



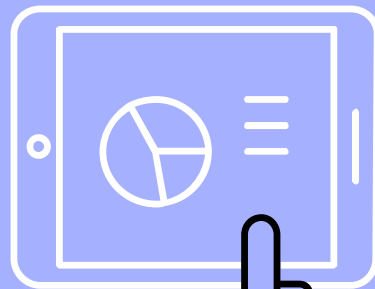
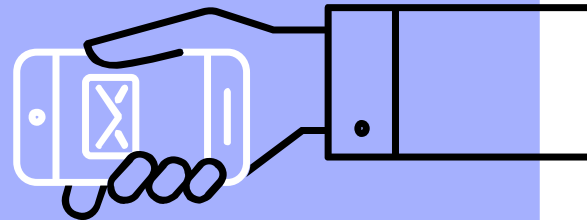
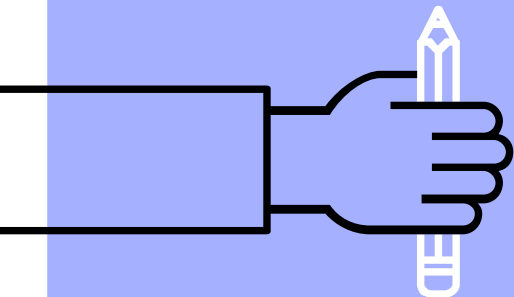
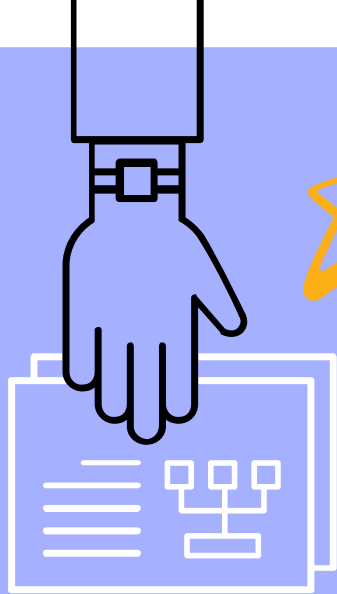
Universidade

Cruzeiro do Sul

Programação Orientada a Objetos

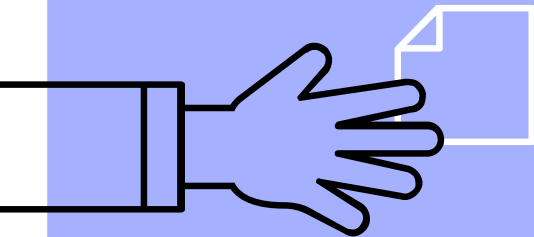
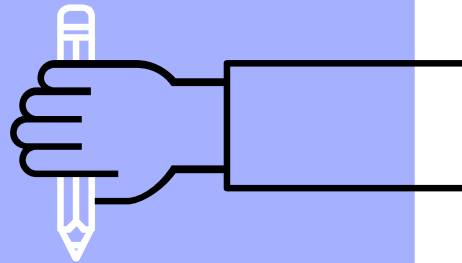
Aula 08

Prof.^a Erika Miranda



8. Unidade

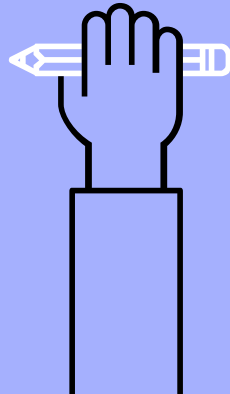
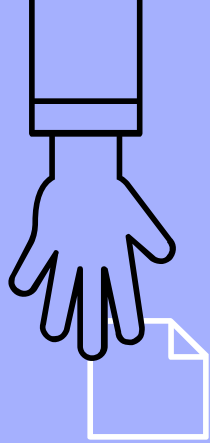
Herança e Classes Abstratas



Herança

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

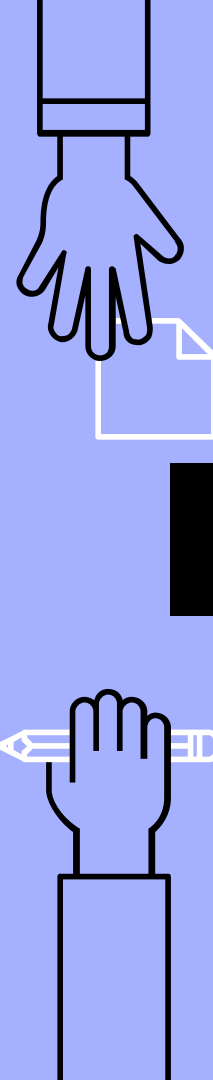
Em Java, a herança ocorre quando uma **classe passa a herdar características** (atributos e métodos) **definidas em uma outra classe**, especificada como sendo sua ancestral ou superclasse.



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**.



Herança

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:

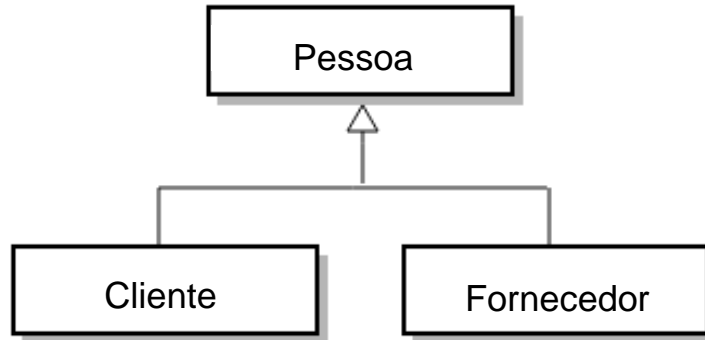
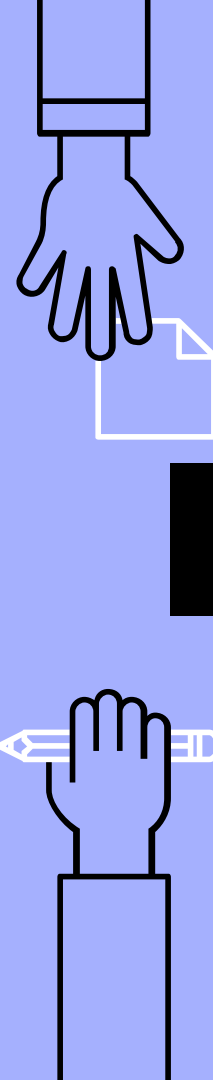
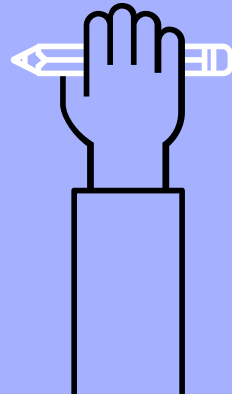
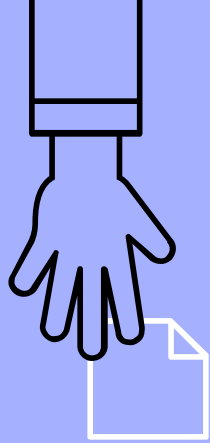
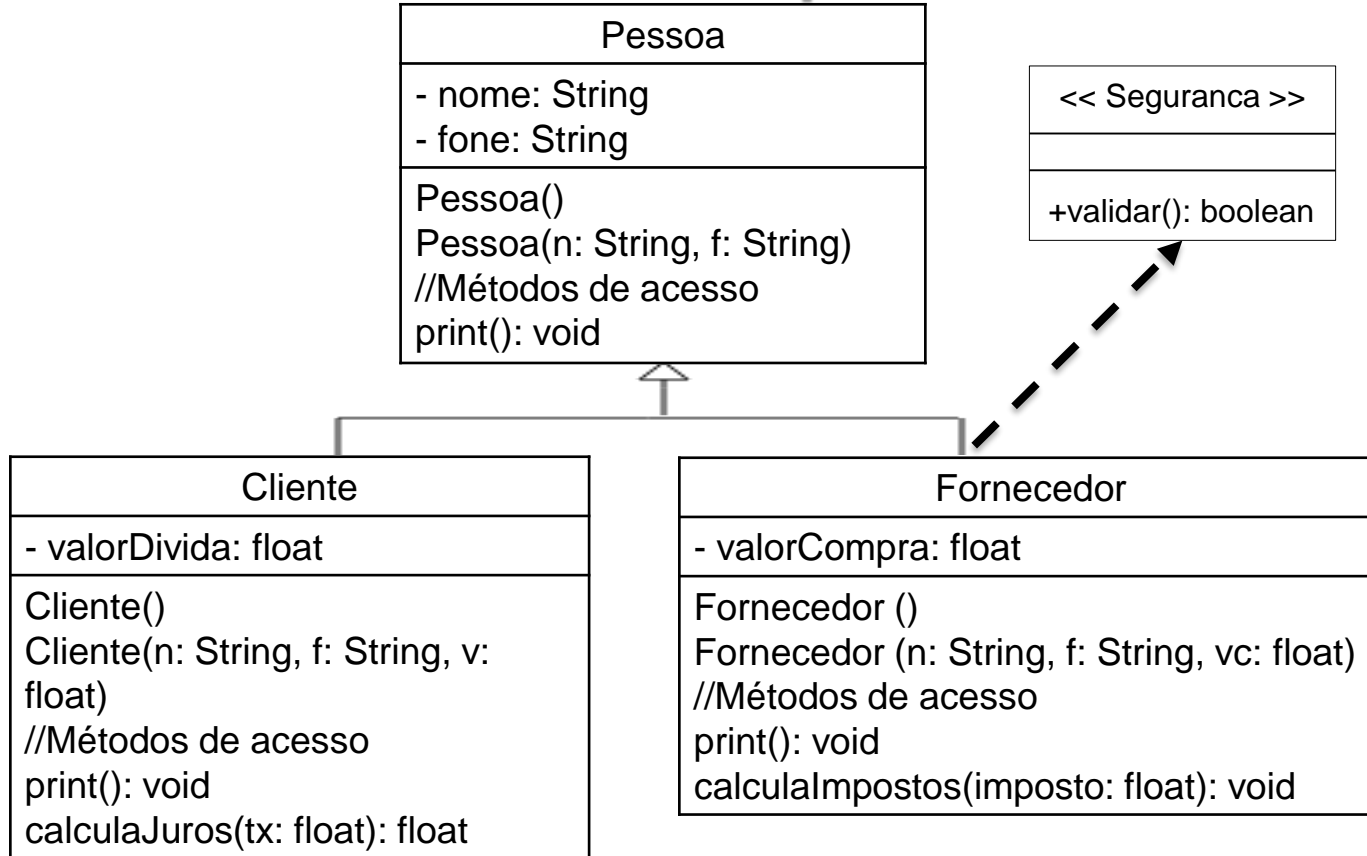


Diagrama UML simplificado (não mostra os métodos e atributos)



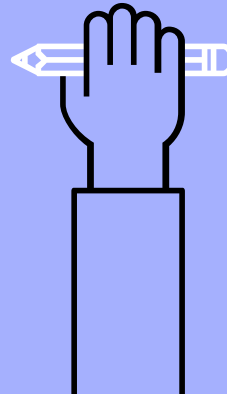
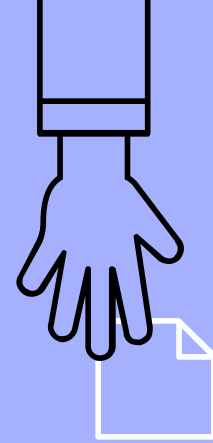
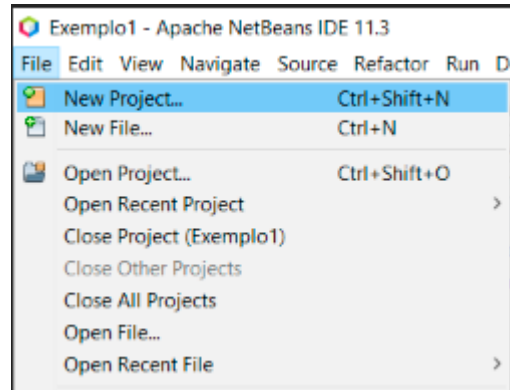
Interfaces - Exemplo



