Sejam bem-vindos!

PROGRAMAÇÃO ORIENTADA A OBJETOS

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Diogenes Carvalho Matias

Formação:

- **Graduação: Sistemas de Informação;**
- **Especialista em: Engenharia e Arquitetura de Software;**
- **MBA EXECUTIVO EM BUSINESS INTELLIGENCE (em andamento);**
- **Mestrado Acadêmico em Engenharia de Computação (UPE em andamento);**

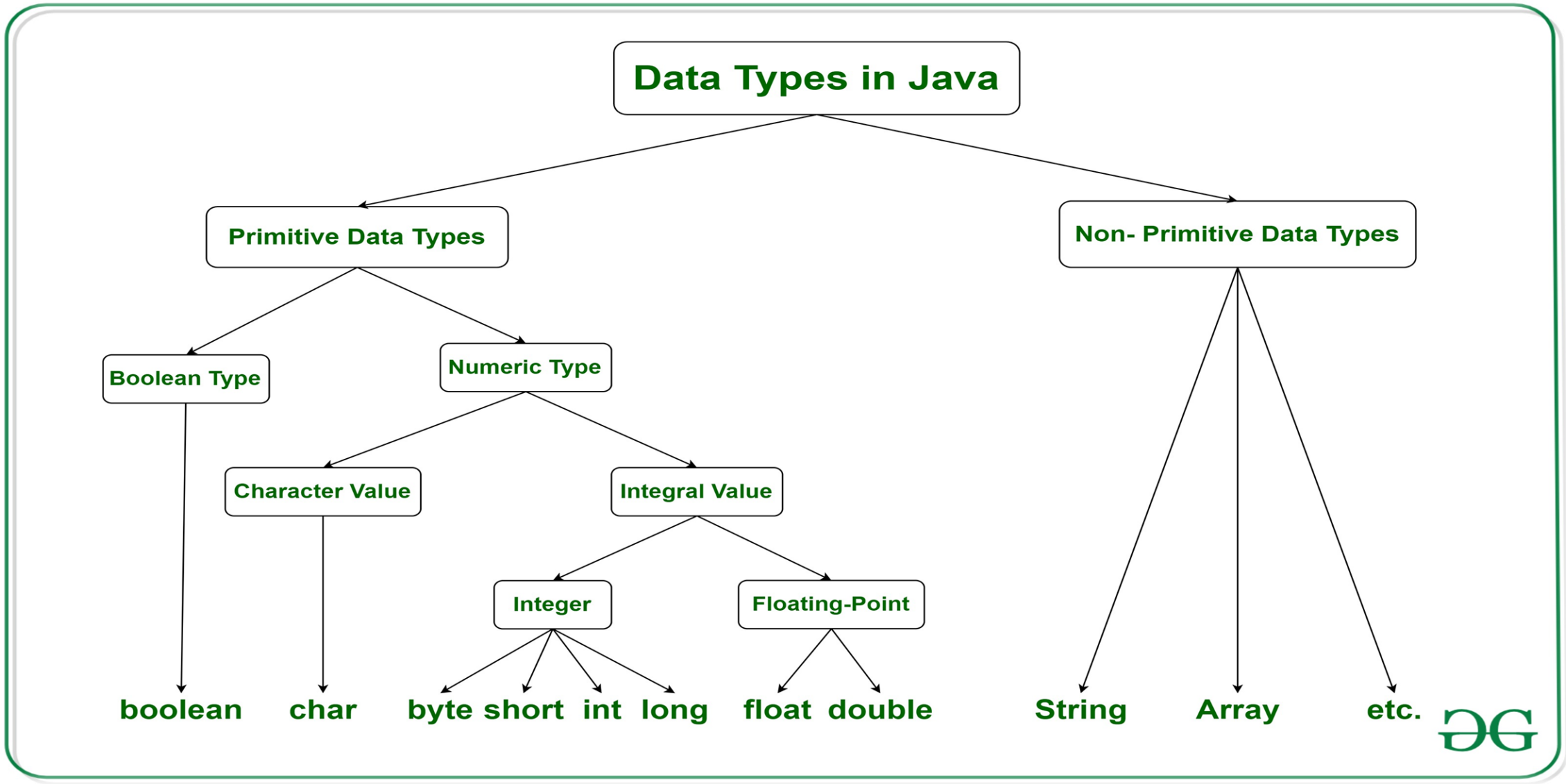
Maiores informações :

Linkedin

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Estrutura Geral de um Programa

```
public class nome do programa {
      public static void main(String[] args) {



      }
}
```


Data Types in Java
Primitive Data Types
Non- Primitive Data Types
Boolean Type
Numeric Type
Character Value
Integral Value
Integer
Floating-Point
boolean
char
byte
short
int
long
float
double
String
Array
etc.

Tipos Primitivos de Dados em Java

Algoritmos	Equivalentes em Java	Tamanho	Valores Permitidos
inteiro	byte short int long	1 byte 2 bytes 4 bytes 8 bytes	-128...127 -32768...32767 $-2^{31}...2^{31}-1$ $-2^{63}...2^{63}-1$
real	float double	4 bytes 8 bytes	
lógico	boolean	1 byte	**true** e **false**
caractere	char	2 bytes	
literal	String		

OBS: Valores do tipo **char** devem ser informados entre aspas simples (exemplo: 'A', 'c') enquanto valores do tipo **String** entre aspas duplas (“UNIFG”, “Diógenes Carvalho Matias”). Valores do tipo **real** utilizam o ponto decimal como nos algoritmos.

Operadores

Operador	Símbolo	Tipo	Para que serve	Precedência
Inverção de sinal	-	Unário	Inverte o sinal de um valor numérico.	7
Manutenção de sinal	+	Unário	Mantém o sinal de um valor numérico.	7
Negação	!	Unário	Inverte o valor de uma expressão lógica.	7
Divisão	/	Binário	Divide dois valores numéricos.	6
Multiplicação	*	Binário	Multiplica dois valores numéricos.	6
Resto da divisão	%	Binário	Dá o resto da divisão entre dois inteiros.	6
Adição	+	Binário	Adiciona dois valores numéricos. Concatena dois literais.	5

Operadores

Subtração	-	Binário	Subtrai dois valores numéricos.	5
Menor que	<	Binário	Verifica se um valor é menor do que outro.	4
Maior que	>	Binário	Verifica se um valor é maior do que outro.	4
Menor ou igual	<=	Binário	Verifica se um valor é menor ou igual do que outro.	4
Maior ou igual	>=	Binário	Verifica se um valor é maior ou igual do que outro.	4
Igual a	==	Binário	Verifica se um valor é igual a outro.	3
Diferente	!=	Binário	Verifica se um valor é diferente a outro.	3
Conjunção (.e.)	&&	Binário	Conjunção de dois valores lógicos.	2
Disjunção (.ou.)	||	Binário	Disjunção de dois valores lógicos.	1

Atribuição

Nos algoritmos em pseudocódigo, representávamos a atribuição com o símbolo ←.

Operador	Exemplo	Significado
=	`A = B`	Atribuição simples
+=	`A += B`	A = A + B
-=	`A -= B`	A = A - B
*=	`A *= B`	A = A * B
/=	`A /= B`	A = A / B
%=	`A %= B`	A = A % B
++	`A++`	A = A + 1
--	`A--`	A = A - 1

Variáveis

Variáveis em Java podem ser declaradas em qualquer lugar do programa, desde que dentro do método **main**. Para se declarar uma variável em Java, basta escrever seu tipo seguido do nome da variável.

```
int a, b;
char caractere;
String texto, msg;
boolean flag;
```

Conversão de Valores

A conversão entre tipos numéricos pode ser feita diretamente, informando o nome do tipo alvo a frente do valor a ser convertido, entre parênteses.

```
int x = (int) 10.0;
double z = (double) (a * 4);
byte y = (byte) b;
```

Conversão de Valores

```
public class Main {
  public static void main(String[] args) {
    double mumeroDouble = 7.78;
    int mumeroInt = (int) mumeroDouble;

    System.out.println(mumeroDouble);
    System.out.println(mumeroInt);
  }
}
```

Conversão de Valores

A conversão entre tipos numéricos pode ser feita diretamente, informando o nome do tipo alvo a frente do valor a ser convertido, entre parênteses.

