



# PROGRAMAÇÃO ORIENTADA A OBJETO - JAVA

PROFESSOR: DANILO FARIAS  
SEMESTRE: 2020.1

23/05/2022

# JAVA – DEFINIÇÃO DE CLASSE

CLASSE MAIN, ATRIBUTOS, MÉTODOS E CONSTRUTORES

```
1 public class Circle extends Shape{
2
3     int x,y;
4     double radius;
5
6     public Circle(int x, int y, int init_perimeter){
7         this.x=x;
8         this.y=y;
9         perimeter=init_perimeter;
10        this.radius = init_perimeter /(2*3.14);
11    }
12
13    public int draw(){
14        System.out.println("Draw Circle");
15        return 0;
16    }
17 }
```

# PROGRAMAÇÃO ORIENTADA A OBJETOS

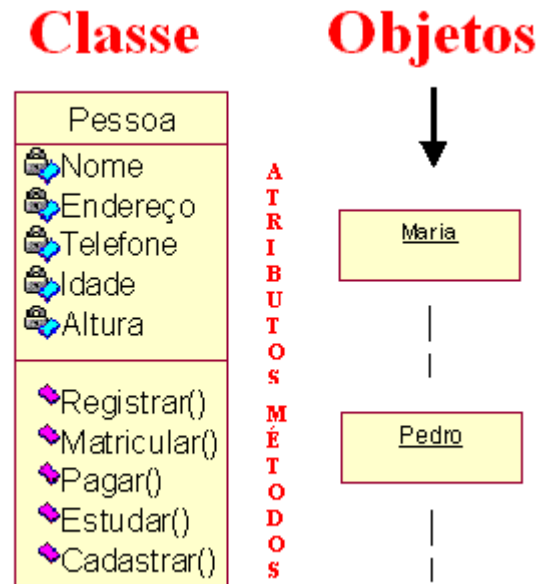
## Introdução

- A POO visa **aproximar** a maneira como construímos os programas de computador do mundo real
- O mundo real é composto por entidades que interagem, que trocam serviços entre si para realizar suas tarefas
- Na proposta de orientação a objeto, estas entidades são chamadas de **objetos**

# DESCREVENDO CLASSES

Classes são utilizadas para descrever a estrutura dos objetos:

- Encapsula:
  - Métodos, atributos e construtores;
- Oculta informações:
  - Privativas, públicas e protegidas.



# DESCREVENDO CLASSES

```
public class Cliente {  
  
    private String nome;  
    private String cpf;  
    private String endereco;  
  
    public void setNome(String nome) {}  
    public String getNome() {}  
  
}
```

# DESCREVENDO CLASSES



## Classe

```
public class NomeDaClasse {  
    CorpoDaClasse  
}
```

O corpo de uma classe pode conter

- atributos
- métodos
- construtores (inicializadores)
- outras classes...

# DESCREVENDO CLASSES



## Classe

```
public class Conta {  
  
    // Atributos  
    private Cliente cliente;  
    private double saldo;  
    private double numero;  
  
    // Métodos  
    //sacar  
    public void sacar(double valor){  
        if (this.saldo >= valor) {  
            this.saldo = this.saldo - valor;  
        }else{  
            // Inconstitucional...  
            System.out.println("Saldo insuficiente.");  
        }  
    }  
}
```



# CLASSE MAIN

## Classe

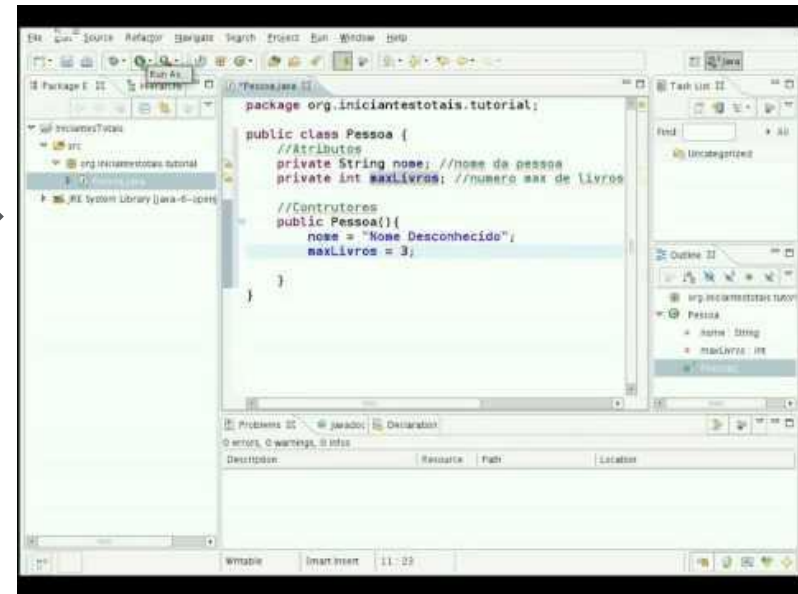
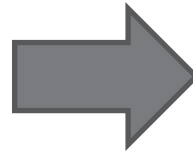
```
public class <nome> {  
    public static void main (<parametros>)  
    {  
        <declarações>  
        <comandos>  
    }  
}
```

Onde, **main**: método por onde se inicia a execução  
**public**: parâmetro de acesso  
**static**: indica que main se aplica à classe  
**void**: indica que main não retorna um valor



