void main(String[] args) {
7         Scanner leitura = new Scanner(System.in);
8         int num;
9         char op;
10        String resp = "";
11
12        do{
13            System.out.println("Informe o número da tabuada que deseja calcular");
14            num = leitura.nextInt();
15            System.out.println("Escolha o operador da tabuada (+, -, * ou /)");
16            op = leitura.next().charAt(0);
17
18            Tabuada tab = new Tabuada(num, op);
19
20            System.out.println("Resultado: \n"+tab.geraTabuada());
21            System.out.println("Deseja calcular outra tabuada (s ou n)?");
22            resp = leitura.next();
23        }while (resp.equalsIgnoreCase("s"));
24    }
25
26 }
27
```

“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