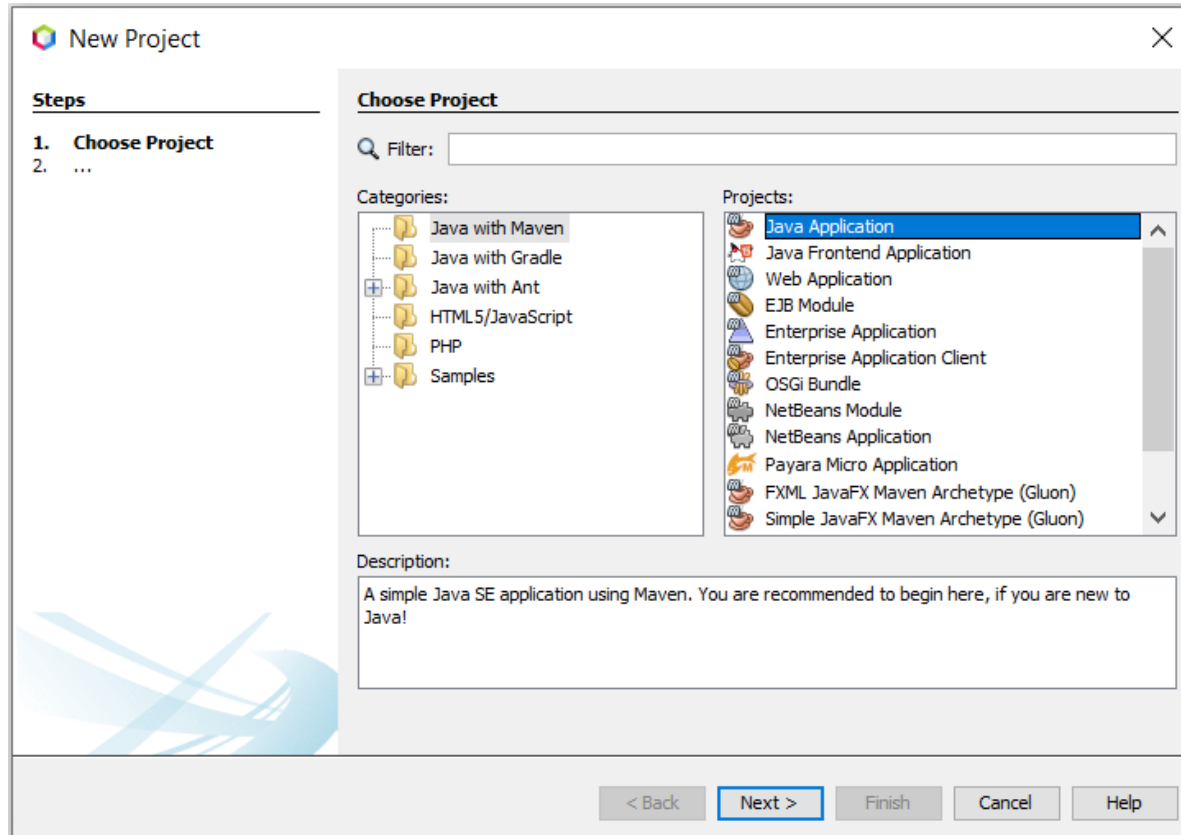
Interfaces - Exemplos

Vamos implementar usando o NetBeans.

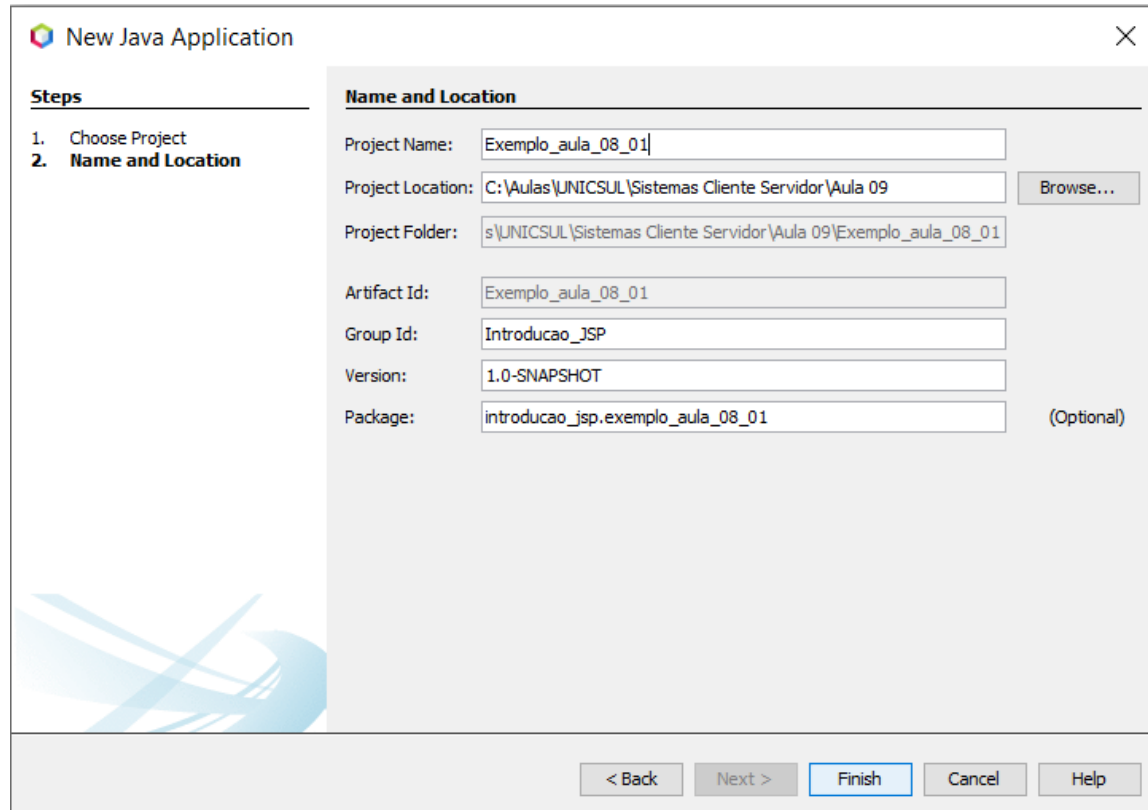
Vamos iniciar um novo projeto:



Interfaces - Exemplos



Interfaces - Exemplos



New Java Application

Steps

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

Name and Location

Project Name:

Project Location:

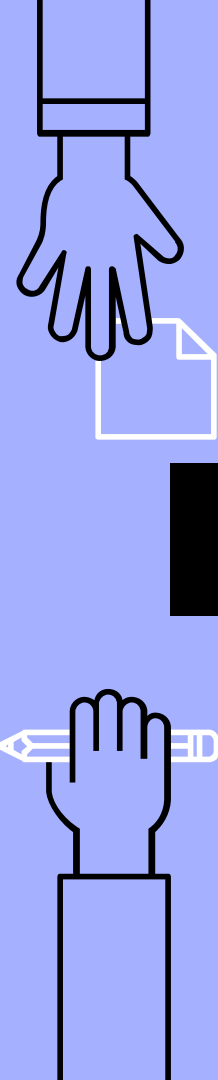
Project Folder:

Artifact Id:

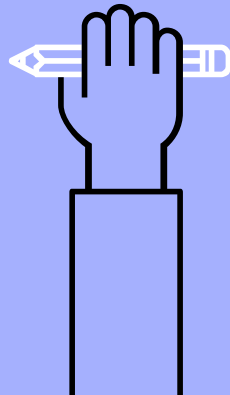
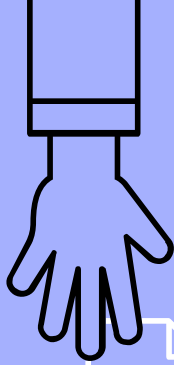
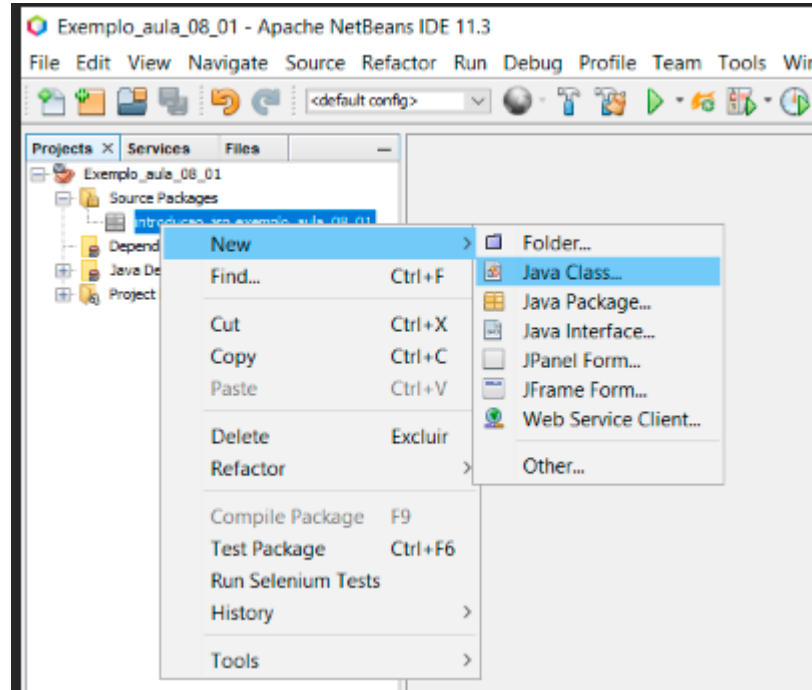
Group Id:

Version:

Package: (Optional)



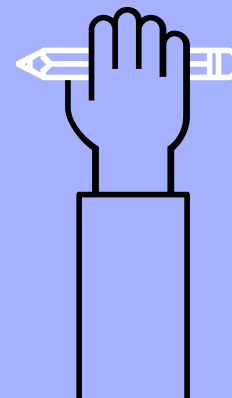
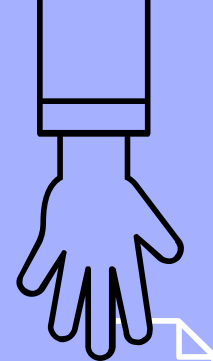
Interfaces - Exemplos



Interfaces - Exemplos

Vamos criar as seguintes classes:

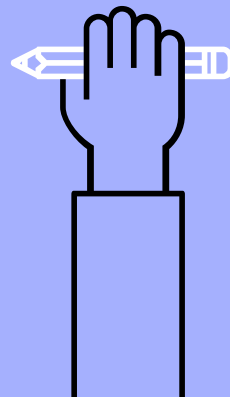
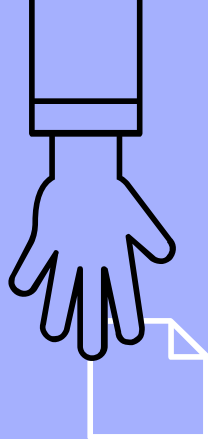
- ▶ **Pessoa**
- ▶ **Fornecedor** (extends Pessoa)
- ▶ **Cliente** (extends Pessoa)
- ▶ **TestePessoa** (public static void main(String[] args))