# VAMOS CRIAR A CLASSE MAIN?

```
public class <nome> {  
    public static void main (<parametros>) {  
        <declarações>  
        <comandos>  
    }  
}
```



## JAVA - DEFINIÇÃO DE ATRIBUTOS

Atributos

```
public class Cliente {  
  
    private String nome;  
    private String cpf;  
    private String endereco;  
  
    public void setNome(String nome) {}  
    public String getNome() {}  
  
}
```

# DESCREVENDO CLASSES

Nome da Classe

```
public class Cliente {  
  
    private String nome;  
    private String cpf;  
    private String endereco;  
  
    public void setNome(String nome) {}  
    public String getNome() {}  
  
}
```

# DESCREVENDO CLASSES

Atributos

```
public class Cliente {  
  
    private String nome;  
    private String cpf;  
    private String endereco;  
  
    public void setNome(String nome) {}  
    public String getNome() {}  
  
}
```

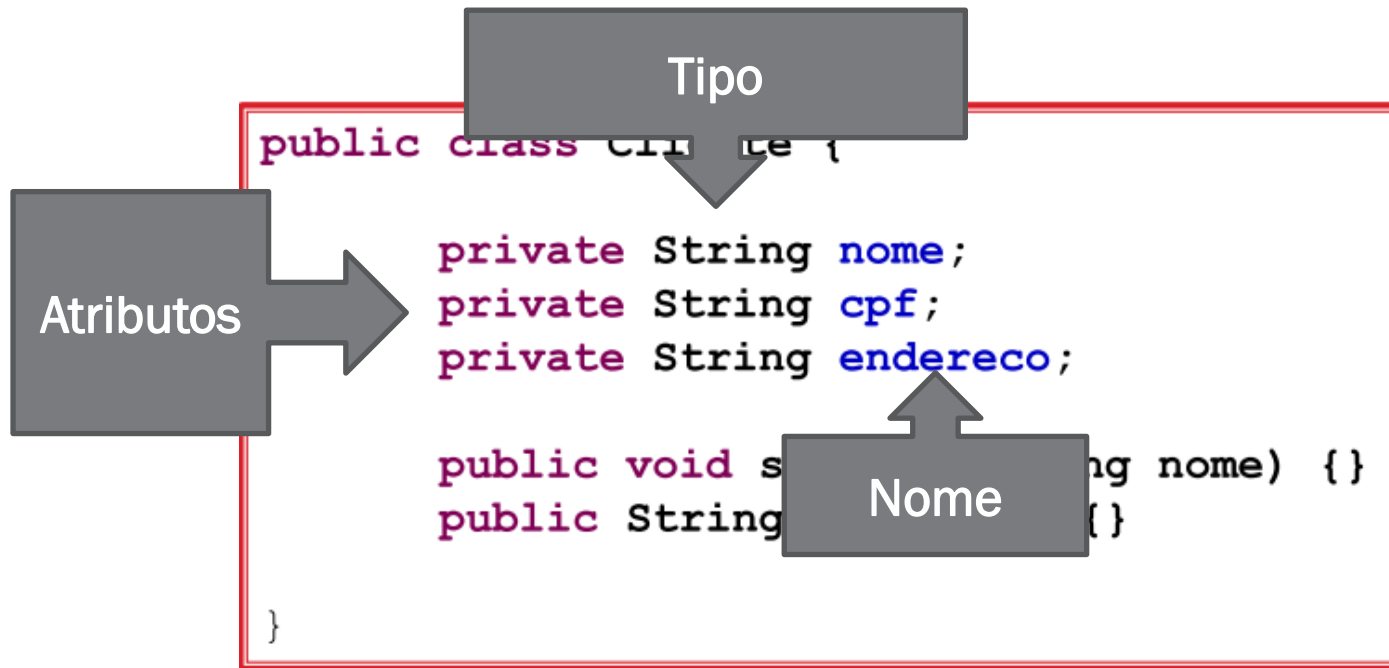
# DEFINIÇÃO DE ATRIBUTOS

Atributos

```
public class Pessoa {  
    private int anoDeNascimento;  
    private String nome, sobrenome;  
    private boolean casado = false;  
    ...  
}
```

- Vários atributos de um mesmo tipo podem ser declarados conjuntamente.
- Podemos especificar que um atributo deve ser inicializado com um valor específico.

# DESCREVENDO CLASSES



# DECLARAÇÃO DE VARIÁVEIS

- Em Java, toda variável tem um tipo que não pode ser mudado, uma vez que declarado:

```
tipoDaVariavel nomeDaVariavel;
```

- Variáveis do mesmo tipo podem ser declaradas de uma única vez:

```
double saldo, salario;
```

# TIPOS DE DADOS EM JAVA

## Primitivos

- char
- inst
- boolean
- double
- ...

## Referenciais

- classes (String, Object, Livro, Conta, etc.)
- interfaces
- arrays

Os elementos de um tipo primitivo são valores, enquanto os elementos de um tipo referência são (referências para) objetos!



