

# POO

## Programação Orientada a Objetos

Material 003

Professor Maromo



# Agenda

JAVA - RECURSOS



Recursos Básicos da linguagem Java

TIPOS



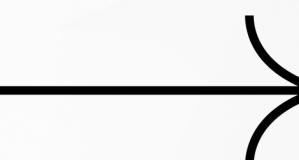
Tipos primitivos, referência, atributos, atributos estáticos

OPERADORES



Aritméticos, unários, lógicos, relacionais e atribuição

ESTRUTURAS



Controle, repetição, break, continue, i/o



# Variáveis

- Representa um endereço de memória.
- Quando definimos um tipos de variável consideramos o seu nome, o tipo e um valor de inicialização.
- É definida dentro de um método, enquanto um atributo é definido dentro de uma Classe.
- Uma variável definida internamente a um método só será visível dentro do método.

# Atributos



- Atributo deve ser criado após a definição da classe, fora de qualquer método. [Campos].
- Podem ser:
  - Estáticos (enquanto a classe estiver carregada na JVM).
  - e não estáticos (enquanto o objeto estiver ativo)



# Tipos de Dados Primitivos

<b>Tipo</b>	<b>Tamanho (bits)</b>	<b>Intervalo/Descrição</b>
<b>`byte`</b>	8	-128 a 127
<b>`short`</b>	16	-32,768 a 32,767
<b>`int`</b>	32	$-2^{31}$ a $2^{31} - 1$
<b>`long`</b>	64	$-2^{63}$ a $2^{63} - 1$
<b>`float`</b>	32	Números com casas decimais (IEEE 754)
<b>`double`</b>	64	Números com casas decimais (mais precisão)
<b>`char`</b>	16	Caracteres Unicode (\u0000 a \uffff)
<b>`boolean`</b>	Não especificado	<b>`true` ou `false`</b>



# Tipos de Dados Referência

No Java, além dos tipos primitivos, temos os tipos de referência. Esses tipos **não armazenam o valor real da variável diretamente, mas sim uma referência** (ou, em termos mais simples, um "endereço") para o local na memória onde o valor real está armazenado. Isso é um contraste direto com os tipos primitivos, que armazenam o valor real diretamente na variável.



# Tipos de Dados Referência

Existem três categorias principais de tipos de referência em Java:

## 1. Classes (incluindo classes wrapper e Strings)

- Exemplos: **Integer**, **Character**, **String**, **Object**, e qualquer classe definida pelo usuário.

## 2. Interfaces

- Uma interface é um tipo de referência que é uma coleção de métodos abstratos (e, a partir do Java 8, métodos padrão e estáticos).

## 3. Arrays

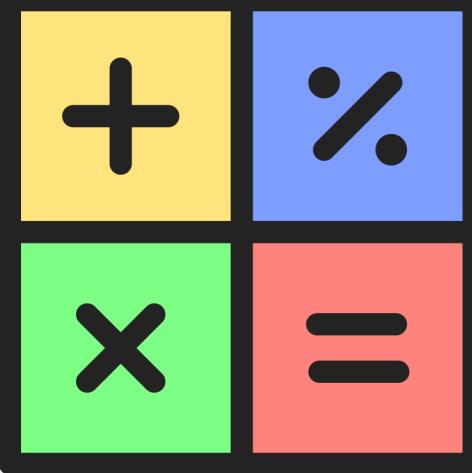
- Um array é uma estrutura de dados que armazena uma coleção fixa de elementos do mesmo tipo, seja ele primitivo ou de referência.