Pessoa.java x Fornecedor.java x TestePessoa.java x Cliente.java x

Source History

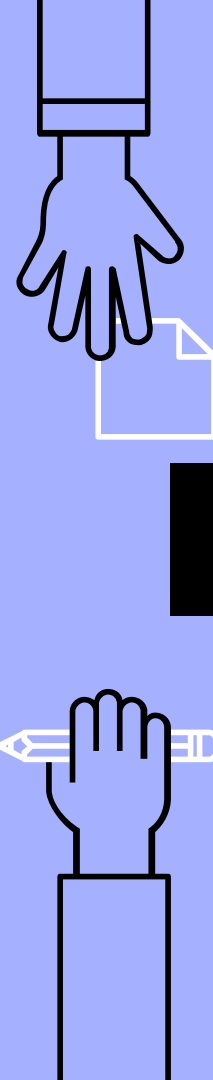
```
1  import javax.swing.JOptionPane;
2
3  @
4  public class Pessoa {
5      private String nome;
6      private String fone;
7
8      Pessoa() {}
9      Pessoa(String n, String f){
10         nome=n;
11         fone=f;
12     }
13
14     public String getNome() {
15         return nome;
16     }
17
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     @
31     public void print(){
32         JOptionPane.showMessageDialog(null,"Dados \nNome: "
33                                     + nome + "\nTelefone: "+ fone);
34     }
35 }
```



Pessoa.java x Fornecedor.java x TestePessoa.java x Cliente.java x

Source History

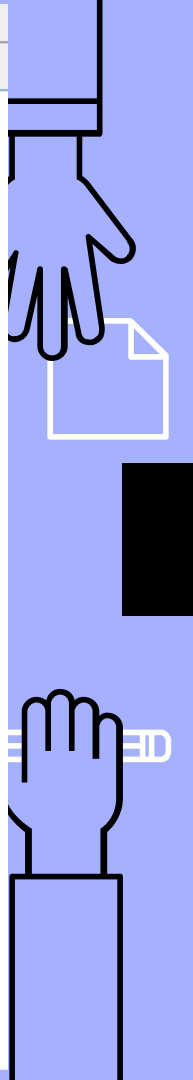
```
1  import javax.swing.JOptionPane;
2
3  public class Fornecedor extends Pessoa{
4      private float valorCompra;
5      Fornecedor() {}
6      Fornecedor(String n, String f, float vc){
7          super(n, f);
8          valorCompra=vc;
9      }
10     public float getValorCompra() {
11         return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

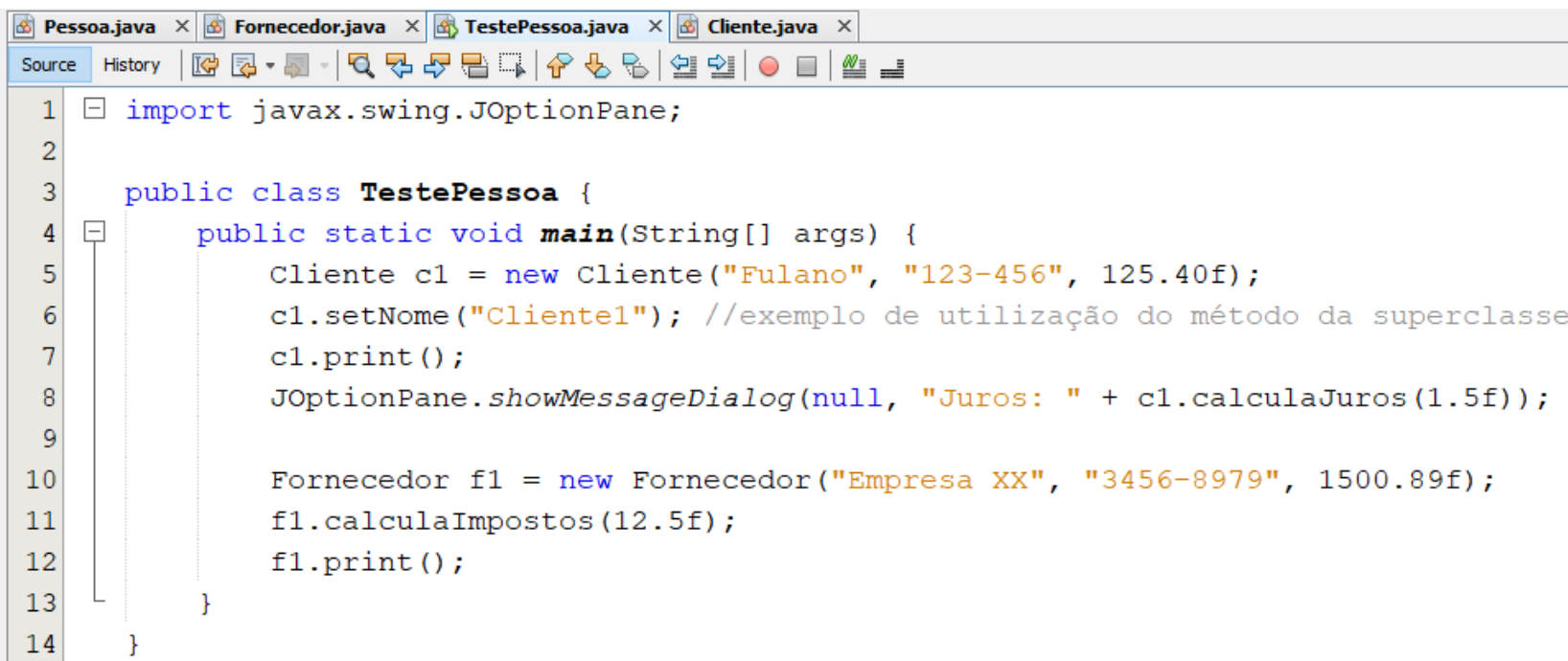