Conversão de String para byte: `Byte.parseByte ( string )`
Conversão de String para short: `Short.parseShort ( string )`
Conversão de String para int: `Integer.parseInt ( string )`
Conversão de String para long: `Long.parseLong ( string )`
Conversão de String para float: `Float.parseFloat ( string )`
Conversão de String para double: `Double.parseDouble ( string )`
Conversão de String para boolean: `Boolean.parseBoolean ( string )`

Conversão de byte para String: `Byte.toString ( byte )`
Conversão de short para String: `Short.toString ( short )`
Conversão de int para String: `Integer.toString ( int )`
Conversão de long para String: `Long.toString ( long )`
Conversão de float para String: `Float.toString ( float )`
Conversão de double para String: `Double.parseDouble ( double )`
Conversão de boolean para String: `Boolean.parseBoolean ( boolean )`

Saída no Console

Com quebra automática de linha:

System.**out**.**println**(variável ou valor String);

Sem quebra automática de linha:

System.**out**.**print**(variável ou valor String);

Precedência e associatividade

```java
int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;


System.out.println("a+b/d = " + (a + b / d));

System.out.println("a+b*d-e/f = " + (a + b * d - e / f));
```

Qual o resultado dessas expressões acima?

Entrada em Console

OBS.: Para utilizar **Scanner** é necessário importar o pacote java.util

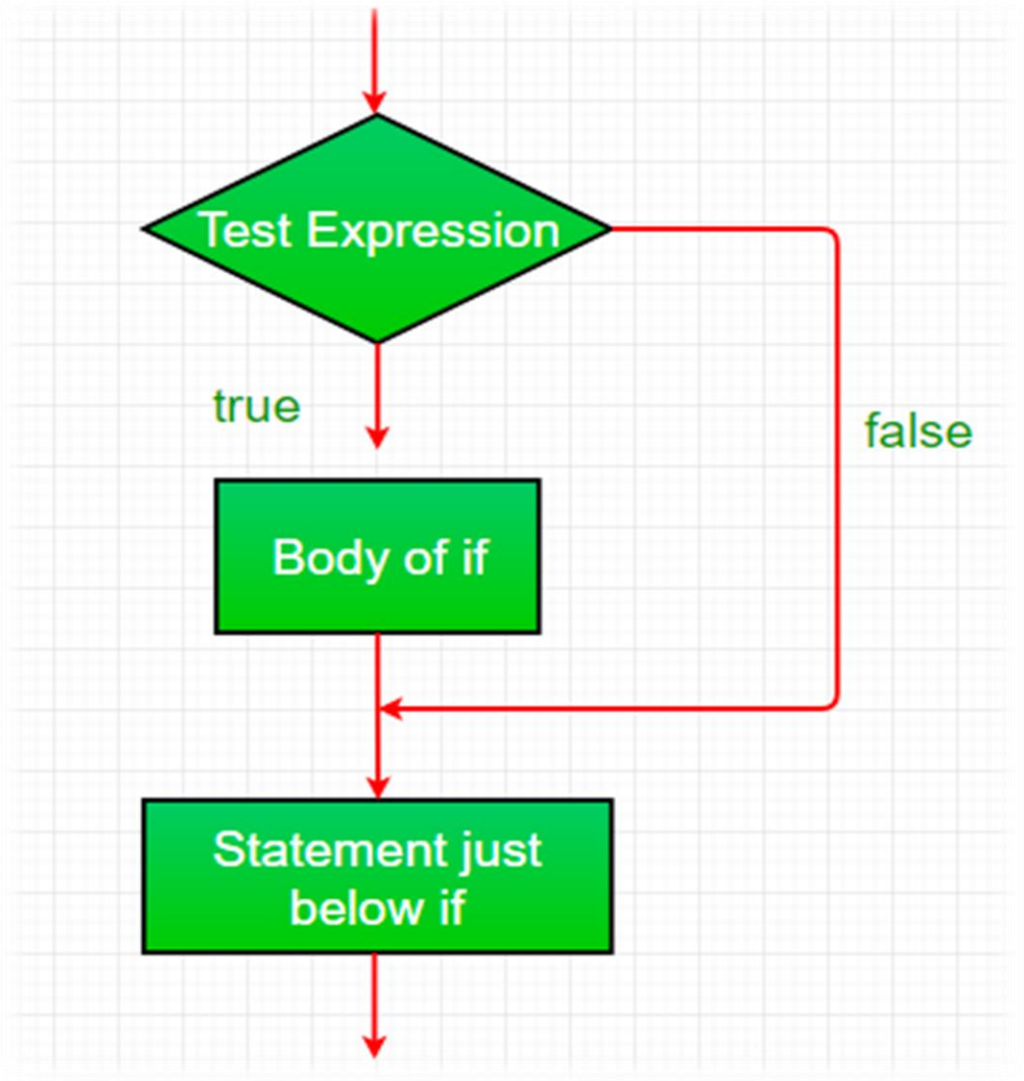
```
import java.util.*;
```

```
Scanner entrada = new Scanner(System.in);
    int x = entrada.nextInt();
    double y = entrada.nextDouble();
    String z = entrada.next();
    boolean b = entrada.nextBoolean();
```

Entrada em Console

```java
import java.util.Scanner;

public class Revisao01 {
    public static void main(String[] args) {

        String nome = "";
        Scanner LerTeclado = new Scanner(System.in);

            System.out.println("É hora da revisão!!!!!");
            System.out.print("Qual é o seu nome?: ");
            nome = LerTeclado.nextLine();
            System.out.println("Olá "+nome);

    }
```

ESTRUTURAS DE CONTROLE



ESTRUTURAS DE CONTROLE

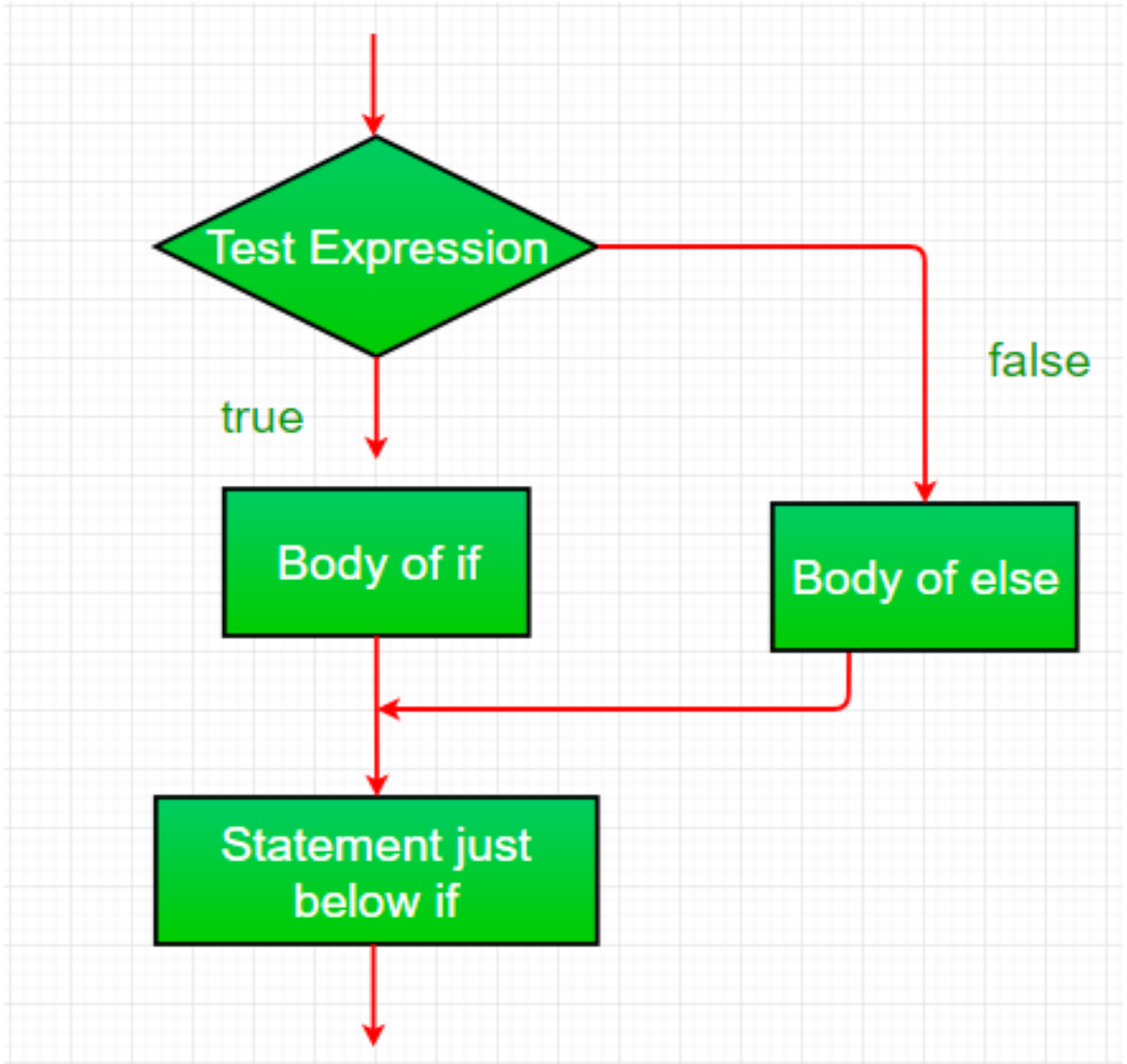
Estrutura If

```
if ( condição )
{
      conjunto de instruções 1
}
else
{
      conjunto de instruções 2
}
```