## Características dos tipos de referência:

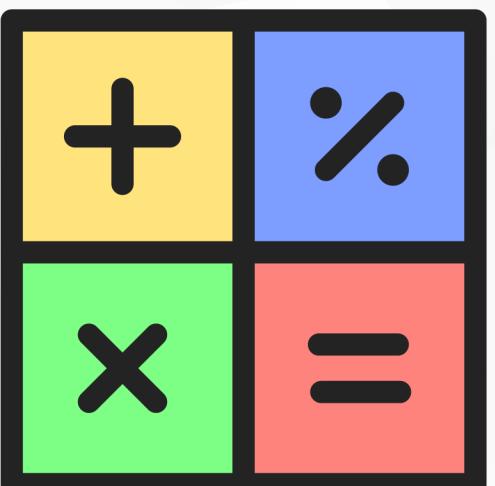
- **Nullabilidade:** Variáveis de tipos de referência podem ser atribuídas ao valor null, indicando que elas não se referem a nenhum objeto.
- **Métodos e Atributos:** Tipos de referência, sendo geralmente objetos, podem ter métodos e atributos associados a eles.
- **Alocação Dinâmica:** Objetos de tipos de referência são criados dinamicamente usando a palavra-chave new. Eles são alocados na memória heap e, quando não são mais necessários, são coletados pelo garbage collector.
- **Passagem por Referência:** Em Java, quando passamos um objeto como argumento para um método, estamos passando a referência do objeto, e não uma cópia do objeto em si. Isso significa que o método pode alterar o estado do objeto.
- **Herança:** Tipos de referência, especialmente classes, suportam o conceito de herança, onde uma classe pode herdar campos e métodos de outra classe.



# Operadores Unários e Ariméticos



Função	Sinal
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto da divisão	%
<b>Operadores Unários</b>	
Incremento	++
Decremento	--

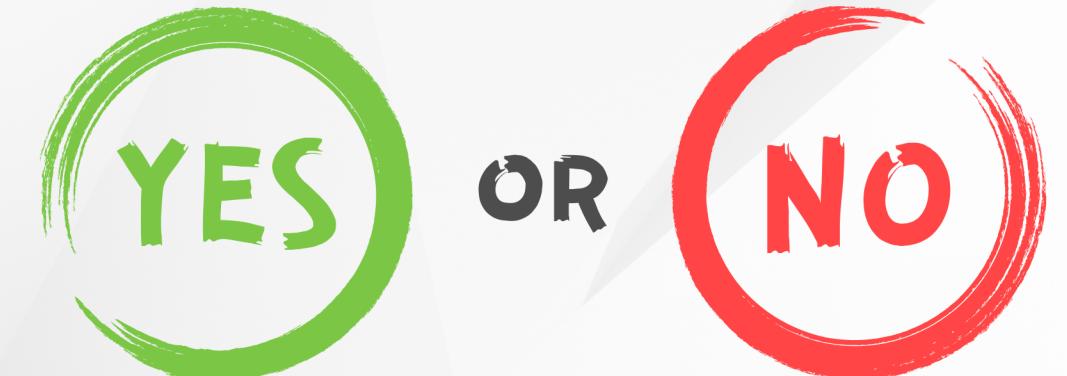


# Operadores Relacionais



Operador	Descrição	Exemplo	Resultado
<code>'=='</code>	Igual a	<code>'5 == 3'</code>	<code>'false'</code>
<code>'!='</code>	Diferente de	<code>'5 != 3'</code>	<code>'true'</code>
<code>'&gt;'</code>	Maior que	<code>'5 &gt; 3'</code>	<code>'true'</code>
<code>'&lt;'</code>	Menor que	<code>'5 &lt; 3'</code>	<code>'false'</code>
<code>'&gt;='</code>	Maior que ou igual a	<code>'5 &gt;= 5'</code>	<code>'true'</code>
<code>'&lt;='</code>	Menor que ou igual a	<code>'5 &lt;= 3'</code>	<code>'false'</code>

# Operadores Lógicos



Função	Sinal
E	&&
OU	
Não	!