# TIPOS DE DADOS EM JAVA

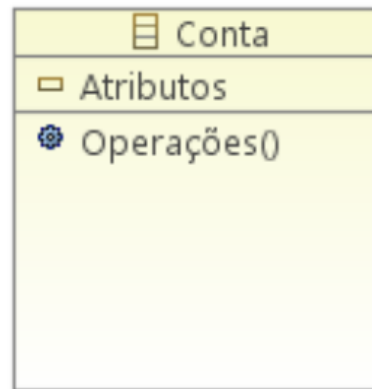
- Java define 8 tipos primitivos, com diferentes tamanhos e valores:

TIPO	FAIXA
byte (8bits)	-128 a 127
short (16bits)	-32768 a 32767
int (32bits)	-2147483648 a 2147483647
long (64bits)	-9223372036854775808 a 9223372036854775807
float (32 bits)	single-precision 32-bit IEEE 754 floating point
double (64 bits)	double-precision 64-bit IEEE 754 floating point
char (16 bits)	'\u0000' (ou 0) a '\uffff' (or 65.535 inclusive)
boolean	Assume true ou false (tamanho não definido)

# CRIANDO UM NOVO TIPO DE DADOS EM JAVA!

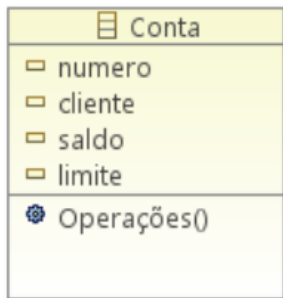


- Em um sistema para um banco percebemos que uma entidade importante para esse sistema é a **conta**.
- Vamos generalizar alguma informação, juntamente com funcionalidades que toda **conta** deve ter.



# TIPOS DE DADOS EM JAVA

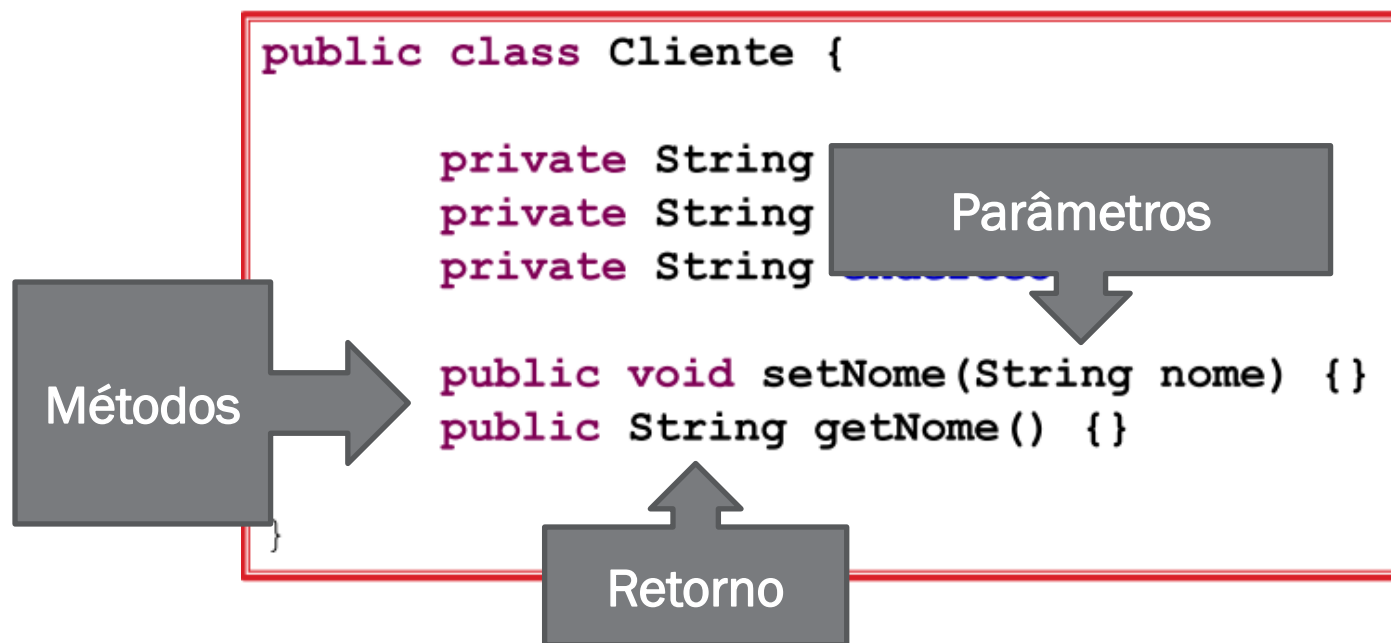
- O que toda conta tem e é importante para nós?
  - número da conta
  - nome do dono da conta
  - saldo
  - limite



```
class Conta {  
  
    int numero;  
    String cliente;  
    double saldo;  
    double limite;  
  
}
```