ESTRUTURAS DE CONTROLE

```
class AulaDiogenes
{
    public static void main(String args[])
    {
        int x = 10;


        if (x > 15) {
            System.out.println("10 menor que 15");
        }
      System.out.println("Não entrou no IF");
    }
}
```

ESTRUTURAS DE CONTROLE



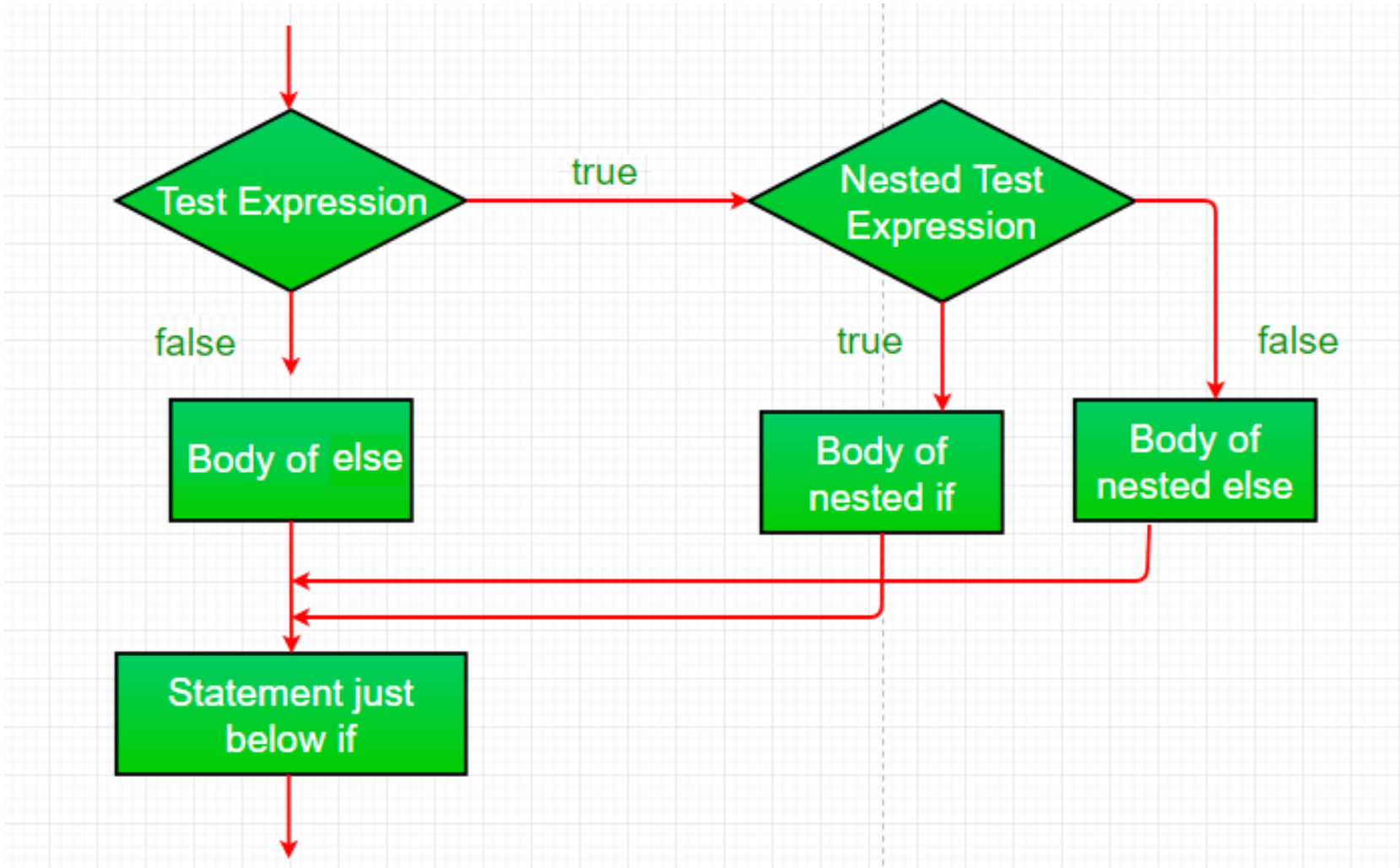
ESTRUTURAS DE CONTROLE

```
class AulaDiogenes
{
    public static void main(String args[])
    {
        int x = 10;

        if (x < 15)
            System.out.println("O x é menor que 15");
        else
            System.out.println("O x é maior que 15");
    }
}
```

ESTRUTURAS DE CONTROLE

```
public class Main {
  public static void main(String[] args) {
    int tempo = 20;
    if (tempo < 18) {
      System.out.println("Bom dia.");
    } else {
      System.out.println("Boa noite.");
    }
  }
}
```

ESTRUTURAS DE CONTROLE



ESTRUTURAS DE CONTROLE

```java
class AulaDiogenes
{
    public static void main(String args[])
    {
        int x = 10;

        if (x == 10)
        {
            //Entrou no primeiro IF
            if (x < 15){
                System.out.println("X é menor que 15");
            }
            // Só será executado se a instrução acima
            // é verdade
            if (x < 12)
                System.out.println("X é menor que 12");
            else
                System.out.println("X é maior que 15");
        }
    }
}
```

ESTRUTURAS DE CONTROLE

```
public class Main {
  public static void main(String[] args) {
    int tempo = 22;
    if (tempo < 10) {
      System.out.println("Bom dia.");
    } else if (tempo  < 20) {
      System.out.println("Boa tarde.");
    }  else {
      System.out.println("Boa noite.");
    }
  }
}
```

ESTRUTURAS DE CONTROLE

Operador ternário

ternário é uma versão abreviada da instrução if-else. Possui três operandos e daí o nome ternário. O formato geral é:

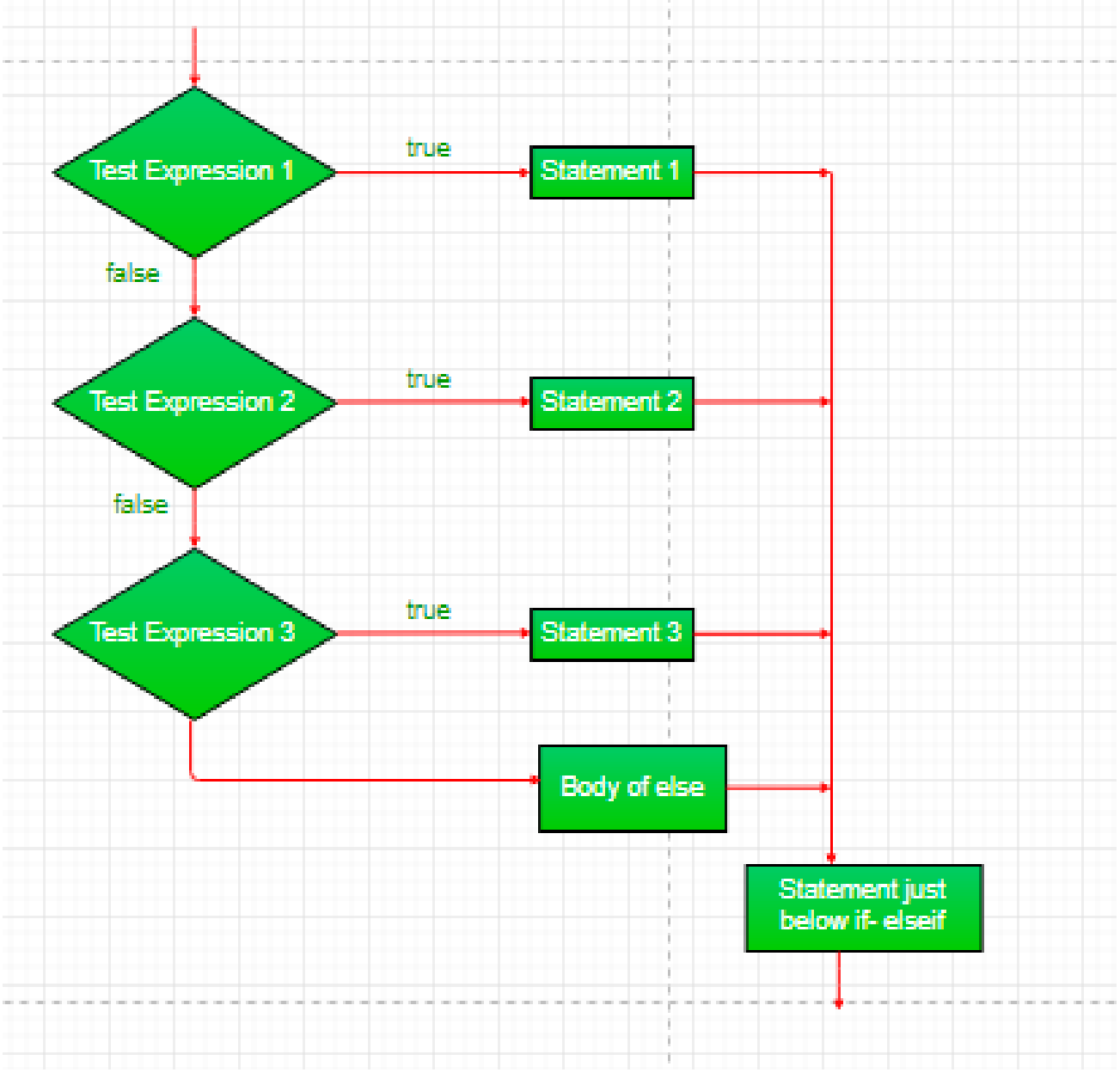
condição ? **se** verdadeiro : **se** falso