# Tabela de Conversão de Tipos

Supondo a variável x	Converter em	y recebe o valor convertido
<b>Entre tipos numéricos</b>		
int x = 10	float	float y = (float) x
int x = 10	double	double y = (double) x
float x = 10.5	int	int y = (int) x
<b>De string para numéricos</b>		
String x = "10"	int	int y = Integer.parseInt(x)
String x = "20.5"	float	float y = Float.parseFloat(x)
String x = "20.5"	double	double y = Double.parseDouble(x)
<b>De numéricos para string</b>		
int x = 10	String	String y = Integer.toString(x) ou String y = String.valueOf(x)
float x = 10.5	String	String y = Float.toString(x) ou String y = String.valueOf(x)
double x = 10.5	String	String y = Double.toString(x) ou String y = String.valueOf(x)

## Comentários em Java



- Comentários em Java são anotações que você pode adicionar ao seu código que não afetam a execução do programa. Eles são usados para adicionar descrições, notas ou qualquer outra informação útil para ajudar desenvolvedores a entender o código. Java suporta três tipos principais de comentários:
  - a.Comentários de uma única linha
  - b.Comentários de múltiplas linhas
  - c.Comentários de documentação

## Nota sobre comentários



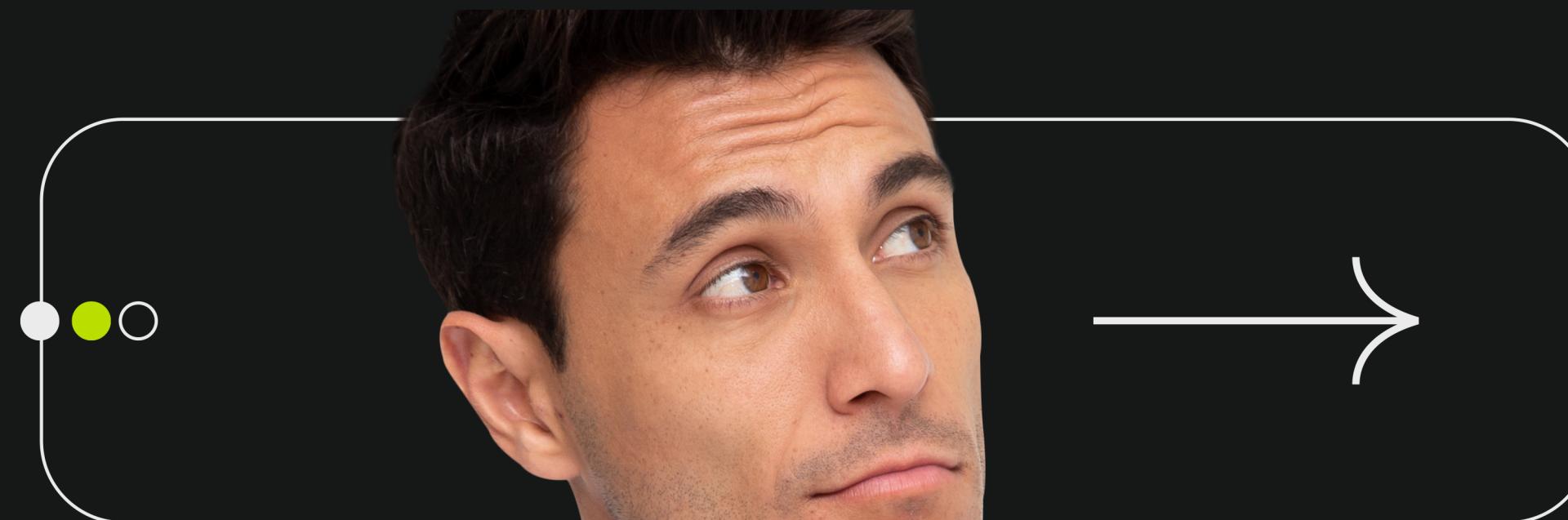
- **Quando usar comentários:**

- Para explicar porções de código que podem ser difíceis de entender.
- Para adicionar informações sobre autores, datas ou modificações.
- Para gerar documentação usando a ferramenta javadoc.
- Para deixar notas para futuras referências, como "TODO" ou "FIXME".