```

1  import javax.swing.JOptionPane;
2  public class Cliente extends Pessoa {
3      private float valorDivida;
4
5      Cliente() {}
6
7      Cliente(String n, String f, float vd) {
8          super(n, f);
9          valorDivida = vd;
10     }
11
12     public float getValorDivida() {
13         return valorDivida;
14     }
15
16     public void setValorDivida(float valorDivida) {
17         this.valorDivida = valorDivida;
18     }
19
20     public void print() {
21         super.print();
22         JOptionPane.showMessageDialog(null, "\nValor da Divida: " + valorDivida);
23     }
24
25     public float calculaJuros(float tx) {
26         return valorDivida * tx / 100;
27     }
28 }

```

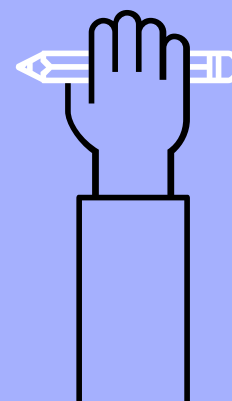
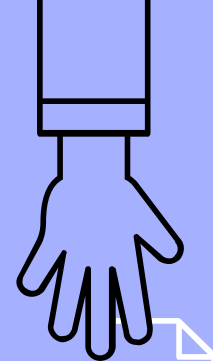




The image shows a screenshot of a Java IDE with four tabs: Pessoa.java, Fornecedor.java, TestePessoa.java, and Cliente.java. The TestePessoa.java tab is active, displaying the following code:

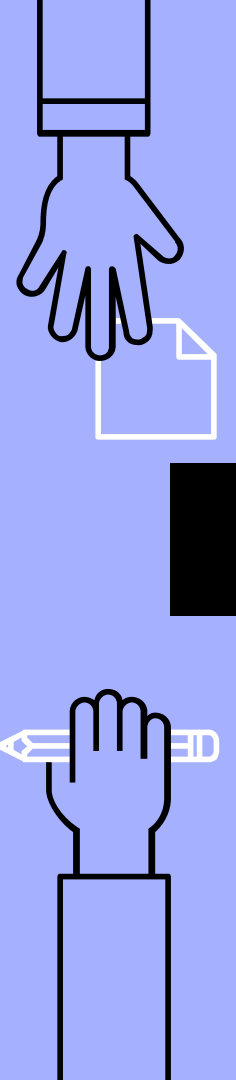
```
1 import javax.swing.JOptionPane;
2
3 public class TestePessoa {
4     public static void main(String[] args) {
5         Cliente c1 = new Cliente("Fulano", "123-456", 125.40f);
6         c1.setNome("Cliente1"); //exemplo de utilização do método da superclasse
7         c1.print();
8         JOptionPane.showMessageDialog(null, "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 }
```

The code defines a `TestePessoa` class with a `main` method. It creates a `Cliente` object `c1` and a `Fornecedor` object `f1`, then calls their respective methods to calculate interest and taxes, displaying the results using `JOptionPane.showMessageDialog`.



Interfaces - Exemplos

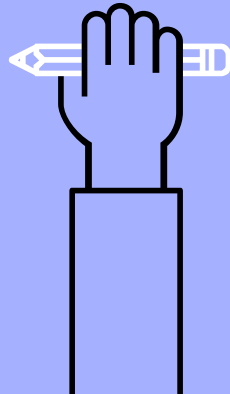
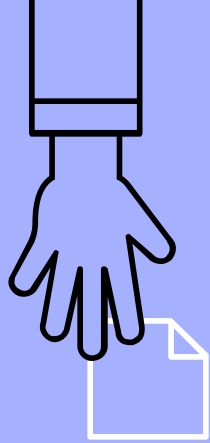
Vamos testar a aplicação!



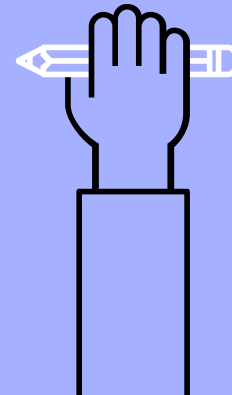
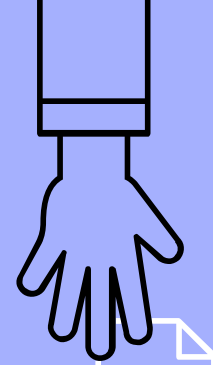
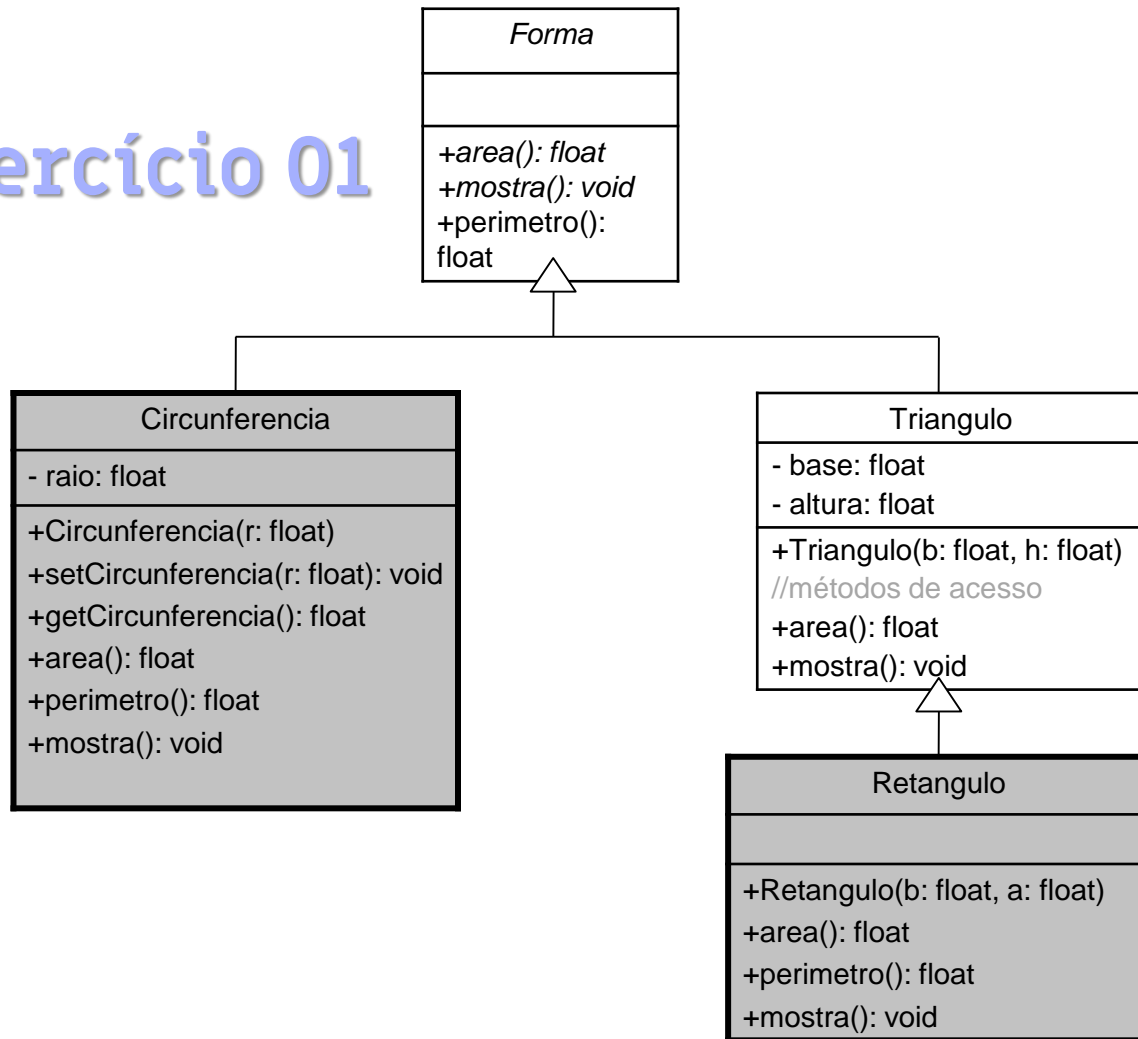
Interfaces x Classes Abstratas

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:

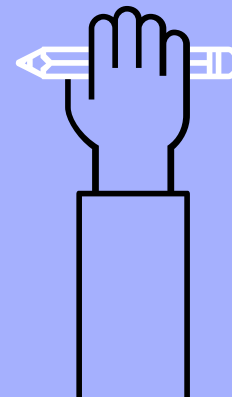
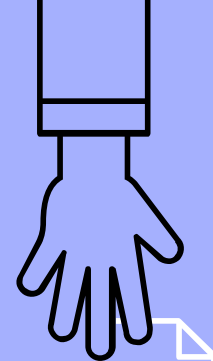
Circunferencia
- raio: float
+Circunferencia(r: float) +setCircunferencia(r: float): void +getCircunferencia(): float +area(): float +perimetro(): float +mostra(): void

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

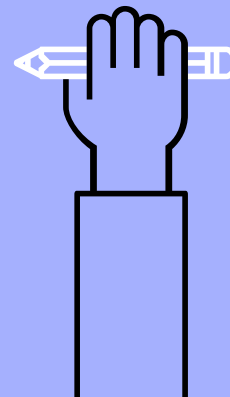
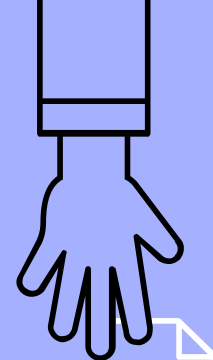
Crie a classe abaixo como subclasse de Triangulo:

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: $\text{perimetro} = (\text{base} * \text{altura}) * 2$

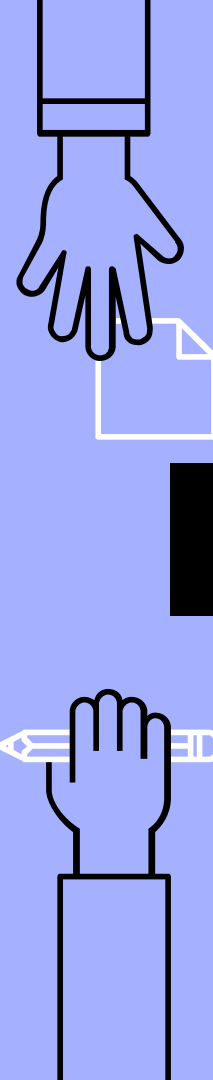
O método `mostra` deve exibir os valores de todos os atributos da classe



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.

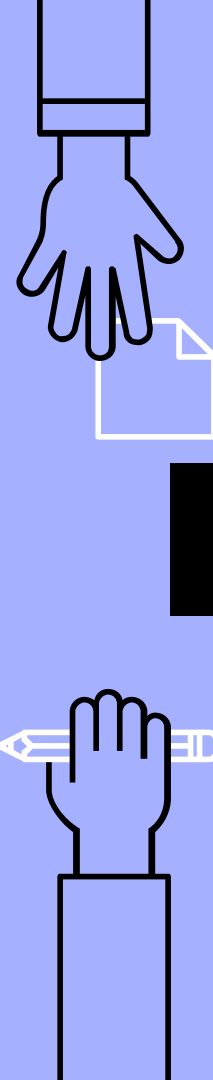
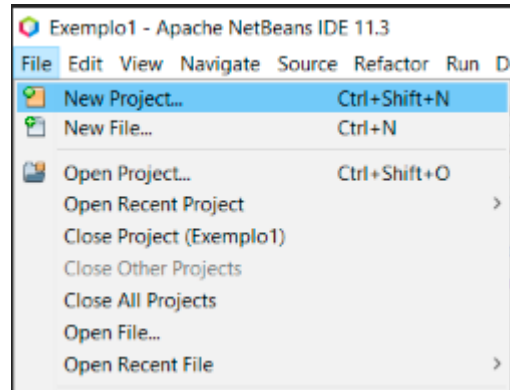
Mostre os dados de cada objeto através do método mostra().



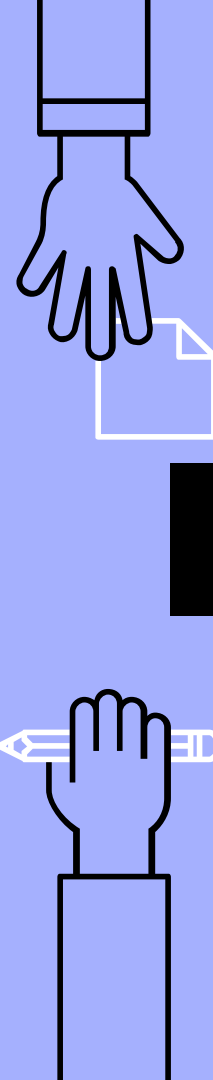
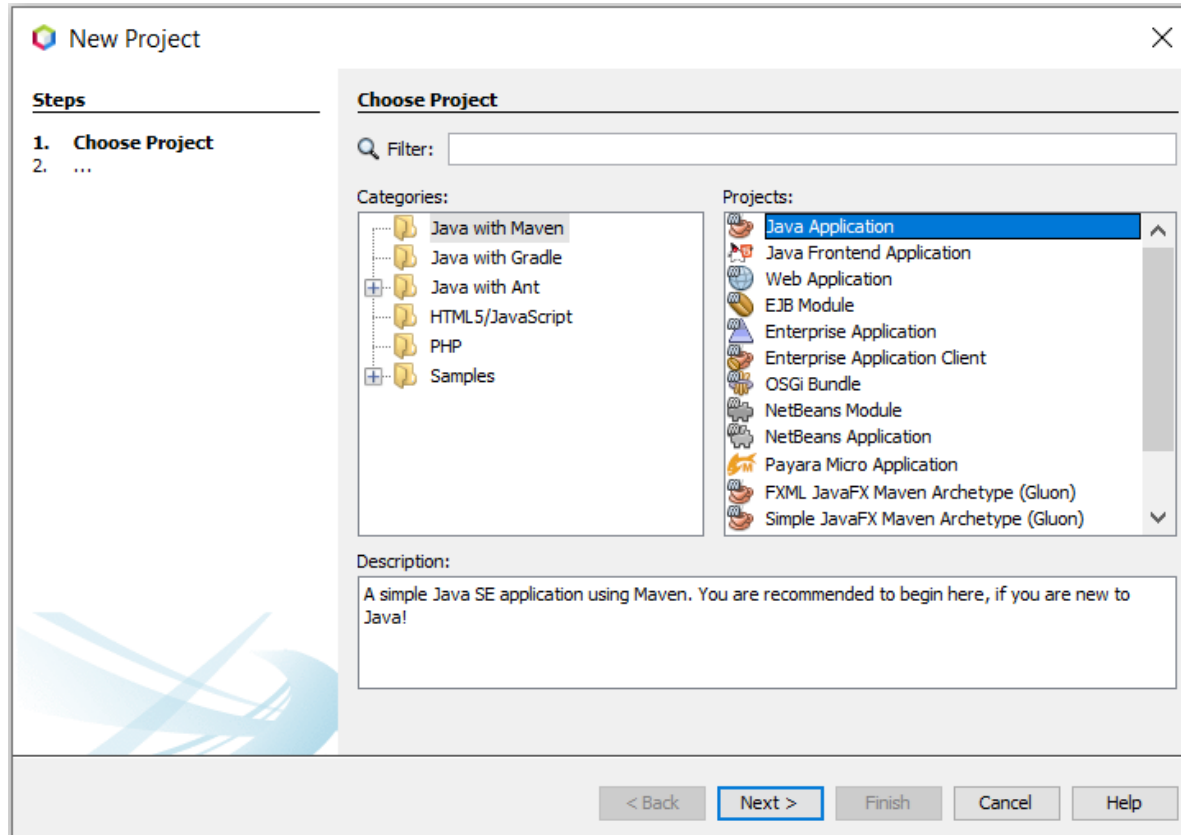
Exercício 01

Vamos implementar usando o NetBeans.

Vamos iniciar um novo projeto:



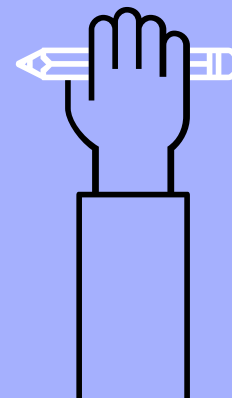
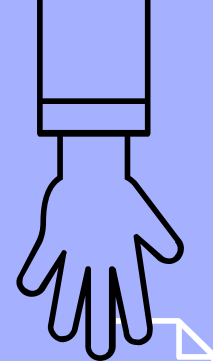
Exercício 01



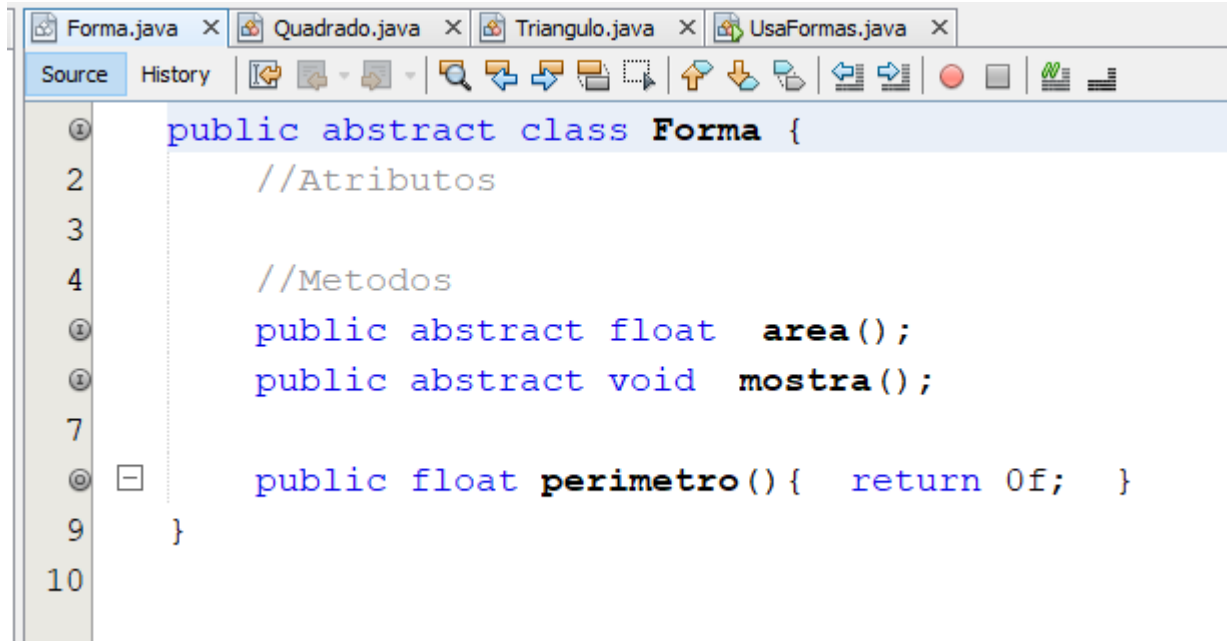
Exercício 01

Vamos criar as seguintes Classes:

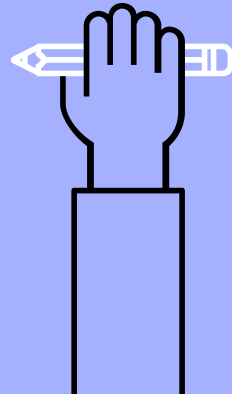
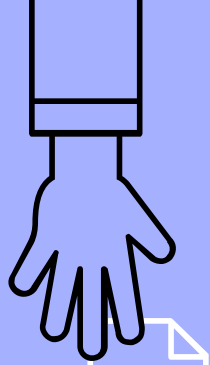
- ▶ **Forma** (Classe Abstrata)
- ▶ **Quadrado** (extends Forma)
- ▶ **Triangulo** (extends Forma)
- ▶ **UsaFormas** (public static void main(String[] args))



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
```