ESTRUTURAS DE CONTROLE

Operador ternário

```
public class Revisao01 {
    public static void main(String[] args) {
        //Ternario
        int a = 30, b = 10, c = 50, resultado;
        resultado = ((a > b)
                    ? (a > c)
                              ? a
                              : c
                    : (b > c)
                              ? b
                              : c);
        System.out.println("Máximo de três números  = "+ resultado);
        }
    }
```

ESTRUTURAS DE CONTROLE



ESTRUTURAS DE CONTROLE

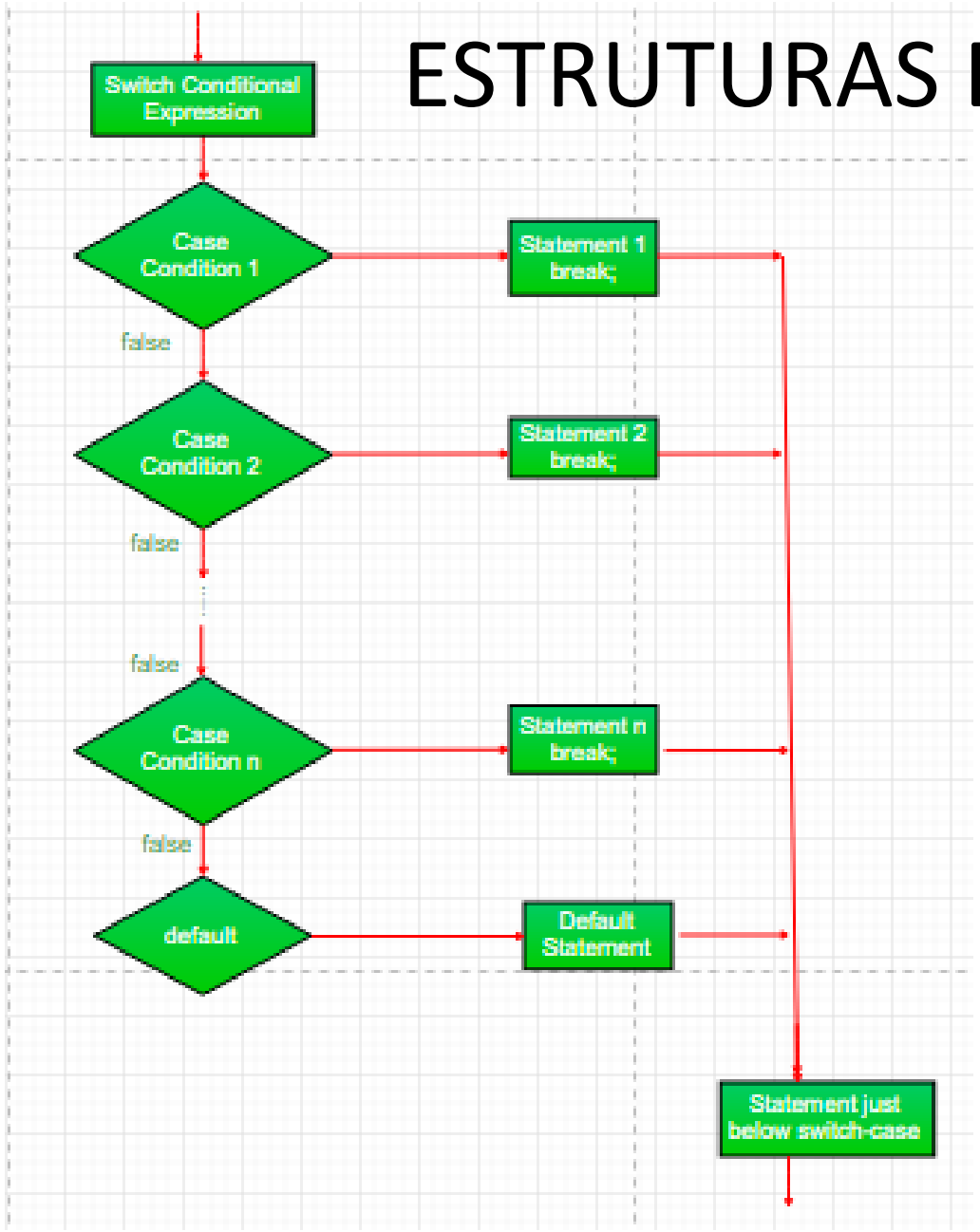
```
class AulaDiogenes
{
    public static void main(String args[])
    {
        int x = 20;

        if (x == 10)
            System.out.println("X é igual a 10");
        else if (x == 15)
            System.out.println("X é igual a 15");
        else if (x == 20)
            System.out.println("X é igual a 20");
        else
            System.out.println("O valor de X não existe na
condicao");
    }
}
```

ESTRUTURAS DE CONTROLE

Estrutura switch

```
switch ( expressão )
{
      case valor1:
            conjunto de instruções 1;
            break;

      case valor2:

             conjunto de instruções 1;
             break;

        ...

        default:
            conjunto de instruções n;
  }
```

ESTRUTURAS DE CONTROLE



```java
public class Main {
  public static void main(String[] args) {
    int dia = 5;
    switch (dia) {
      case 1:
        System.out.println("Segunda-feira");
        break;
      case 2:
        System.out.println("Terça-feira");
        break;
      case 3:
        System.out.println("Quarta-feira");
        break;
      case 4:
        System.out.println("Quinta-feira");
        break;
      case 5:
        System.out.println("Sexta-feira");
        break;
      case 6:
        System.out.println("Sabado");
        break;
      case 7:
        System.out.println("Domingo");
        break;
    }
  }
}
```