- **Quando evitar comentários:**

- Quando o código pode ser reescrito de forma mais clara, tornando o comentário desnecessário.
- Para explicar o óbvio. Comentários como `i++ // incrementa i` são redundantes e não adicionam valor.

# Estruturas de Controle



# Desvio Condicional

```
if (num1>=10) {  
    System.out.println("Condição verdadeira!");  
}else{  
    System.out.println("Condição falsa!");  
}
```

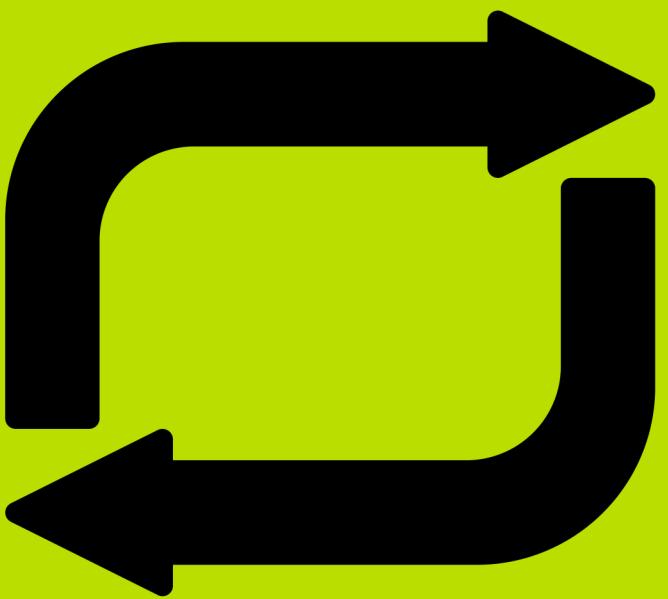
```
switch (op) {  
case 1:  
    System.out.println("Caso op igual a 1...");  
    break;  
case 2:  
    System.out.println("Caso op igual a 2...");  
    break;  
case 3:  
    System.out.println("Caso op igual a 3...");  
    break;  
default:  
    System.out.println("Caso op não seja 1, 2 ou 3");  
    break;  
}
```



## Controles de Repetição

Comandos:

- for
- while
- do
- break
- continue



## Comando for

---



```
for(int i=0; i<10; i++){  
    System.out.println(i);  
}
```

## Comando while

---



```
while(isActivate){  
    System.out.println("Ativado");  
    isActivate=false;  
}
```

## Comando do

---

```
boolean isActive = true;  
do{  
    System.out.println("Ativado");  
    isActive=false;  
}while(isActive)
```



## Comando break

O comando **break** em Java é utilizado para interromper a execução de loops (for, while, do-while) ou blocos de switch.



### Comando break - exemplo forçando a saída

```
for(int i = 0; i < 10; i++) {  
    if(i == 5) {  
        break; // Termina o loop quando i é igual a 5  
    }  
    System.out.println(i);  
}  
// A saída será: 0 1 2 3 4
```

## Comando continue

O comando continue em Java é usado **para pular a iteração** atual de um loop (for, while ou do-while) e continuar com a próxima iteração. Ao contrário do break, que interrompe completamente o loop, o continue simplesmente pula o resto da iteração atual e continua com o loop.



### Comando continue - exemplo pulando a iteração

```
for(int i = 0; i < 10; i++) {  
    if(i % 2 == 0) {  
        continue; // Pula a iteração atual se i é par  
    }  
    System.out.println(i);  
}  
// A saída será: 1 3 5 7 9
```



# CLASSE SCANNER

# Scanner

---

A classe Scanner fornece métodos para ler primitivos e strings de várias fontes, incluindo fluxos de entrada, strings e arquivos. É comumente usada para ler a entrada do teclado.