**String** - String é uma classe em Java. Ela guarda uma cadeia de caracteres, uma frase completa.

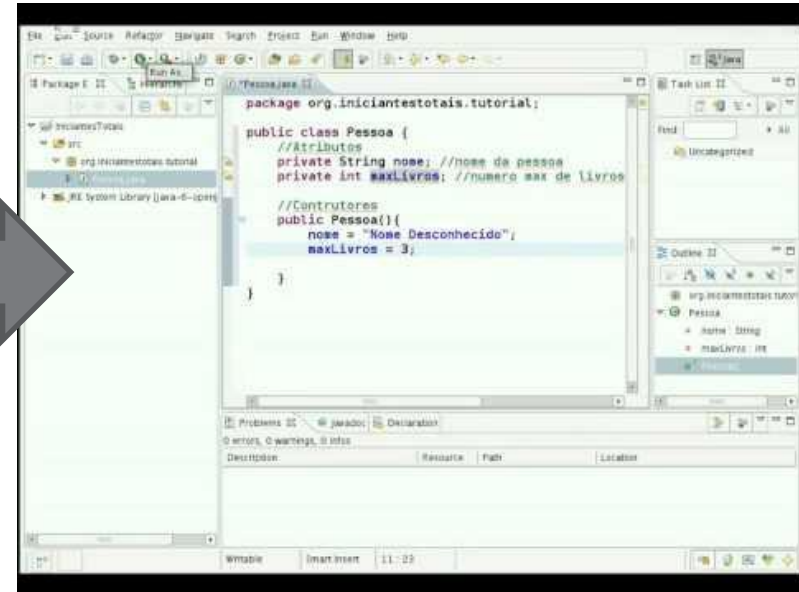
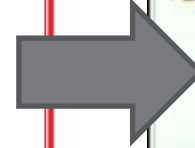
# DESCREVENDO CLASSES



# VAMOS CRIAR A CLASSE CLIENTE?



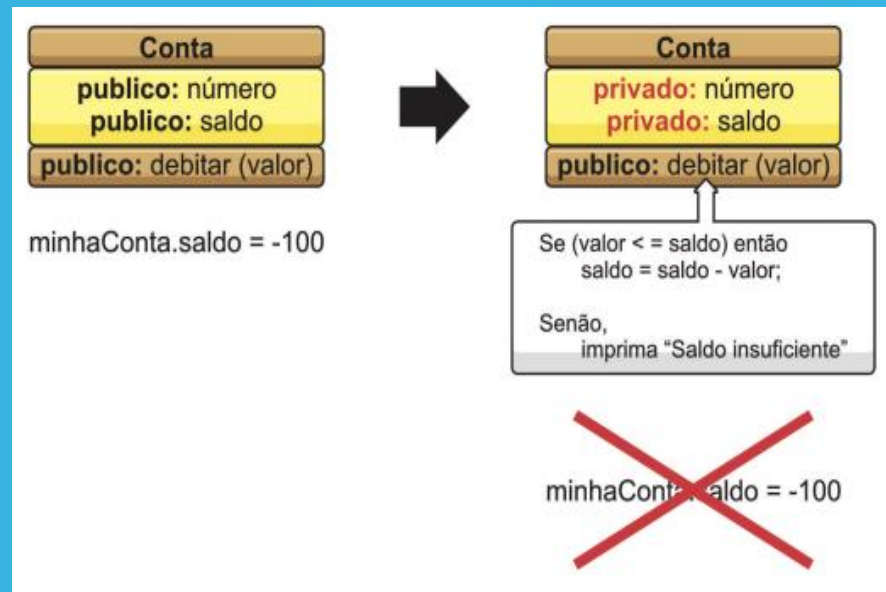
```
public class Cliente {  
  
    private String nome;  
    private String cpf;  
    private String endereco;  
  
    public void setNome(String nome) {}  
    public String getNome() {}  
  
}
```



23/05/2022



## JAVA – ENCAPSULAMENTO



# ENCAPSULAMENTO

## ○ Encapsulamento

- Consiste em separar os aspectos externos de um objeto dos detalhes internos de implementação do objeto
- Evita que objetos possuam grandes dependências entre si, de modo que uma simples mudança em um objeto possa trazer grandes efeitos colaterais e problemas para outros objetos
- Em POO, o acesso a componentes de um objeto é controlado e, especialmente, os atributos de um objeto só devem ser modificados pelos métodos do próprio objeto

# MODIFICADORES DE ACESSO

```
public class Cliente {  
  
    private String nome;  
    private String cpf;  
    private String endereco;  
  
    private void setNome(String nome) {}  
    private String getNome() {}  
}
```

A palavra reservada **private** indica que os atributos só podem ser acessados (isto é, lidos ou modificados) pelas operações da classe correspondente.



# MODIFICADORES DE ACESSO

- Java não obriga o uso de **private**, mas vários autores consideram isto uma pré-condição para programação orientada a objetos
- Grande impacto em extensibilidade
- Usem **private**!

Modificador	Classe	Pacote	Subclasse	Globalmente
Public	sim	sim	sim	sim
Protected	sim	sim	sim	não
Sem Modificador (Padrão)	sim	sim	não	não
Private	sim	não	não	não

Fonte: [Controlling Access to Members of a Class](#)

# MODIFICADORES DE ACESSO

## ○ public

- Usados em:
  - Atributos
    - Podem ser acessados (lidos, alterados) por objetos de qualquer classe
  - Métodos
    - Podem ser chamados por métodos de qualquer classe
  - Classes
    - Podem ser instanciados por qualquer classe

# MODIFICADORES DE ACESSO

## ○ **protected**::Subclasses + Pacotes

- Usado somente para:
  - Atributos
    - Podem ser acessados (lidos, alterados) por objetos de classes dentro do mesmo pacote ou de qualquer subclasse da classe ao qual ele pertence
  - Métodos
    - Podem ser chamados por objetos de classes dentro do mesmo pacote ou de qualquer subclasse da classe ao qual ele pertence

# MODIFICADORES DE ACESSO

- **default (friendly) :: Pacotes**
  - Usado em:
    - Atributos
      - Só são visíveis para objetos de classes do mesmo pacote
    - Métodos
      - Só são chamados a partir de objetos de classes do mesmo pacote
    - Classes
      - Só são visíveis por classes do mesmo pacote