ESTRUTURAS DE CONTROLE

ESTRUTURAS DE CONTROLE

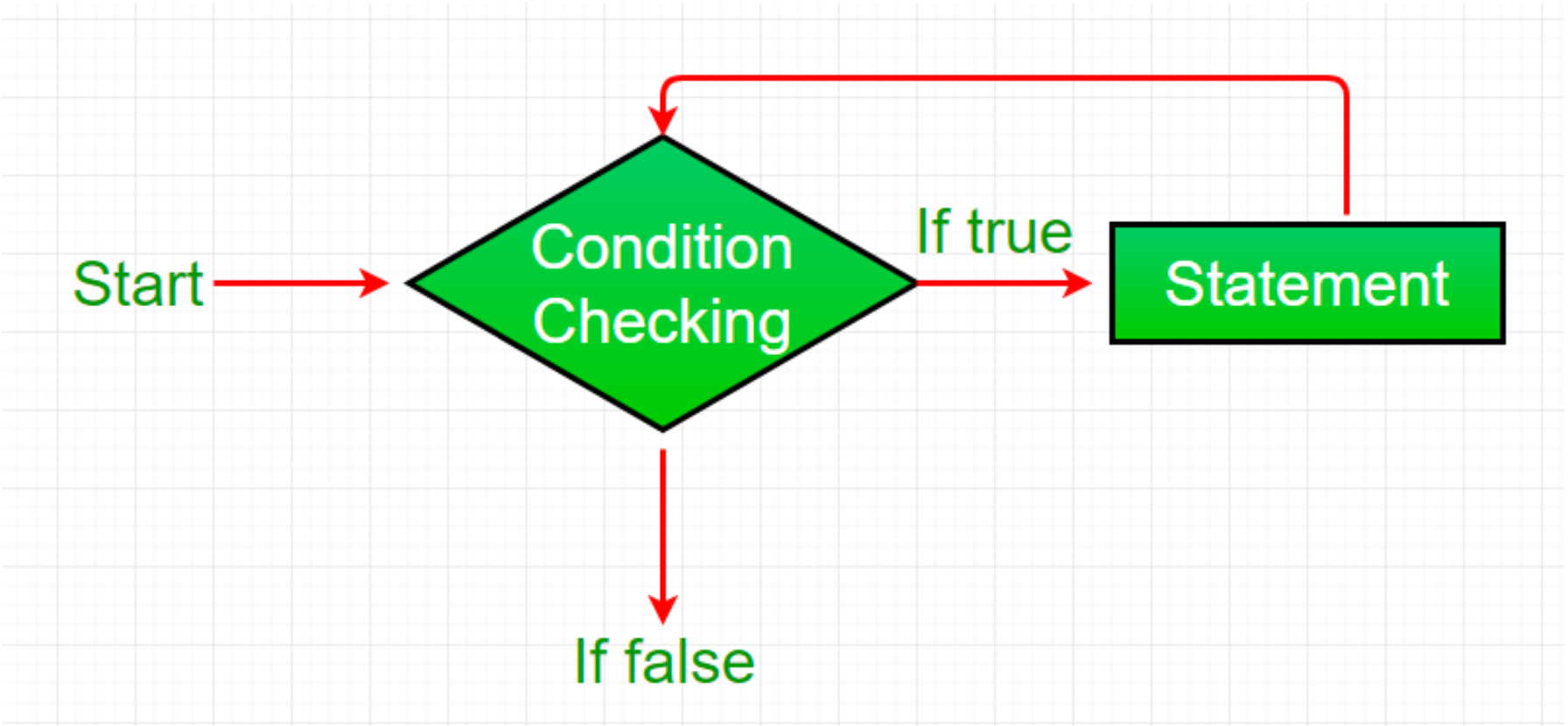
```
public class Main {
  public static void main(String[] args) {
    int dia = 4;
    switch (dia) {
      case 6:
        System.out.println("Hoje é sábado ");
        break;
      case 7:
        System.out.println("Hoje é domingo");
        break;
      default:
        System.out.println("Vamos chegou o fim de semana ");
    }
  }
}
```

ESTRUTURAS DE CONTROLE

Estrutura while

```
while ( condição )
{
      conjunto de instruções;
}
```

ESTRUTURAS DE CONTROLE



ESTRUTURAS DE CONTROLE

```
class AulaDiogenes
{
    public static void main(String args[])
    {
        int z = 1;
        while (z <= 4)
        {
            System.out.println("O valor de Z:" + z);
            // Vamos fazer o incremento do laço
            // Para a proxima interação
            z++;
        }
    }
}
```

ESTRUTURAS DE CONTROLE

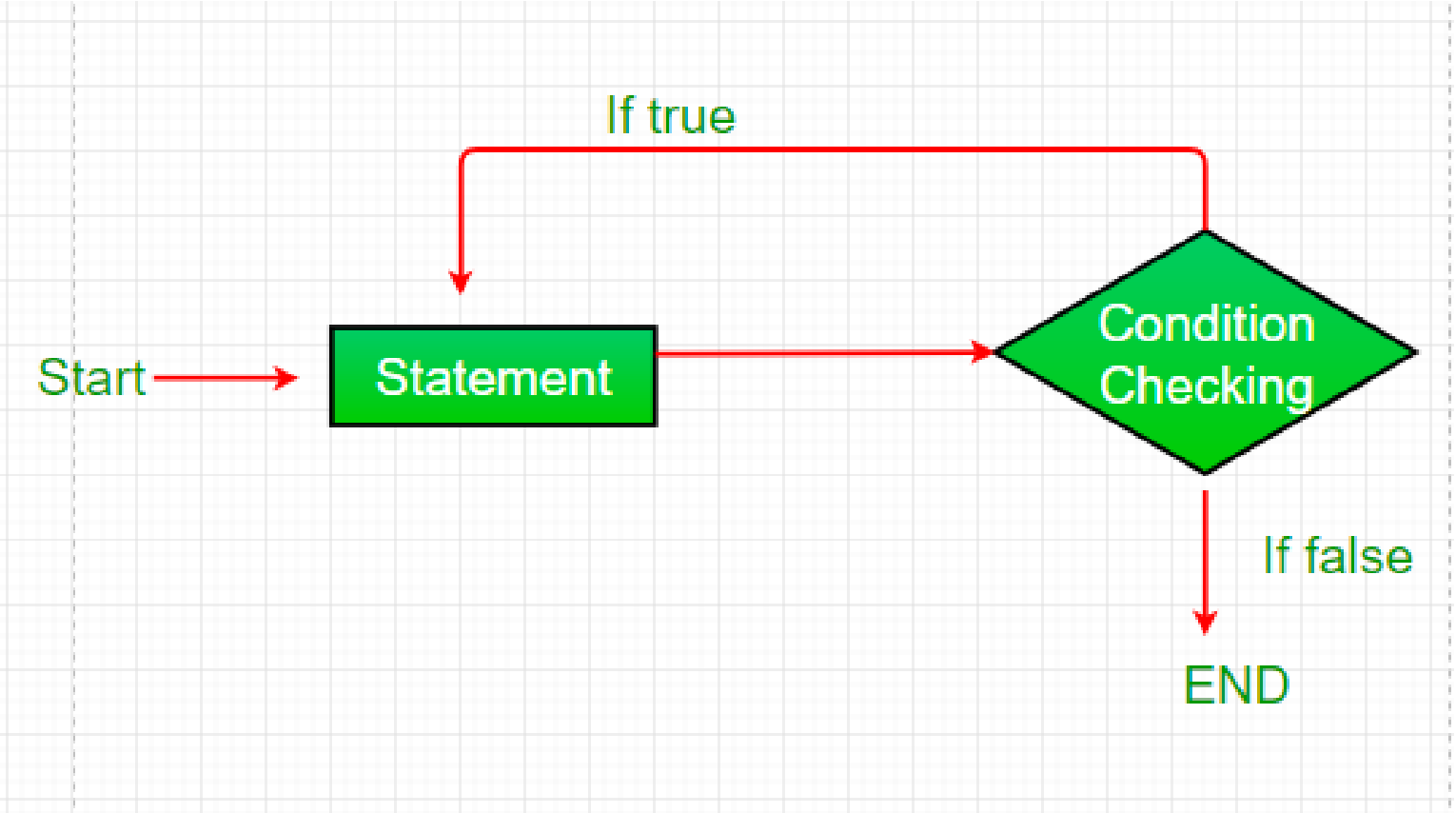
```
public class Main {
  public static void main(String[] args) {
    int x = 0;
    while (x < 5) {
      System.out.println(x);
      x++;
    }
  }
}
```

ESTRUTURAS DE CONTROLE

Estrutura do...while

```
do
{
        conjunto de instruções;
}while ( condição );
```

ESTRUTURAS DE CONTROLE



ESTRUTURAS DE CONTROLE

```
class AulaDiogenes
{
    public static void main(String args[])
    {
        int z = 21;
        do
        {
            //Vai ser executado 1 vez pelomenos
            System.out.println("O valor de Z:" + z);
            z++;
        }
        while (z < 20);
    }
}
```

ESTRUTURAS DE CONTROLE

```
public class Main {
  public static void main(String[] args) {
    int x = 0;
    do {
      System.out.println(x);
      x++;
    }
    while (x < 5);
  }
}
```