## **Características principais:**

1. Pode ler dados de tipos diferentes, como int, float, double, String, etc.
2. Suporta padrões regulares, o que pode ser útil para processar e validar a entrada.
3. É flexível e pode ser usada para ler de várias fontes, não apenas a entrada padrão.

# Scanner - Exemplo

```
import java.util.Scanner;

public class EntradaDados {

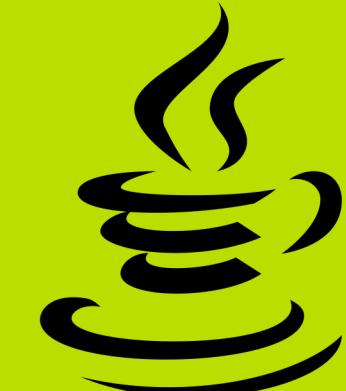
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // Cria um objeto Scanner

        System.out.println("Digite seu nome:");
        String nome = scanner.nextLine(); // Lê uma linha completa de entrada

        System.out.println("Digite sua idade:");
        int idade = scanner.nextInt(); // Lê um valor inteiro

        System.out.println("Olá, " + nome + "! Você tem " + idade + " anos.")

        scanner.close(); // Fecha o scanner para liberar recursos
    }
}
```





# EXERCÍCIOS PARA PRÁTICA

# Projeto: ProjetoAcampamento

## Exercício 1

1. Crie um projeto com o nome acima.
2. Como será o projeto:

Acampamento		
f	nome	String
f	equipe	char
f	idade	int
m	imprimir()	void
m	separarGrupo()	void

AcampamentoTeste		
J	main(String[])	void

## Exercício 1

Parte a) Desenvolver uma classe chamada **Acampamento** com os seguintes atributos: **nome, equipe, idade**. Em seguida implementar os **seguinte métodos**:

Método	Descrição
imprimir()	Este método não retorna valor e deve exibir os valores dos atributos na tela.
separarGrupo()	Este método não retorna valor e deverá verificar as seguintes condições: se a idade estiver entre 6 e 10 anos, atribuir A ao atributo equipe; e a idade estiver entre 11 e 20, atribuir B ao atributo equipe; se a idade for superior a 21 anos, atribuir C ao atributo equipe.

## Exercício 1

Parte b) Na segunda classe Java chamada AcampamentoTeste com a seguinte estrutura, realize no método main():

- Criar um objeto chamado membro da classe Acampamentos e atribuir valores aos seus atributos nome e idade.
- Executar o método imprimir() e analisar o resultado na tela.
- Executar o método separarGrupo().
- Executar o método imprimir() novamente e analisar o que será exibido na tela.

## Exercício 2

### ProjetoComputador

**Crie um projeto com o nome acima.**

**Como será o projeto:**

Computador		
f	marca	String
f	cor	String
f	modelo	String
f	numeroSerie	long
f	preco	double
m	imprimir()	void
m	calcularValor()	void
m	alterarValor(double)	int

ComputadorTeste		
m	main(String[])	void

## Exercício 2

NO GEFH

### ProjetoComputador

Parte a) Desenvolver uma classe chamada **Computador** com os seguintes atributos: **marca, cor, modelo, numeroSerie, preco**. Implementar os seguintes métodos:

Método	Descrição
imprimir()	Este método não retorna valor e deve exibir os valores dos atributos na tela.
calcularValor()	Não retorna valor e deverá verificar as seguintes condições: caso a marca seja HP, acrescentar 30% ao preço; caso seja IBM, acrescentar 50% ao preço; caso seja qualquer outra marca, manter o preço original.
alterarValor()	Este método recebe um valor como parâmetro. Atribuir este valor ao atributo preço, caso o valor do parâmetro recebido seja maior do que 0. Neste caso, o método alterarValor() deverá além de alterar o valor, retornar 1. Caso contrário não atribuir o valor ao atributo preço e retornar 0.