# MODIFICADORES DE ACESSO

## ○ private

- Usado em:
  - Atributos
    - Só podem ser acessados por objetos da mesma classe
  - Métodos
    - Só podem ser chamados por métodos da classe onde são declarados

# MODIFICADORES DE ACESSO

## ◦ final

- Usado em:
  - Atributos
    - Torna o atributo constante
  - Métodos
    - Não podem ser redefinidos
  - Classes
    - Não podem ser estendidas

# MODIFICADORES DE ACESSO

## ○ static

- Usado em:
  - Atributos
    - Atributos static pertencem à classe e não aos objetos
    - Só existe uma cópia de um atributo static de uma classe, mesmo que haja vários objetos da classe
    - São muito usados para constantes
    - O acesso é feito usando o nome da classe:  
`int numCheques = numTaloos * ConstantesBanco.NUN_CHEQUES_TALAO;`
  - Métodos
    - Pertencem a classes e não a objetos
    - Podem ser usados mesmo se criar os objetos
    - Só podem acessar diretamente atributos estáticos
    - O acesso é feito usando o nome da classe:  
`x = Math.random();`

# ENCAPSULAMENTO

- Todos os atributos de uma classe normalmente são **privados**
- São criados os métodos get's e set's para dar acesso de leitura e escrita aos atributos de uma classe



# ENCAPSULAMENTO – FIQUEM DE OLHO!

```
public class Conta {  
    private String numero;  
    private float saldo;  
    private Pessoa correntista;  
    public Conta(String numero, float saldo, Pessoa correntista) {  
  
    public Conta(String num, Pessoa pessoa){  
  
    public boolean debitar(float valor){  
        boolean retorno = false;  
        if(this.saldo - valor >= 0){  
            this.saldo -= valor;  
        }  
        return retorno;  
    }  
  
    public String getNumero() {  
        return numero;  
    }  
  
    public float getSaldo() {  
        return saldo;  
    }  
  
    public Pessoa getCorrentista() {  
        return correntista;  
    }  
  
    public void creditar(float valor){  
  
    public void transferencia(float valor, Conta destino){  
  
}
```

Se uma classe externa tivesse acesso ao atributo saldo, o cliente iria poder ter um saldo negativo

# ENCAPSULAMENTO – FIQUEM DE OLHO!

```
public class Teste {  
    public static void main(String[] args) {  
        ...  
        conta.numero = "010020-XX";  
        String numero = conta.numero;  
    }  
}
```

**ERRO!**

A classe Teste não pode acessar o atributo numero da classe Conta, pois este atributo está privado.

```
public class Teste {  
    public static void main(String[] args) {  
        ...  
        String numero = conta.getNumero();  
        System.out.println(numero);  
    }  
}
```

**OK!**

A classe Teste pode acessar o atributo numero da classe Conta através dos métodos acessores (get's e set's)

**OBS: Perceba que o atributo número não pode ser modificado depois que o objeto é instanciado, pois ele está privado e Conta não possui o método setNumero.**

# DESCREVENDO CLASSES

Todo arquivo .java deve ter pelo menos uma classe pública

```
public class Cliente {  
  
    private String nome;  
    private String cpf;  
    private String endereco;  
  
    public void setNome(String nome) {}  
    public String getNome() {}  
  
}
```

# DESCREVENDO CLASSES

Publicitação de Informações



```
public class Cliente {  
  
    private String nome;  
    private String cpf;  
    private String endereco;  
  
    public void setNome(String nome) {}  
    public String getNome() {}  
  
}
```

Ocultamento de  
Informações



# INSTANCIANDO OBJETOS DE UMA CLASSE

```
public static void main(String[] args) {  
    Cliente cliente = new Cliente();  
}
```

# INSTANCIANDO OBJETOS DE UMA CLASSE



```
public static void main(String[] args) {  
    Cliente cliente = new Cliente();  
}
```

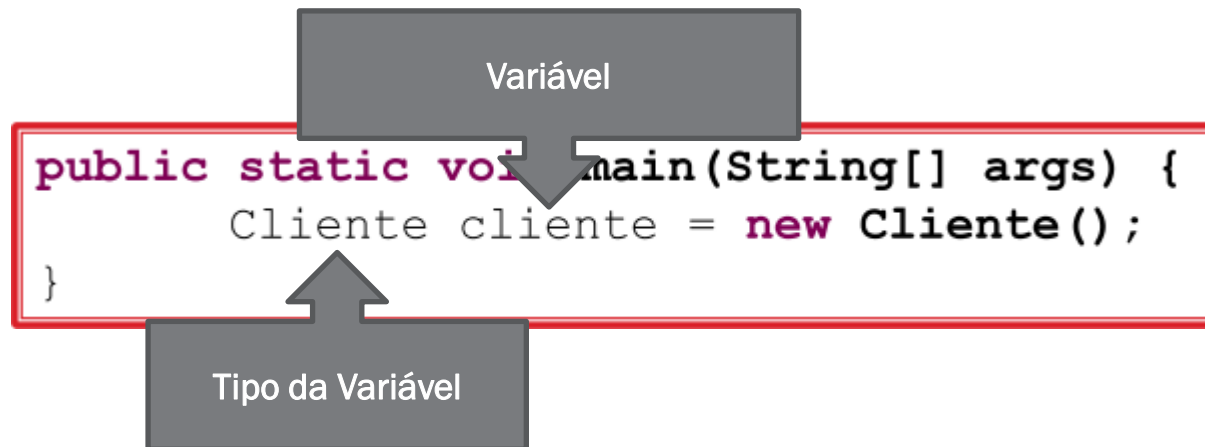
# INSTANCIANDO OBJETOS DE UMA CLASSE

```
public static void main(String[] args) {  
    Cliente cliente = new Cliente();  
}
```