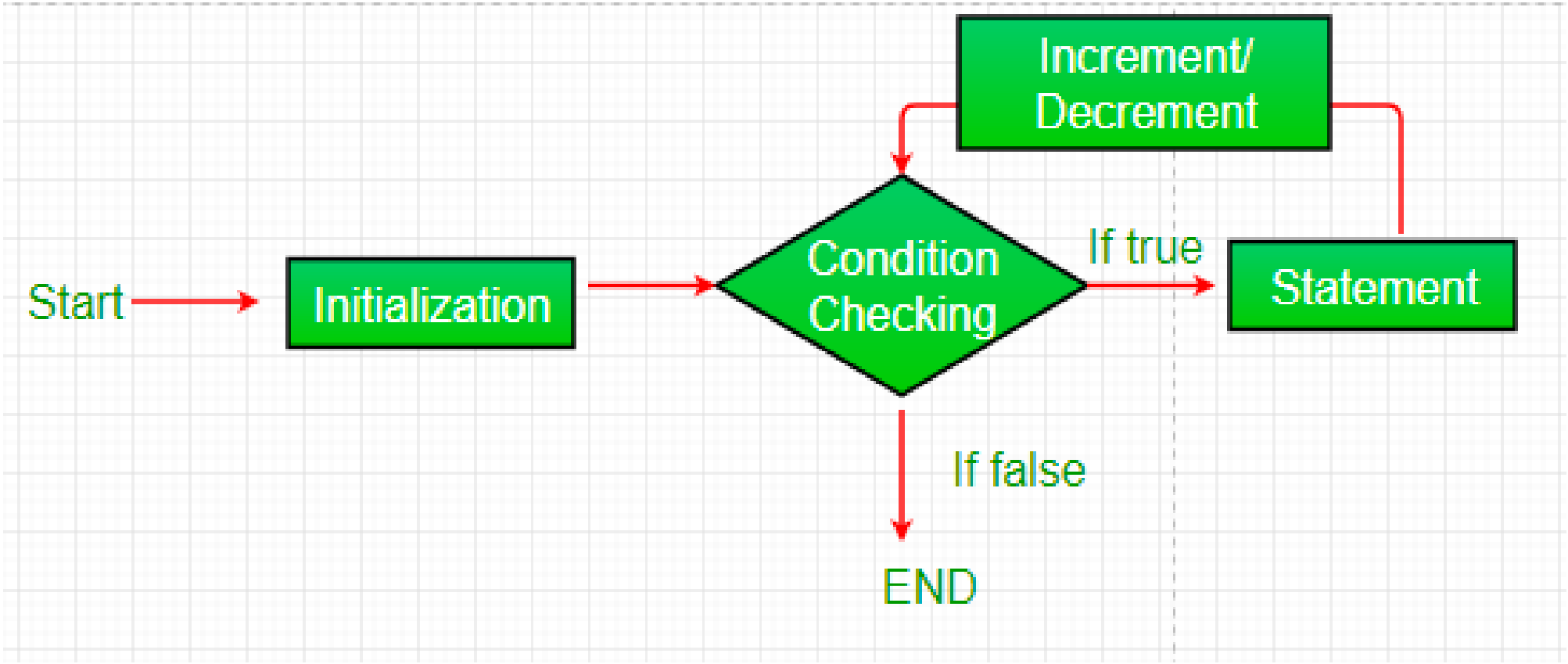
ESTRUTURAS DE CONTROLE

Estrutura for

```
for(início; condição; fim)
{
      conjunto de instruções;
}
```

- **início** : Executado antes de iniciar a repetição do laço;
- **condição** : O laço repete enquanto ela for verdadeira;
- **fim** : Executado após cada repetição.

ESTRUTURAS DE CONTROLE



ESTRUTURAS DE CONTROLE

```
class AulaDiogenes
{
    public static void main(String args[])
    {
        // Vamos começar com a inicialização de y=2
        // e a logica de parada é y <=4
        for (int y = 2; y <= 4; y++)
            System.out.println("O valor de Y:" + y);
    }
}
```

ESTRUTURAS DE CONTROLE

```
public class Main {
  public static void main(String[] args) {
    for (int x = 0; x < 5; x++) {
      System.out.println(x);
    }
  }
}
```

ESTRUTURAS DE CONTROLE

```
public class Main {
  public static void main(String[] args) {
    for (int x = 0; x < 10; i++) {
      if (x == 4) {
        break;
      }
      System.out.println(i);
    }
  }
}
```

ESTRUTURAS DE CONTROLE

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 5) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

```java
public class AulaDiogenes
{
    public static void main(String args[])
    {
    String conjunto[] = {"Diogenes", "Carvalho", "Matias"};

        //Nosso laço
        for (String x : conjunto)
        {
            System.out.println(x);
        }

        /* O mesmo resultado acima observe a diferença
        for (int x = 0; x < conjunto.length; x++)
        {
            System.out.println(conjunto[x]);
        }
        */
    }
}
```

ESTRUTURAS DE CONTROLE

Programação Orientada a Objetos

Abstração

A abstração consiste em um dos pontos mais importantes dentro de qualquer linguagem Orientada a Objetos.

Como estamos lidando com uma representação de um objeto real (o que dá nome ao paradigma), temos que imaginar o que esse objeto irá realizar dentro do Sistema.

Programação Orientada a Objetos

Classe

A palavra classe vem da taxonomia da biologia. Todos os seres vivos de uma mesma classe biológica têm uma série de atributos e comportamentos em comum, mas não são iguais, podem variar nos valores desses atributos e como realizam esses comportamentos.

Programação Orientada a Objetos

Classe

Uma classe é uma forma de definir um tipo de dado em uma linguagem orientada a objeto. Ela é formada por dados e comportamentos.

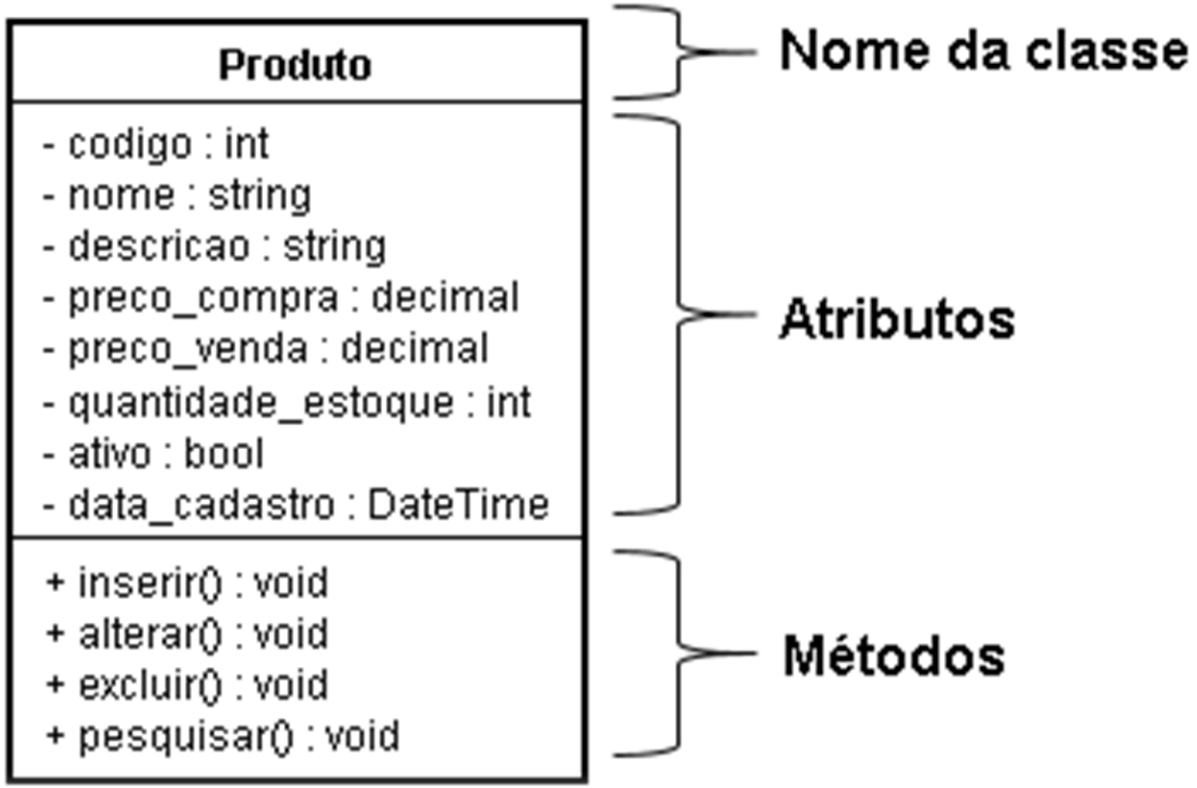
Programação Orientada a Objetos

Classe

Para definir os dados são utilizados os atributos, e para definir o comportamento são utilizados métodos. Depois que uma classe é definida podem ser criados diferentes objetos que utilizam a classe.