**Parte b)** Na segunda classe Java chamada **ComputadorTeste** realize:

- Criar um objeto da classe Computador e atribuir valores a seus atributos. Atribuir HP ao atributo marca.
- Executar o método imprimir() e analisar o que será exibido na tela.
- Executar o método calcularValor().
- Executar o método imprimir() e analisar o que será exibido na tela.
- Criar um segundo objeto e atribuir valores a seus atributos. Atribuir IBM ao atributo marca do novo objeto.
- Executar o método imprimir() do novo objeto e analisar o que será exibido na tela.
- Executar o método calcularValor() do novo objeto.
- Executar o método imprimir() do novo objeto e analisar o que será exibido na tela.
- Executar para o novo objeto o método alterarValor() com um valor positivo.
- Verificar no método main() o retorno do método alterarValor() e mostrar a mensagem de “Valor alterado” caso este retorne 1, e “Valor NÃO alterado” caso retorne 0.
- Executar o método imprimir() deste objeto e analisar o que será exibido na tela.

## Exercício 3

### ProjetoConta

**Crie um projeto com o nome acima.**

**Como será o projeto:**

Conta		
f	conta	String
f	agencia	String
f	saldo	double
f	nomeCliente	String
m	sacar(double)	int
m	depositar(double)	void
m	imprimir()	void

ContaTeste		
f	cc	Conta
m	main(String[])	void
m	execCadastrar()	void
m	execConsultar()	void
m	execSacar()	void
m	execDepositar()	void

## Exercício 3

### ProjetoConta

Parte a) Desenvolver uma classe Java chamada Conta com a seguinte estrutura: **conta, agencia, saldo e nomeCliente**. Em seguida implementar os seguintes métodos:

Método	Descrição
sacar()	Retorna valor 1 caso o saque seja realizado ou 0 se não houver saldo suficiente na conta. Deverá receber como parâmetro o valor a ser sacado.
depositar()	Realiza o depósito do valor recebido como parâmetro. Não deve retornar valor.
imprimir()	Exibe na tela os atributos da classe. Este método não retorna nada.