Forma.java x Quadrado.java x Triangulo.java x UsaFormas.java x

Source History

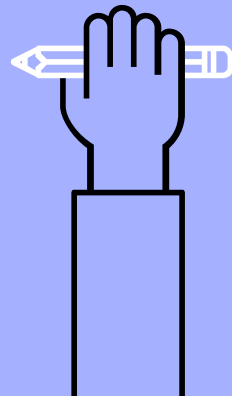
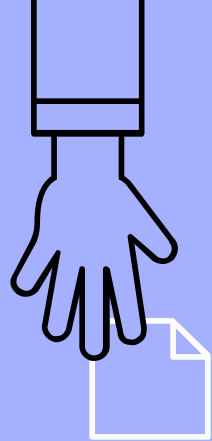
```
1 public class Quadrado extends Forma {
2     //Atributos
3     private float base;
4
5     //Construtor
6     public Quadrado(float b){ base = b; }
7
8     //Metodos de acesso
9     public float getBase() { return base; }
10    public void setBase(float b) { base = b; }
11
12    //sobreposição do 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 }
```

Forma.java x Quadrado.java x Triangulo.java x UsaFormas.java x

Source History

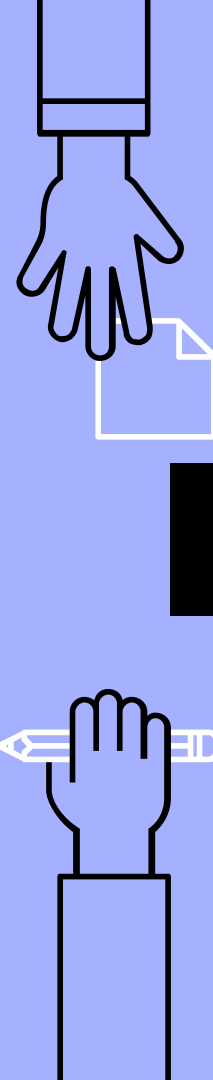
```
1 public class Triangulo extends Forma {
2     //Atributos
3     private float base, altura;
4
5     //Construtor
6     public Triangulo(float b, float h){
7         base = b;
8         altura = h;
9     }
10
11     //Metodos de acesso
12     public float getBase() { return base; }
13     public 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 }
```



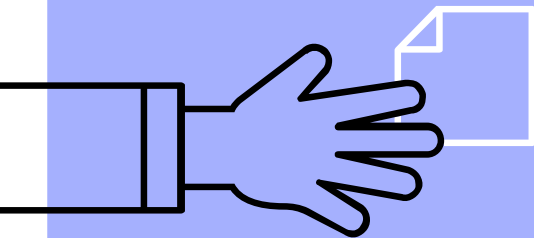
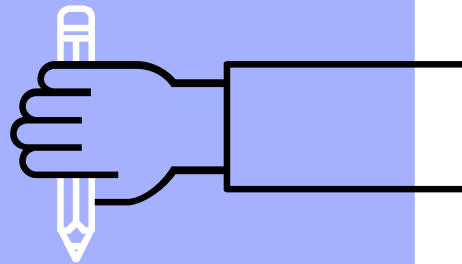
Exercício 01

Vamos testar nossa aplicação!



8. Unidade

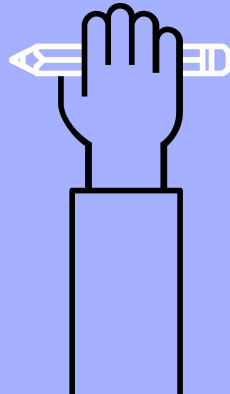
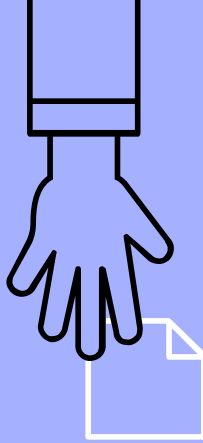
Estrutura de Repetição



Estrutura de Repetição

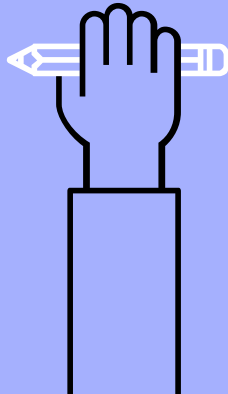
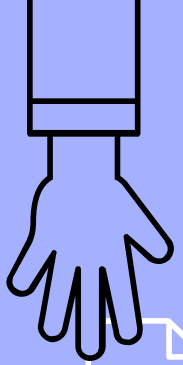
Quando tivermos que repetir um trecho de um programa por determinado número de vezes, precisaremos do auxílio de uma estrutura de repetição.

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

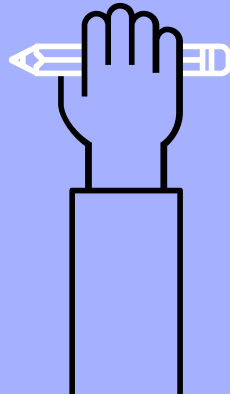
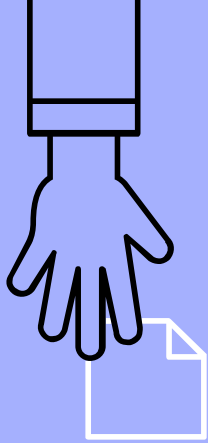
Estrutura de repetição while

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));  
    }  
}
```

Variável
Contador

Condição de execução

Variável
Acumulador

Estrutura de Repetição

Estrutura de repetição do-while

Comando da linguagem Java.

{ iniciar a variável de controle }

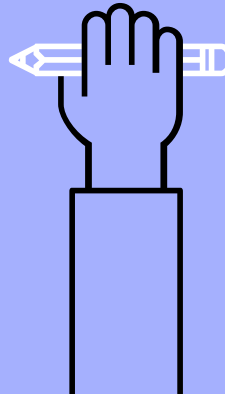
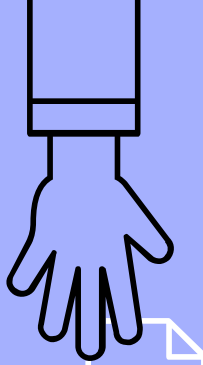
do{

{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



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

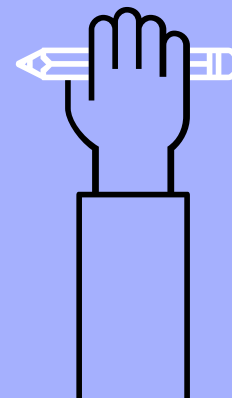
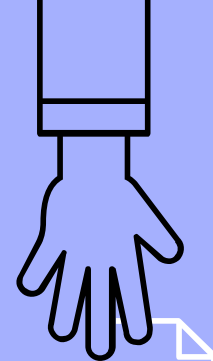
Estrutura de Repetição

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++) {
```

Condição de execução

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

Variável Contador

```
            soma += A;
```

Variável Acumulador

```
        }
```

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

Lê o que foi digitado como valor inteiro

```
    }
```

```
}
```

Estrutura de Repetição

Estrutura de repetição for

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

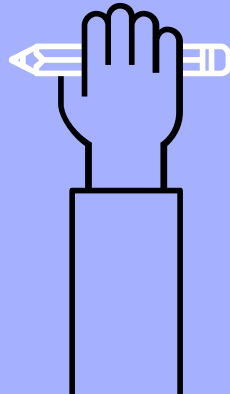
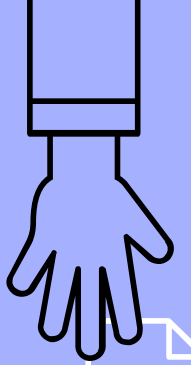
O laço contado **for** funciona da seguinte maneira:

- ▶ 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

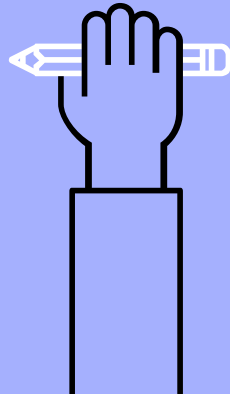
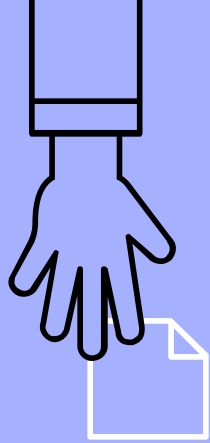
Obs: as chaves { } são necessárias para uma estrutura de bloco (quando desejamos repetir mais de um comando).
Se formos repetir um comando simples poderíamos omitir estas chaves.