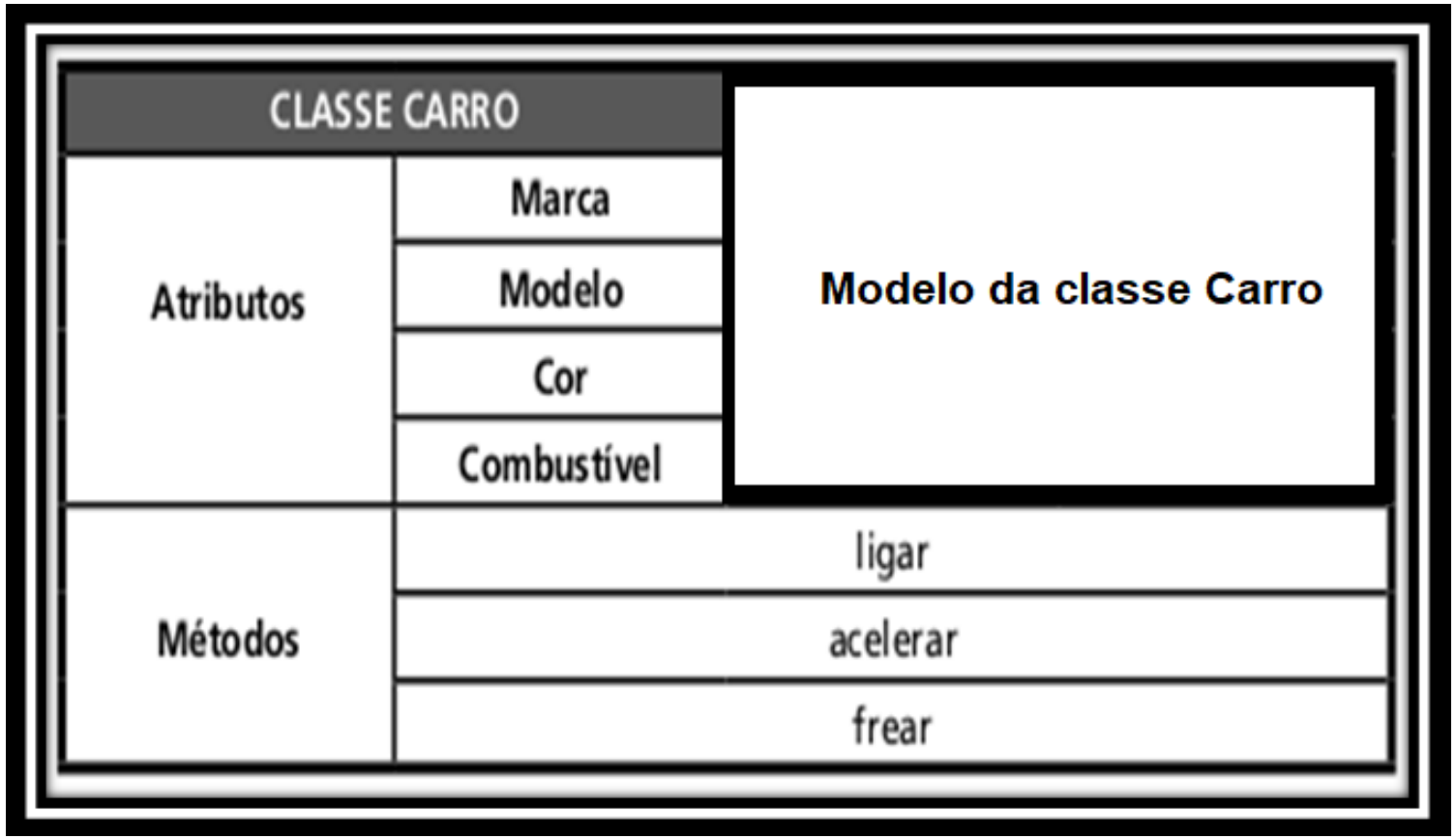
Programação Orientada a Objetos

Classe



Programação Orientada a Objetos

Classe



Programação Orientada a Objetos

Classe

Programação Orientada a Objetos

Classe

class **Casa**{

}

Programação Orientada a Objetos

Atributos

Um atributo consiste em um dado ou informação de estado, para o qual cada tem seu próprio valor. Existem dois tipos de atributos em um sistema orientado a objetos: os atributos de objetos e os atributos de classes.

```
class Casa{
        private int tamanho;
        private  String  endereco;
        private  int quantidade_quartos;
}
```

Programação Orientada a Objetos

Métodos

São similares a procedimentos e funções e consistem nas descrições das operações ou seja ações a ser executa.

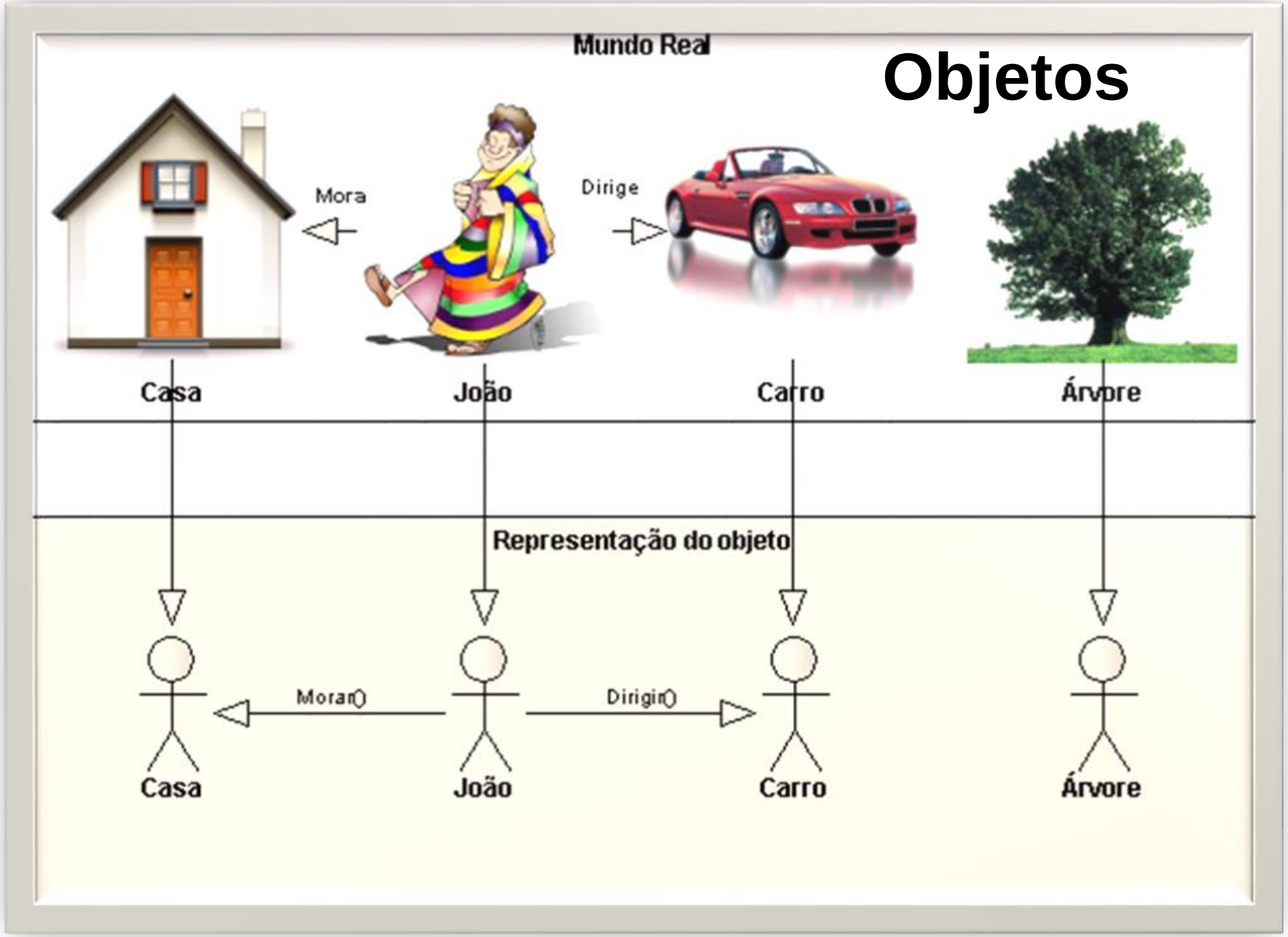
```
class Casa{
        private int tamanho;
        private  String  endereco;
        private  int quantidade_quartos;
        public void acende_luz(){
        …
        }
        public double consume_energia(double conta_luz){
        …
        }
}
```

Programação Orientada a Objetos

Objetos

Refere-se a um modelo materializado de uma classe ,que passa a existir a partir de uma instância da classe.