Tipo da Variável

Cliente

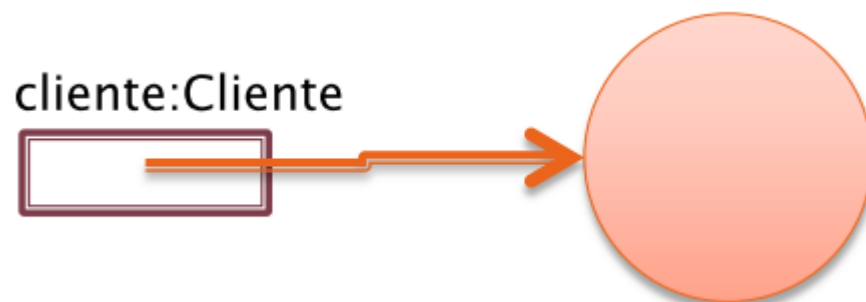
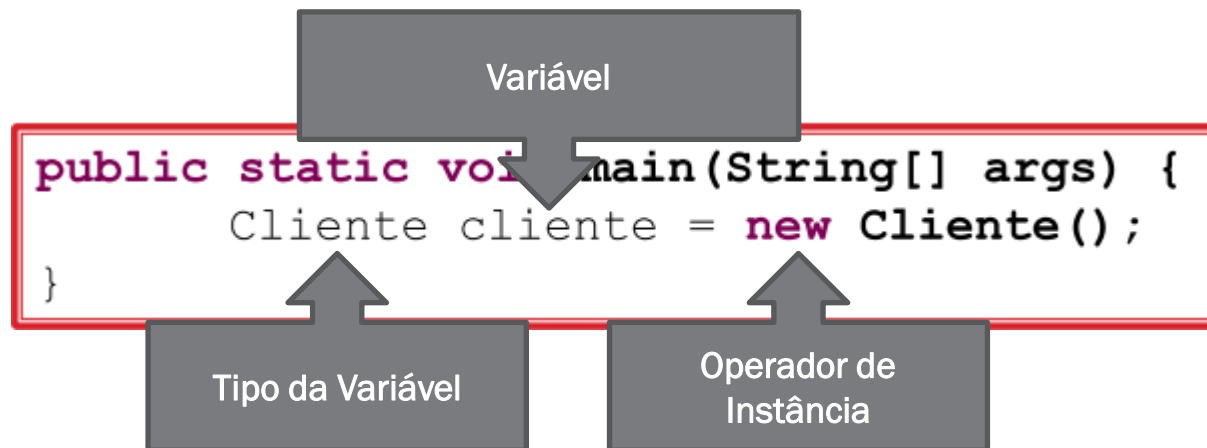
# INSTANCIANDO OBJETOS DE UMA CLASSE



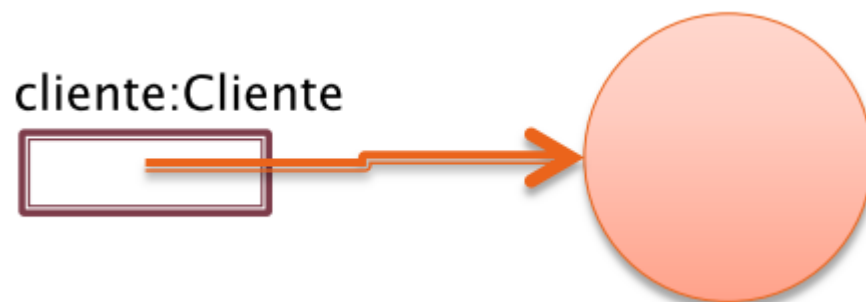
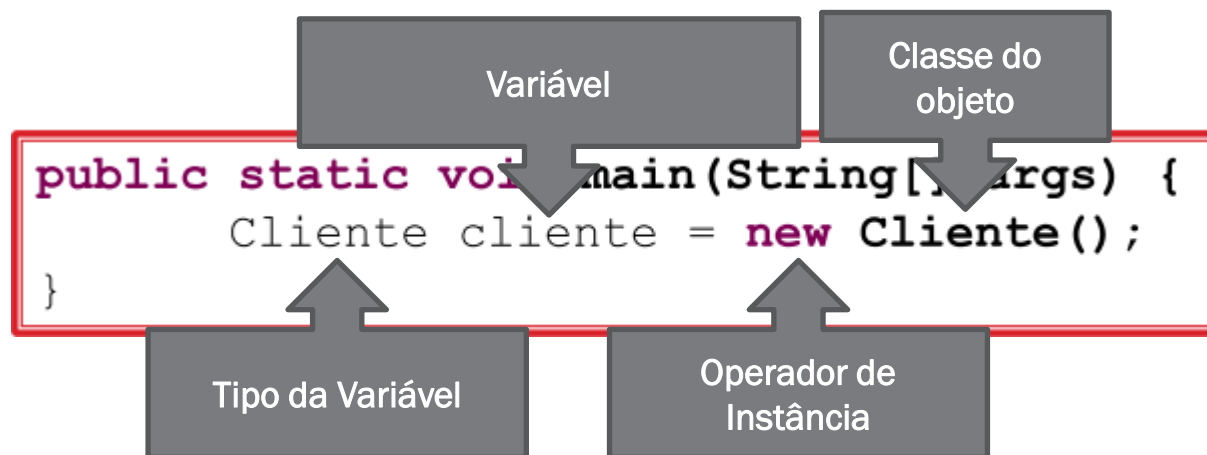
cliente:Cliente