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 Exercicio_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: ")

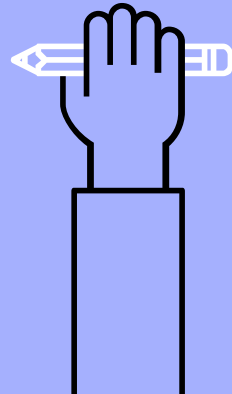
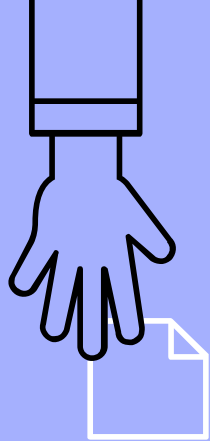
 ler (valor)

 soma <- soma + valor

fim para

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

fim

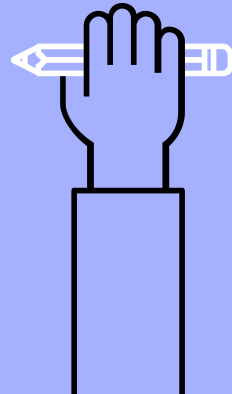
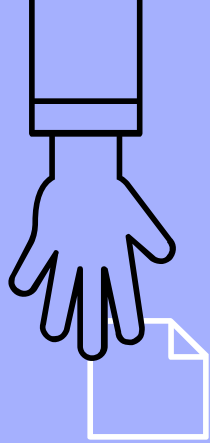


```
ExemploFor.java X
Source History
1 import javax.swing.JOptionPane;
2
3 public class ExemploFor {
4
5     public static void main(String arg[]){
6         int i, n;
7         double valor, soma;
8
9         n = Integer.parseInt(JOptionPane.showInputDialog(null,
10             "Digite a quantidade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

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 Exercio_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)

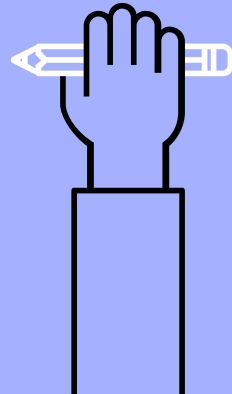
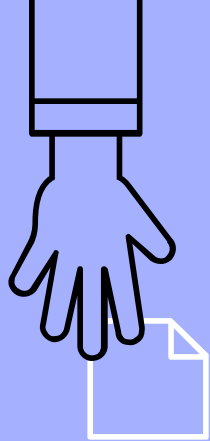
soma <- soma + valor

i <- 1 + 1

fim para

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

fim

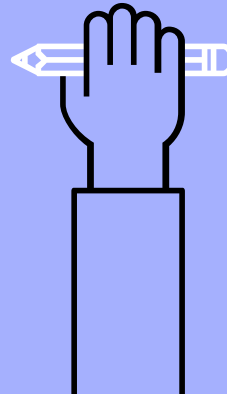
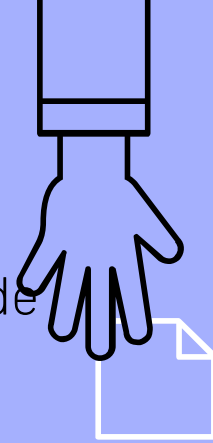


```
1  import javax.swing.JOptionPane;
2
3  public class ExemploFor {
4
5      public static void main(String arg[]){
6          int i, n;
7          double valor, soma;
8
9          n = Integer.parseInt(JOptionPane.showInputDialog(null,
10              "Digite a quantidade de valores que serão processados:",
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:

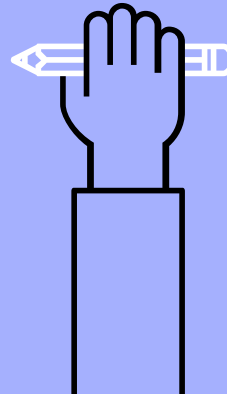
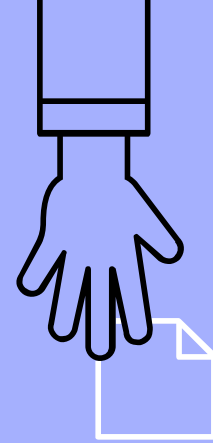
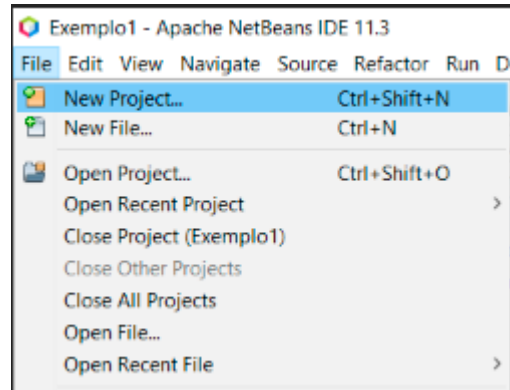
Tabuada
<ul style="list-style-type: none">- numero: int- operador: char
<ul style="list-style-type: none">+ Tabuada(numero:int, operador: char)+ getNumero(): int+ getOperador(): char+ setNumero(numero: int): void+ setOperador(operador: char): void+ geraTabuada(): String



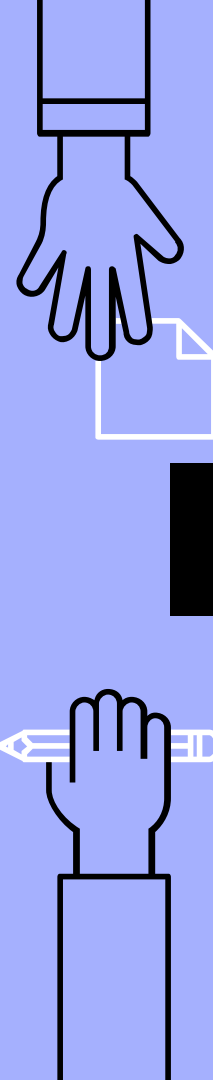
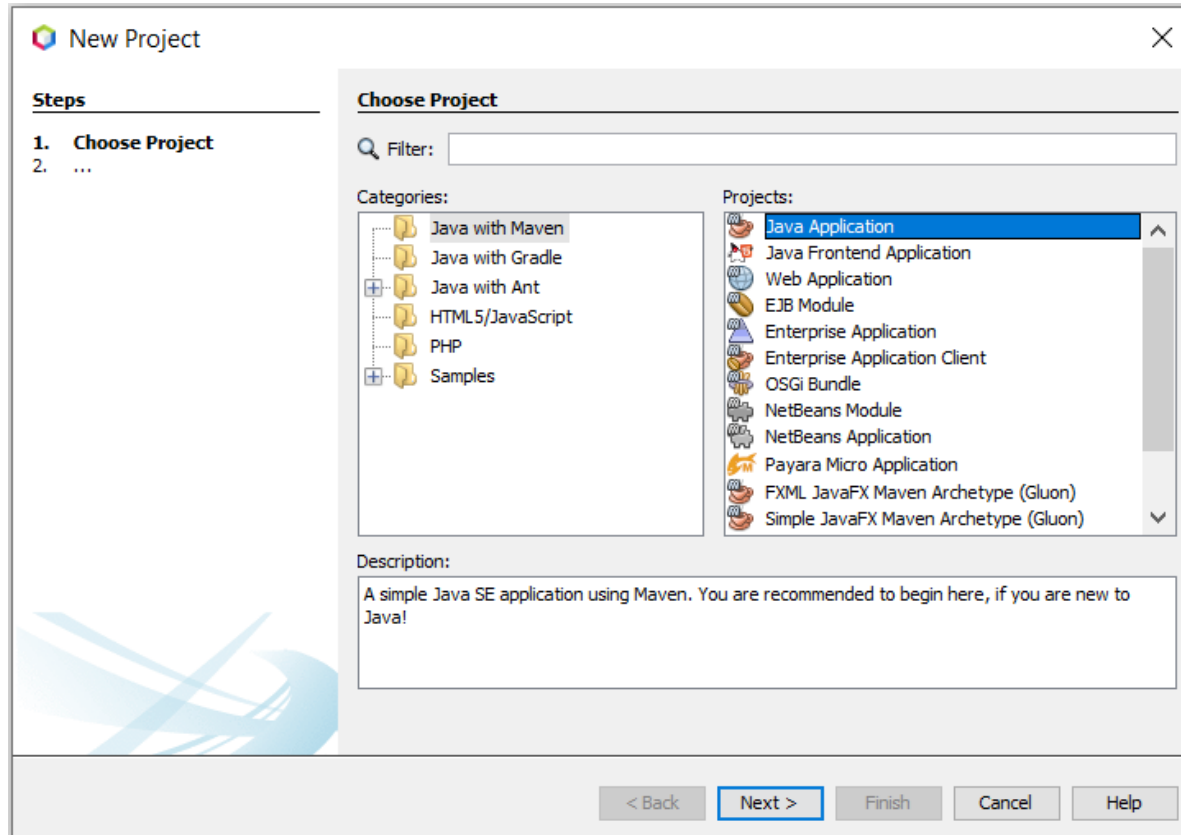
Estrutura de Repetição

Vamos implementar usando o NetBeans.

Vamos iniciar um novo projeto:



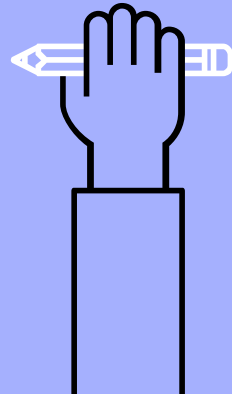
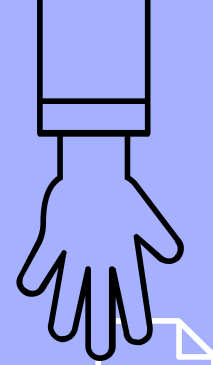
Estrutura de Repetição



Estrutura de Repetição

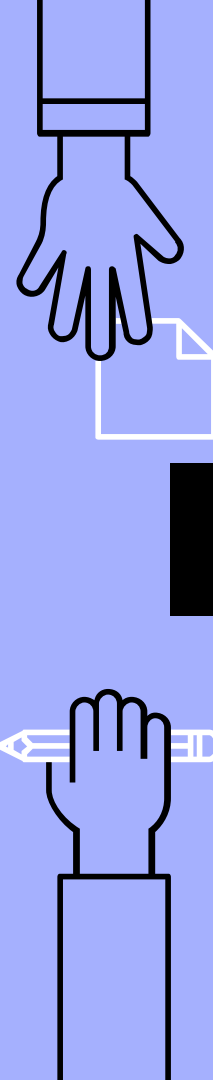
Vamos criar as seguintes Classes:

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




```
Tabuada.java x UsaTabuada.java x
Source History

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 }
```

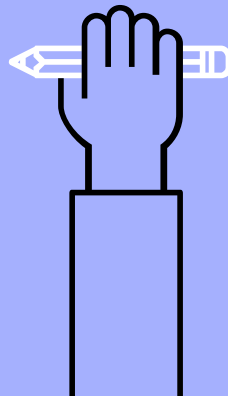
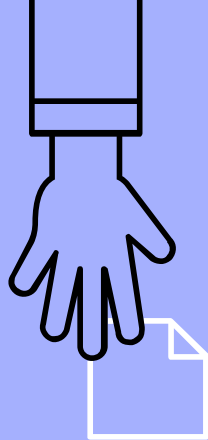


Tabuada.java x UsaTabuada.java x

Source History

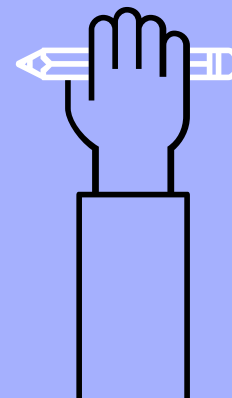
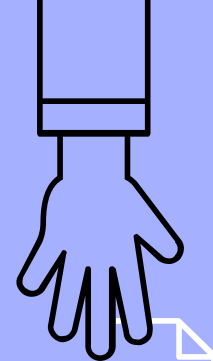


```
1
2 import java.util.Scanner;
3
4 public class UsaTabuada {
5
6     public static void main(String[] args) {
7         Scanner leitura = new Scanner(System.in);
8         int num;
9         char op;
10        String resp = "";
11
12        do{
13            System.out.println("Informe o número da 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
```



Vetores ou Arrays

- ▶ Definição
 - Conjunto de variáveis de mesmo tipo de dado.
- ▶ Exemplos:
 - Média de alunos;
 - Altura de atletas;
 - Idades de eleitores, etc.