## Exercício 3

### ProjetoConta

**Parte b)** Na segunda classe java chamada **ContaTeste** defina:

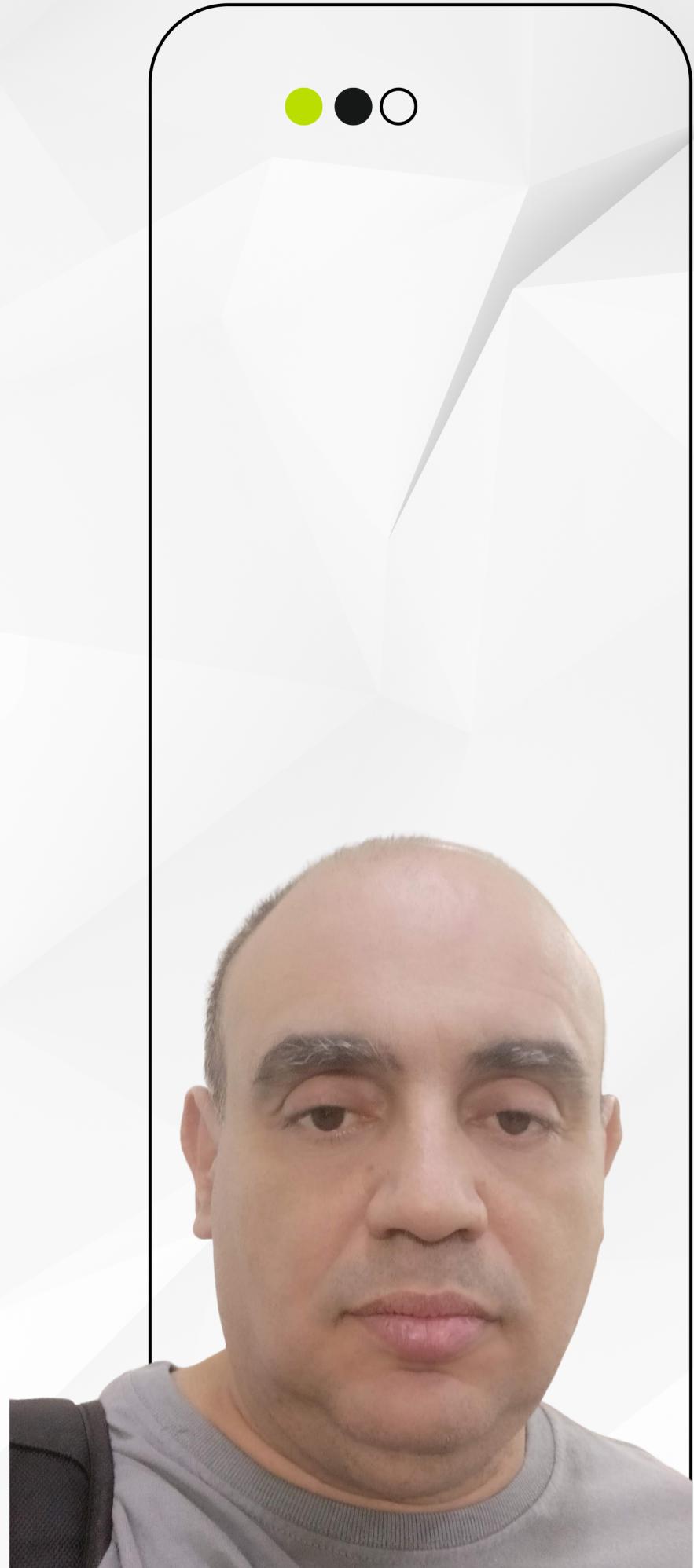
- Criar um atributo público da classe Conta para ser usado pelos métodos da classe para realizar saques e depósitos (fora do método main()). Não se esquecer de executar o operador new para o atributo criado.

Implemente os **métodos dispostos no próximo slide.**

# Exercício 3

## Projeto Conta

Método	Descrição
<b>main()</b>	Implementá-lo conforme padrão da linguagem Java. O método main() deverá criar um loop para o usuário escolher entre as opções cadastrar, depositar, sacar, consultar. Se for selecionada a opção sacar, executar o método <b>execSacar</b> . Se for selecionado depositar, executar o método <b>execDepositar</b> . Para a opção consultar, executar o método <b>execConsultar</b> . Para a opção cadastrar, executar o método <b>execCadastrar</b> .
<b>execSacar()</b>	Solicitar ao usuário que digite um valor e executar o método sacar() da classe ContasCorrentes usando o atributo criado. Testar o retorno do método sacar(). Se for retornado 1, exibir “Saque realizado”, caso contrário, exibir “Saque não realizado”.
<b>execDepositar()</b>	Solicitar ao usuário que digite um valor e executar o método depositar() da classe ContasCorrentes usando o objeto criado anteriormente.
<b>execConsultar()</b>	Apresentar os atributos na tela executando o método imprimir() da classe ContasCorrentes.
<b>execCadastrar()</b>	Solicitar que o usuário realize a leitura dos dados via teclado e em seguida realize a atribuição dos valores lidos do teclado aos atributos do objeto classe ContasCorrentes, criado como atributo dessa classe.



# Obrigado

## fim

Até a próxima aula



# Referências Bibliográficas



- Mendes; **Java com Ênfase em Orientação a Objetos**, Novatec.
- Deitel; **Java, como programar** – 10º edição. Java SE 7 e 8
- Arnold, Gosling, Holmes; **A linguagem de programação Java** – 4º edição.
- Apostilas da Caelum.