# INSTANCIANDO OBJETOS DE UMA CLASSE



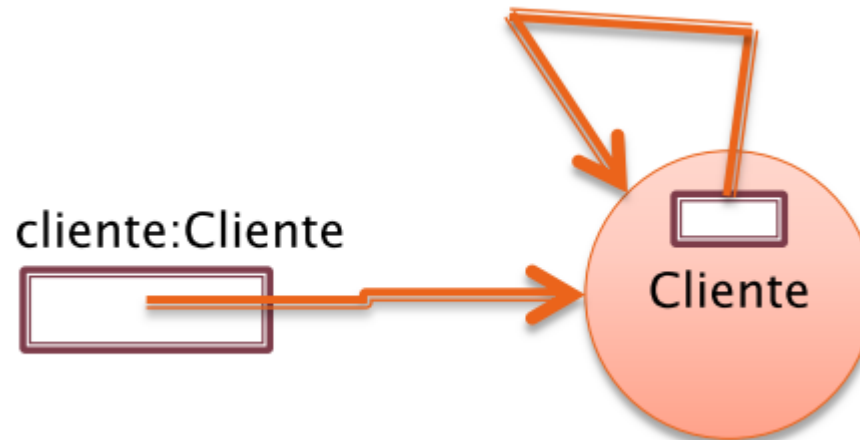
# INSTANCIANDO OBJETOS DE UMA CLASSE



# OPERADOR THIS

Um “atributo” que possui uma referência para si mesmo:

- Utilizado para o objeto acessar os seus atributos e métodos.



## DEFININDO DOS CONSTRUTORES

```
public class Conta {  
    String numero;  
    float saldo;  
    Pessoa correntista;  
    Conta(String num, float valor, Pessoa pessoa) {  
        numero = num;  
        saldo = valor;  
        correntista = pessoa;  
    }  
  
    Conta(String num, Pessoa pessoa){  
        numero = num;  
        correntista = pessoa;  
        saldo = 0.0f;  
    }  
  
    void debitar(float valor){  
  
    void creditar(float valor){  
  
    }
```

# CONSTRUTORES

- Objetos precisam ser criados antes de serem utilizados
- Construtores precisam ser definidos na classe
- O operador **new** é usado para **criar uma nova instancia** de uma classe (objeto)

- `Conta c = new Conta();`

Construtor

# CONSTRUTORES

## ○ Exemplo:

1. Conta conta;
2. conta.setNumero("10.200-x"); **ERRO**
3. conta = new Conta();
  - O erro acima ocorreu porque conta ainda não tinha sido instanciada, ou seja, ela estava referenciando *null*
1. Conta conta;
2. conta = new Conta();
3. conta.setNumero("10.200-x"); //OK

# CONSTRUTORES

- Construtores definem como os atributos de uma classe devem ser inicializados
- São semelhantes a métodos, mas não têm tipo de retorno
- O nome do construtor deve ser exatamente o nome da classe
- Uma classe pode ter diversos construtores, diferenciados pelos parâmetros

# CONSTRUTORES

- Exemplo:

```
public class Conta {  
    String numero;  
    float saldo;  
    Pessoa correntista;  
  
    Conta(String num, float valor, Pessoa pessoa) {  
        numero = num;  
        saldo = valor;  
        correntista = pessoa;  
    }  
  
    Conta(String num, Pessoa pessoa) {  
        numero = num;  
        correntista = pessoa;  
        saldo = 0.0f;  
    }  
  
    void debitar(float valor) {  
  
    }  
  
    void creditar(float valor) {  
  
    }  
}
```



# CONSTRUTORES

- Usando um construtor:

```
Pessoa correntista = new Pessoa();
```

```
Conta conta = new Conta("10.299-0", 100.0f, correntista);
```

# CONSTRUTORES

- Quando não é definido um construtor explicitamente, um construtor *default* é fornecido implicitamente
- O construtor *default* inicializa os atributos com seus valores default
- O construtor *default* não tem parâmetros

# CONSTRUTORES

- Exemplo

```
public class Conta {  
    String numero;  
    float saldo;  
    Pessoa correntista;  
  
    void debitar(float valor) {..  
  
    void creditar(float valor) {..  
  
}
```

- Mesmo sem construtor explicito, ainda podemos instanciar esta classe devido ao construtor *default*.
  - Conta conta = new Conta();

# CONSTRUTORES

- Quando criamos explicitamente algum construtor, o construtor default não existirá

```
public class Conta {  
    String numero;  
    float saldo;  
    Pessoa correntista;  
    Conta(String num, float valor, Pessoa pessoa) {  
        numero = num;  
        saldo = valor;  
        correntista = pessoa;  
    }  
    ...  
}
```