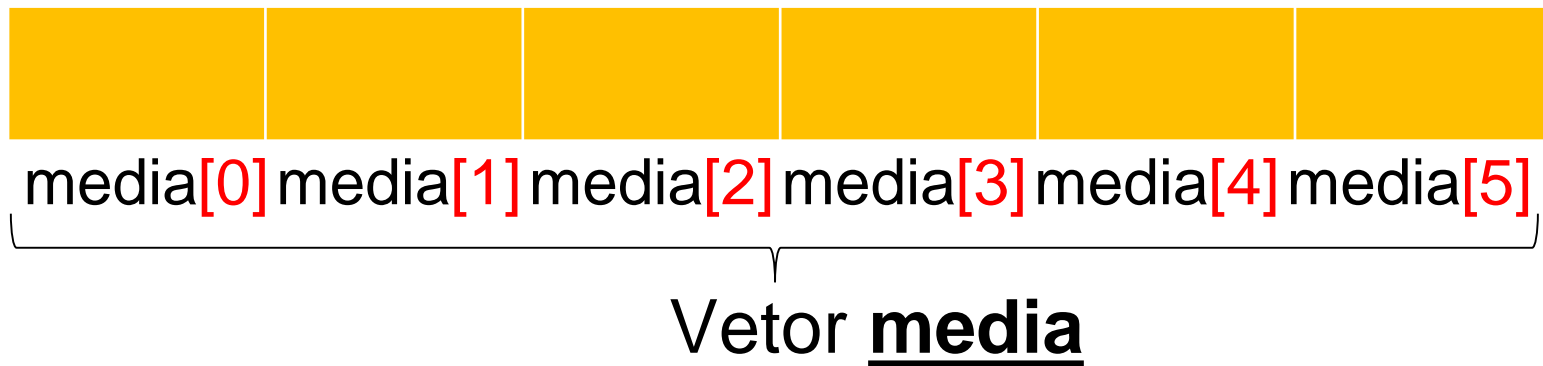


Vetores ou Arrays

- Declaração de um vetor:

Java:

```
double media [ ] = new double[6];
```



Vetores ou Arrays

- Inicialização de um vetor:

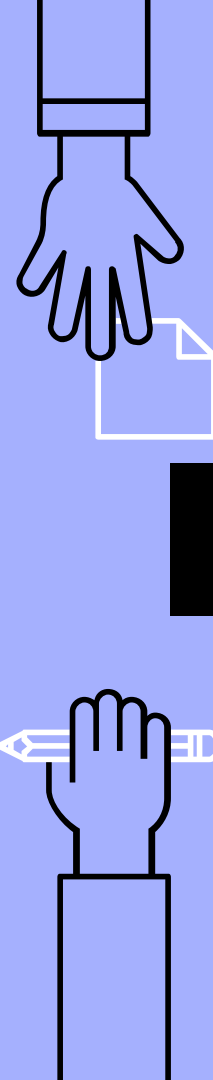
Java:

```
double [ ] media = {2.2, 1.8, 8.5, 0.0, 5.3, 7.9};
```



media[0] media[1] media[2] media[3] media[4] media[5]

Vetor media



Vetores ou Arrays

- Inicialização de um vetor:

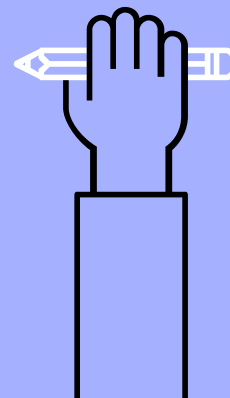
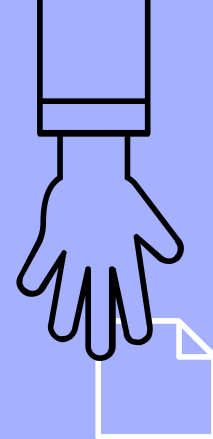
Java:

```
double [ ] media = {2.2, 1.8, 8.5, 0.0, 5.3, 7.9};
```

2.2	1.8	8.5	0.0	5.3	7.9
------------	-----	-----	-----	-----	-----

media[0] media[1] media[2] media[3] media[4] media[5]

Vetor media

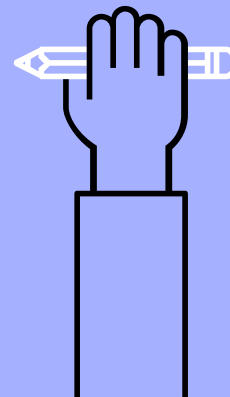
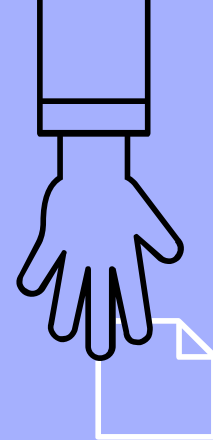
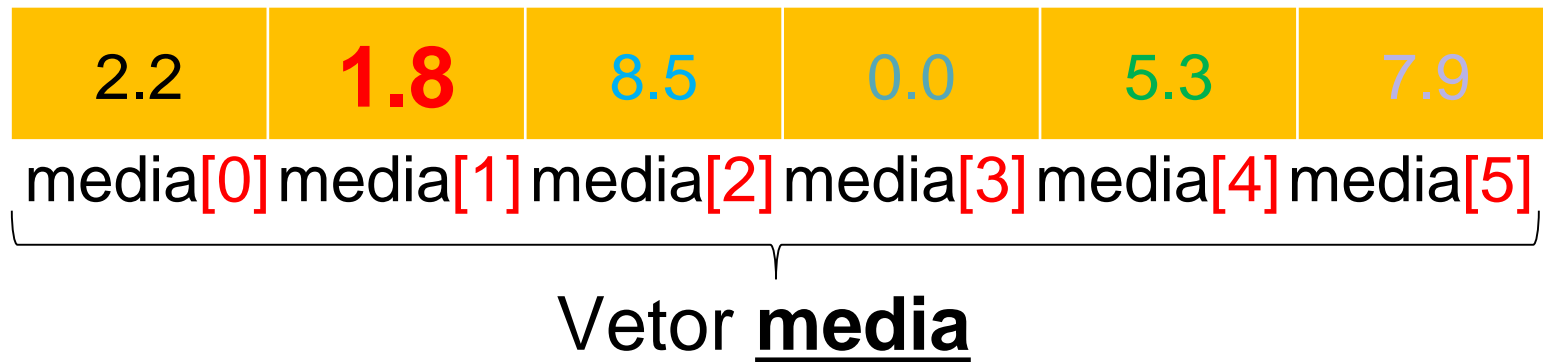


Vetores ou Arrays

- Inicialização de um vetor:

Java:

```
double [ ] media = {2.2, 1.8, 8.5, 0.0, 5.3, 7.9};
```

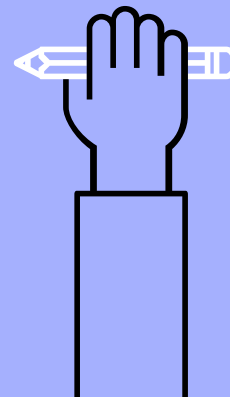
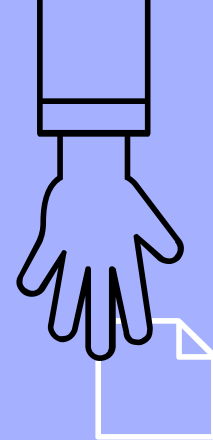
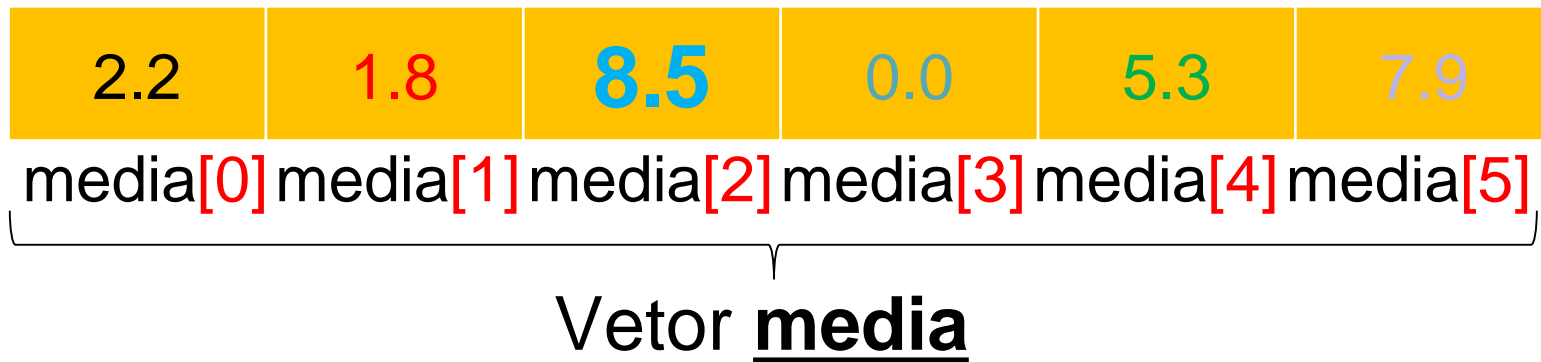


Vetores ou Arrays

- Inicialização de um vetor:

Java:

```
double [ ] media = {2.2, 1.8, 8.5, 0.0, 5.3, 7.9};
```

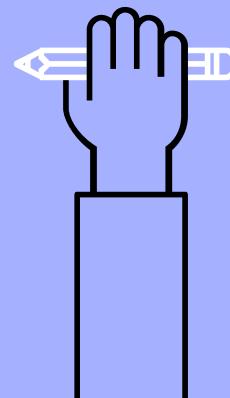
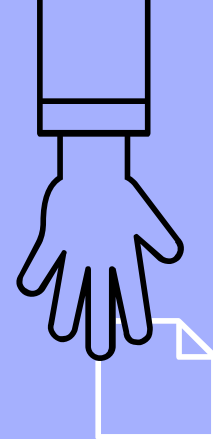
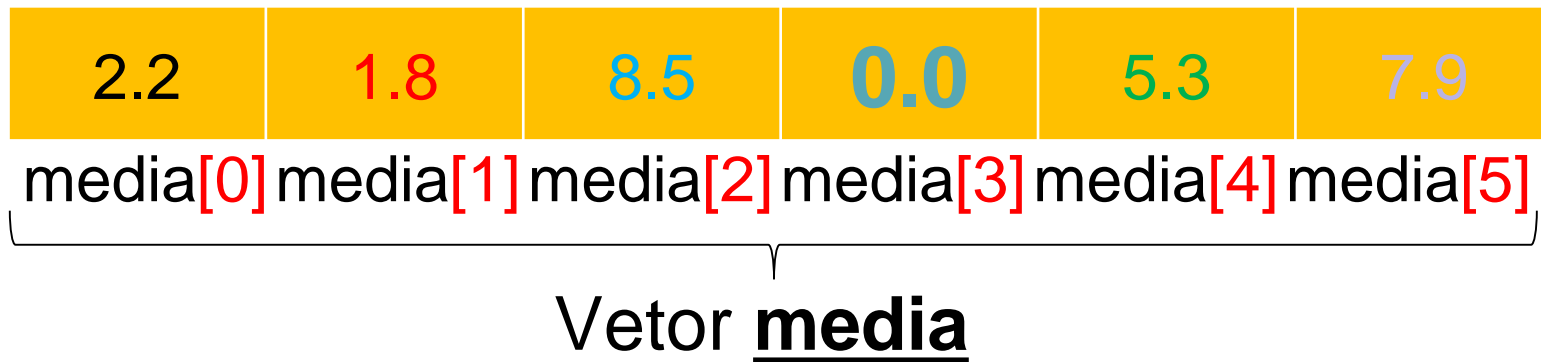


Vetores ou Arrays

- Inicialização de um vetor:

Java:

```
double [ ] media = {2.2, 1.8, 8.5, 0.0, 5.3, 7.9};
```

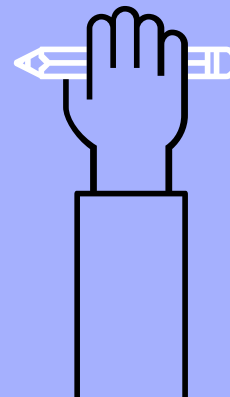
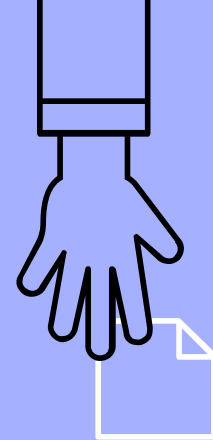
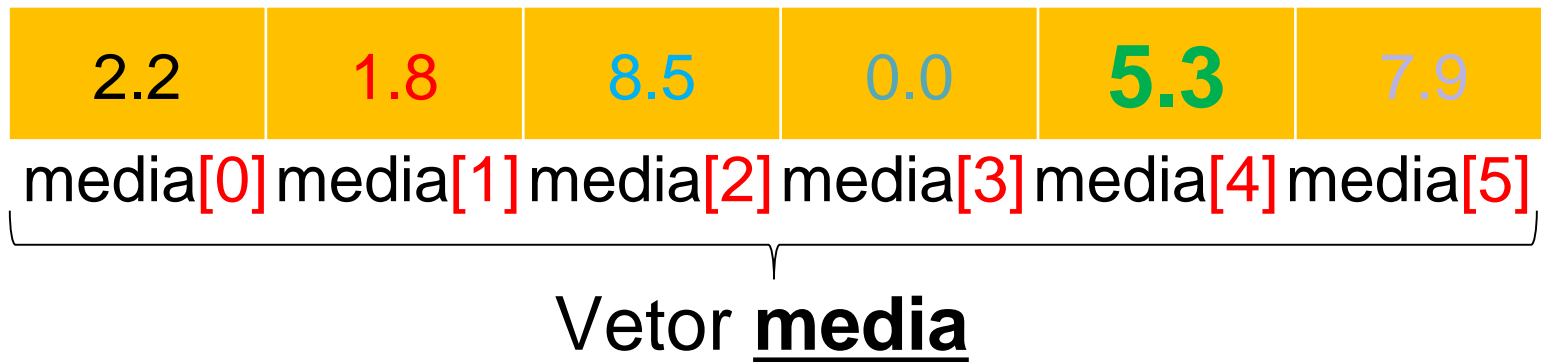


Vetores ou Arrays

- Inicialização de um vetor:

Java:

```
double [ ] media = {2.2, 1.8, 8.5, 0.0, 5.3, 7.9};
```

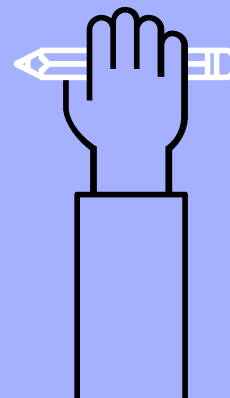
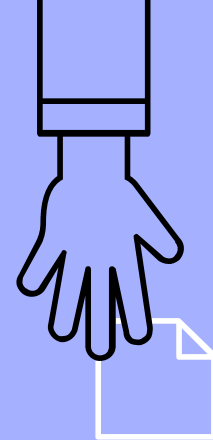
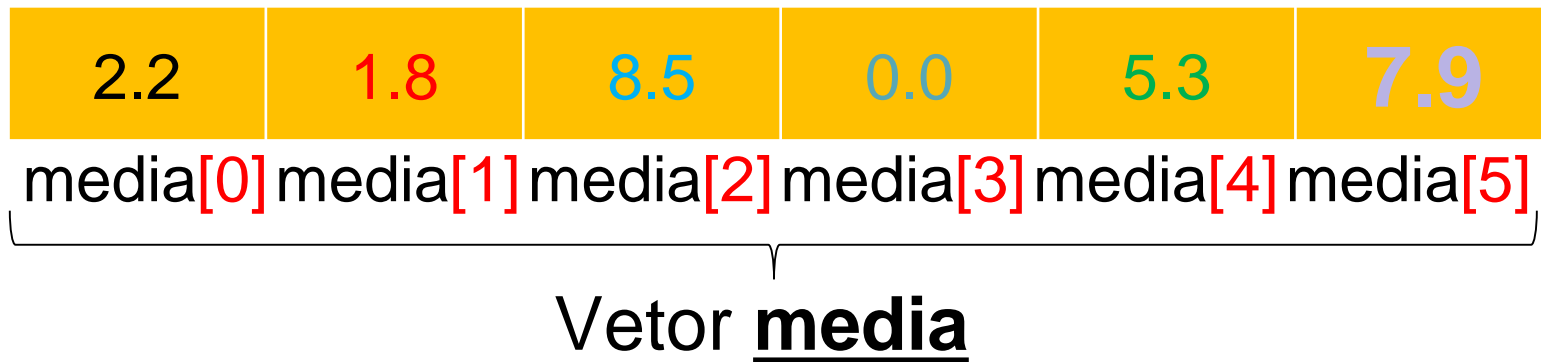


Vetores ou Arrays

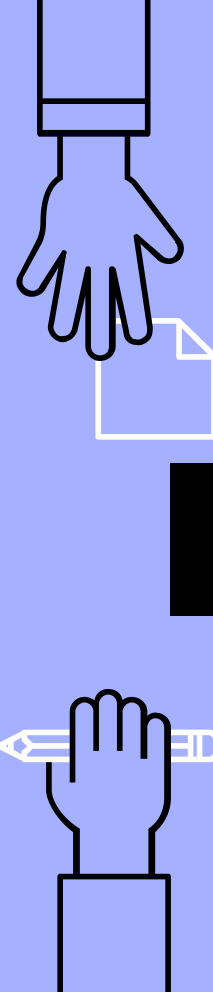
- Inicialização de um vetor:

Java:

```
double [ ] media = {2.2, 1.8, 8.5, 0.0, 5.3, 7.9};
```



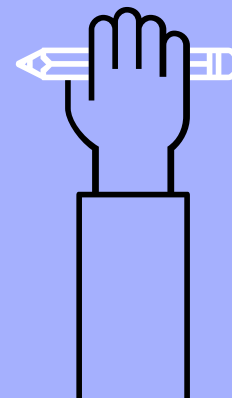
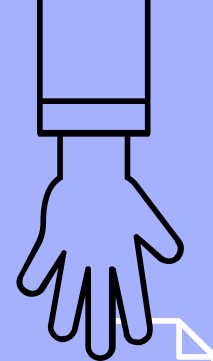
```
ExemploVetor.java x
Source History
1 package introducao_jsp.exemplo_vetor;
2
3 public class ExemploVetor {
4     public static void main(String[] args){
5         final int TAMANHO = 30;
6         int vetorNumeros[] = new int[TAMANHO];
7
8         for(int indice = 0; indice < TAMANHO; indice++){
9             vetorNumeros[indice] = indice + 10;
10
11             if (vetorNumeros[indice]%2 ==0){
12                 switch(vetorNumeros[indice]){
13                     case 20:
14                         System.out.print("Vinte");
15                         break;
16                     case 30:
17                         System.out.print("Trinta");
18                         break;
19                     default:
20                         System.out.print(vetorNumeros[indice] + " ");
21                         break;
22                 }
23             }
24         }
25     }
26 }
```



10 12 14 16 18 vinte 22 24 26 28 trinta 32 34 36 38

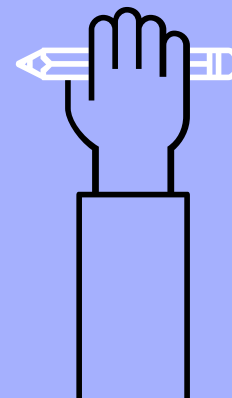
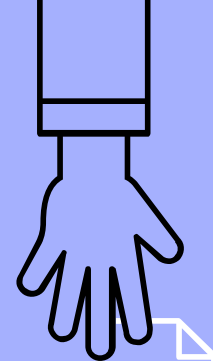
Vetores ou Arrays

- ▶ ArrayList
 - Pertence à classe *java.util.ArrayList*.



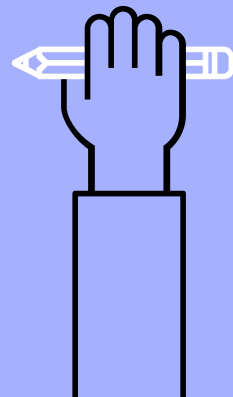
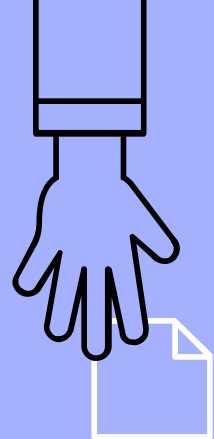
Vetores ou Arrays

- ▶ ArrayList
 - Pertence à classe `java.util.ArrayList`.
 - O *array* pode ter tamanho variável.



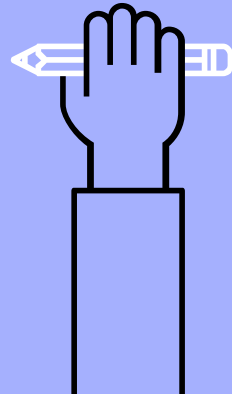
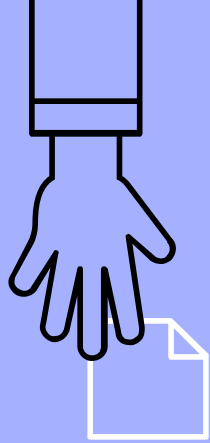
Vetores ou Arrays

- ▶ ArrayList
 - Pertence à classe *java.util.ArrayList*.
 - O array pode ter tamanho variável.
 - Possui métodos muito úteis:
 - *add(valor OU objeto); add(índice, valor OU objeto); size(); get(índice); remove(índice); set(índice, valor OU objeto); clear()*, dentre outros.



Exercício

Desenvolva um programa na linguagem java, que receba a média da temperatura diária durante um período de 7 dias (armazene as informações em um vetor), calcule a média da temperatura desse período (semanal) e informe quantos dias a temperatura ficou acima da média e quantos ficou abaixo da média semanal.



“

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



Winston Churchill

Obrigado!

Se precisar ...

Prof. Claudio Benossi

cbenossi@cruzeirodosul.edu.br