- Conta conta = new Conta(); **ERRO**

# DEFININDO OS MÉTODOS

```
public class Cliente {  
    private String nome;  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return this.nome;  
    }  
}
```

# DEFININDO OS MÉTODOS

```
public class Cliente {  
    private String nome;  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return this.nome;  
    }  
}
```

# DEFININDO OS MÉTODOS

```
public class C {  
    private String nome;  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return this.nome;  
    }  
}
```

Nome do método

Retorno do método

# DEFININDO OS MÉTODOS

```
public class C {  
    private String nome;  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return this.nome;  
    }  
}
```

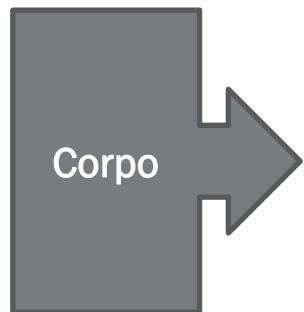
Nome do método

Retorno do método

Parâmetros do método




# DEFININDO OS MÉTODOS



```
public class Cliente {  
    private String nome;  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return this.nome;  
    }  
}
```

# DEFININDO OS MÉTODOS

```
public class Cliente {  
    private String nome;  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return this.nome;  
    }  
}
```



Referencia o próprio  
objeto

# INSTANCIANDO E INVOCANDO MÉTODOS

```
public class Programa {  
    public static void main(String[] args){  
        Cliente cliente = new Cliente();  
        cliente.setNome("Robson Medeiros");  
  
        System.out.println("O nome do cliente é: " + cliente.getNome());  
    }  
}
```

Instanciando a  
classe Cliente

# INSTANCIANDO E INVOCANDO MÉTODOS

```
public class Programa {  
    public static void main(String[] args){  
        Cliente cliente = new Cliente();  
        cliente.setNome("Robson Medeiros");  
  
        System.out.println("O nome do cliente é: " + cliente.getNome());  
    }  
}
```


Usando o  
método  
setNome

# INSTANCIANDO E INVOCANDO MÉTODOS

```
public class Programa {  
    public static void main(String[] args){  
        Cliente cliente = new Cliente();  
        cliente.setNome("Robson Medeiros");  
  
        System.out.println("O nome do cliente é: " + cliente.getNome());  
    }  
}
```



O método println



Usando o método  
getNome

# DEFININDO OS MÉTODOS

```
public class Conta {  
    private String    numero;  
    private double    saldo;  
    private Cliente  cliente;  
  
    public void creditar(double valor) {  
        saldo = saldo + valor;  
    }  
    ...  
}
```

**Um método é uma operação que realiza ações e modifica os valores dos atributos do objeto responsável pela sua execução**

# DEFININDO OS MÉTODOS

```
public class Conta {  
    ...
```

Parâmetros  
do método

```
    public void debitar(double valor) {  
        saldo = saldo - valor;  
    }
```

Tipo de  
retorno

Corpo do  
método

Por quê o debitar não tem como parâmetro o número da conta?

# OPERADORES EM JAVA

- Atribuição

=, +=, -=, \*=, /=

- Unários

++, --

- Ternários (condicional)

$A = x ? B : c \Rightarrow \text{if } (x) \{a = b;\} \text{ else } \{a = c;\}$



# OPERADORES EM JAVA

- Aritméticos

+ - \* / %

O operador % calcula o resto de uma divisão inteira.

- Concatenação

+ (String)

## Relacionais

>, <, >=, <=, ==, !=

# OPERADORES EM JAVA

- Lógicos
  - Avaliação parcial
    - `&&` (E lógico)
    - `||` (OU lógico)
  - Avaliação Completa
    - `&` (E lógico bit-a-bit)
    - `|` (OU lógico bit-a-bit)
    - `^` (OU exclusivo bit-a-bit)

# EXERCÍCIO 1

Construir a classe ContaBancaria com:

- Atributos: **double** saldo , **Cliente** cliente e **String** numero
- Métodos: **creditar** e **debitar**

# DEFININDO OS MÉTODOS

- O tipo do valor a ser retornado pelo método.
- Nome do método.
- Lista, possivelmente vazia, indicando o tipo e o nome dos argumentos a serem recebidos pelo método.

Usa-se **void** para indicar que o método não retorna nenhum valor, apenas altera os valores dos atributos de um objeto

# DEFININDO OS MÉTODOS

- Quando alguém pedir para depositar, ele também vai dizer quanto quer depositar.
- Por isso precisamos declarar o método com algo dentro dos parênteses - o que vai aí dentro é chamado de argumento do método (ou parâmetro).

```
class {  
    // ... outros atributos e métodos ...  
  
    void deposita(double quantidade) {  
        this.saldo += quantidade;  
    }  
}
```

Sem retorno

# DEFININDO OS MÉTODOS

- Métodos com retorno:

```
class Conta {  
    //...  
    public String getCliente() {  
        return this.cliente;  
    }  
  
    public double getSaldo() {  
        return this.saldo;  
    }  
}
```

**Com retorno**

## EXERCÍCIO 2

Escrever um programa que cria duas contas bancárias com saldo 50.0 e 20.0 e debita 35.0 da primeira e credita 35.0 na segunda.

- Ao final exibe o número e o saldo de cada conta.



# PROGRAMAÇÃO ORIENTADA A OBJETO - JAVA

PROFESSOR: DANILO FARIAS  
SEMESTRE: 2020.1

23/05/2022