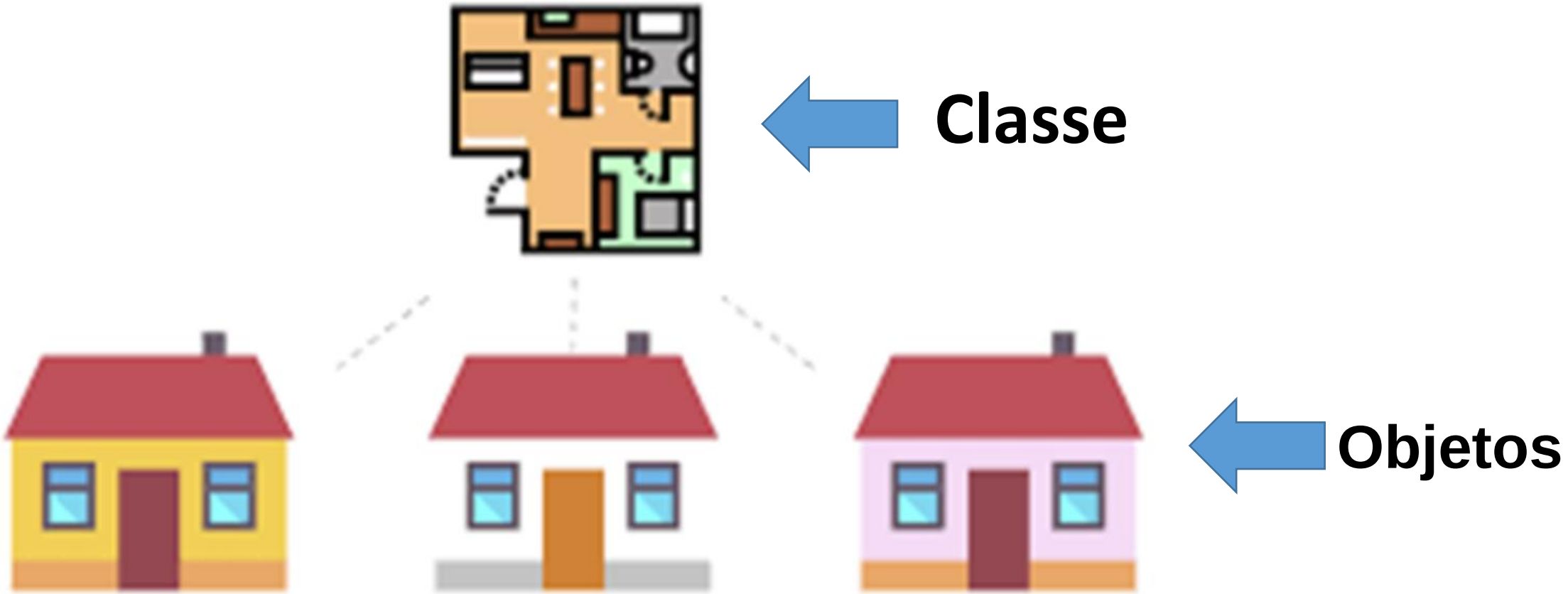
Programação Orientada a Objetos



Programação Orientada a Objetos

CLASSE CARRO		OBJETO CARRO A	OBJETO CARRO B
Atributos de objeto	Marca	Ford	Mitsubishi
	Modelo	Fiesta	L-200
	Cor	branco	azul royal
	Combustível	gasolina	diesel
Métodos	ligar		
	acelerar		
	frear		

Programação Orientada a Objetos



Programação Orientada a Objetos

Classe

```
class Casa{
        private int tamanho;
        private  String  endereco;
        private  int quantidade_quartos;
        public void acende_luz(){
        …
        }
        public double consume_energia(double conta_luz){
        …
        }
}
```

Objetos

```
Casa objCasa_1 = new Casa();
Casa objCasa_2 = new Casa();
Casa objCasa_3 = new Casa();
```

Programação Orientada a Objetos

Obs: arquivo Casa.java

```
class Casa{
   private int tamanho;
   private  String  endereco;
   private  int quantidade_quartos;
   private double consumoLuz;
   public void acende_luz(){
            Syste.out.println(“Luz esta ligada!”)
            }
   public double consume_energia(double conta_luz){
             this.consumoLuz = conta_luz;
            return this.consumoLuz;
   }

}
```

Obs: arquivo Meu_programa_Diogenes.java

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

                        Casa objCasa_1;
                         Casa objCasa_2;
                         Casa objCasa_3 ;

                         objCasa_1 = new Casa();
                         objCasa_2 = new Casa();
                         objCasa_3 = new Casa();

objCasa_1.endereço = “Avenida General Manoel Rabelo”;
objCasa_1.quantidade_quartos = 3;
objCasa_1.tamanho = 120;
objCasa_1. consume_energia(234);

                        }
 }
```

Programação Orientada a Objetos

Construtores

Os construtores são os responsáveis por criar o objeto em memória, ou seja, instanciar a classe que foi definida. Eles são obrigatórios e são declarados.

Programação Orientada a Objetos

Construtores

```
public class Carro{

/* CONSTRUTOR DA CLASSE Carro */
public Carro(){
        //Faça o que desejar
        //na construção do objeto
}

}
```

Programação Orientada a Objetos

Construtores

modificadores de acesso (public nesse caso) + nome da classe (Carro nesse caso) + parâmetros (nenhum definido neste caso).

OBS: O construtor pode ter níveis como: public, private ou protected.

Programação Orientada a Objetos

Construtores

```
public class Carro{

/* CONSTRUTOR DA CLASSE Carro */
public Carro(){
        //Construção do objeto
}

}

public class Aplicacao {

public static void main(String[] args) {
//Chamamos o construtor sem nenhum parâmetro
Carro celta_suv = new Carro();
}

}
```

Programação Orientada a Objetos

Construtores

```
class Conta {
    String titular;
    int numero;
    double saldo;

    //Construtor
    Conta() {
        System.out.println("Construindo uma conta.");
    }

}
```

Programação Orientada a Objetos

Construtores

Por padrão, o Java já cria esse construtor sem parâmetros para todas as classes, então você não precisa fazer isso se utilizará apenas construtores sem parâmetros.
Por outro lado, poderá criar mais de um construtor para uma mesma classe.
Onde, podemos criar um construtor sem parâmetros, com dois parâmetros e outro com três parâmetros.

Programação Orientada a Objetos

Construtores

```
class Conta {
    String titular;
    int numero;
    double saldo;

    //Construtor
    Conta(String titular) {
        this.titular = titular;
    }


}
```

```
public class Carro{

private String cor;
private double preco;
private String modelo;

/* CONSTRUTOR PADRÃO */
public Carro(){

}

/* CONSTRUTOR COM 2 PARÂMETROS */
public Carro(String modelo, double preco){
        //Se for escolhido o construtor sem a COR do veículo
        // definimos a cor padrão como sendo Azul
this.cor = “Azul”;
this.modelo = modelo;
this.preco = preco;
}

/* CONSTRUTOR COM 3 PARÂMETROS */
public Carro(String cor, String modelo, double preco){
this.cor = cor;
this.modelo = modelo;
this.preco = preco;
}

}
```

Programação Orientada a Objetos

Construtores

Programação Orientada a Objetos

Construtores

```
public class AplicacaoPrincipal {


public static void main(String[] args) {
String nome = "Diogenes Carvalho Matias";
Conta c = new Conta(nome);
System.out.println(c.titular);
}
```

```
public class Carro{

private String cor;
private double preco;
private String modelo;

/* CONSTRUTOR PADRÃO */
public Carro(){

}

/* CONSTRUTOR COM 2 PARÂMETROS */
public Carro(String modelo, double preco){
        //Se for escolhido o construtor sem a COR do veículo
        // definimos a cor padrão como sendo Azul
this.cor = “Azul”;
this.modelo = modelo;
this.preco = preco;
}

/* CONSTRUTOR COM 3 PARÂMETROS */
public Carro(String cor, String modelo, double preco){
this.cor = cor;
this.modelo = modelo;
this.preco = preco;
}

}
```

Programação Orientada a Objetos

Construtores

Agora três construtores padrões, ou seja, podemos criar um novo Carro sem definir sua Cor, ou podemos criar um novo carro definindo todos os seus atributos.

Programação Orientada a Objetos

Construtores

OBS: Podemos criar vários carros com diferentes construtores.

```
public class AplicacaoPrincipal {


public static void main(String[] args) {
//Construtor sem parâmetros
Carro prototipoDeCarro = new Carro();

//Construtor com 2 parâmetros
Carro celtaPreto = new Carro("New Celta","54000");

//Construtor com 3 parâmetros
Carro fuscaAmarelo = new Carro("Fusca", "88000");
}

}
```

Programação Orientada a Objetos

Destrutores

Não existe o conceito de destrutores em Java, mas você nem precisa se preocupar com isto, pois não tem como literalmente destruir um objeto, assim como você faz em C/C++. Isso porque não é garantido que o **Garbage Collection** irá destruir este objeto, já que ele o faz na hora que achar conveniente e o programador não tem nenhum controle sobre isso.

A forma mais adequada de “tentar” destruir um objeto em Java é atribuir valores nulos a ele. Assim, quando o **Garbage Collection** for realizar o seu trabalho, verá que seu objeto não está sendo mais utilizado e o destruirá.

Programação Orientada a Objetos

Vamos analisar o seguinte cenário

Um banco, é bem fácil perceber que uma entidade extremamente importante para o nosso sistema é a conta e o que toda conta tem e é importante para nós é?

- número da conta
- nome do titular da conta
- saldo

Programação Orientada a Objetos

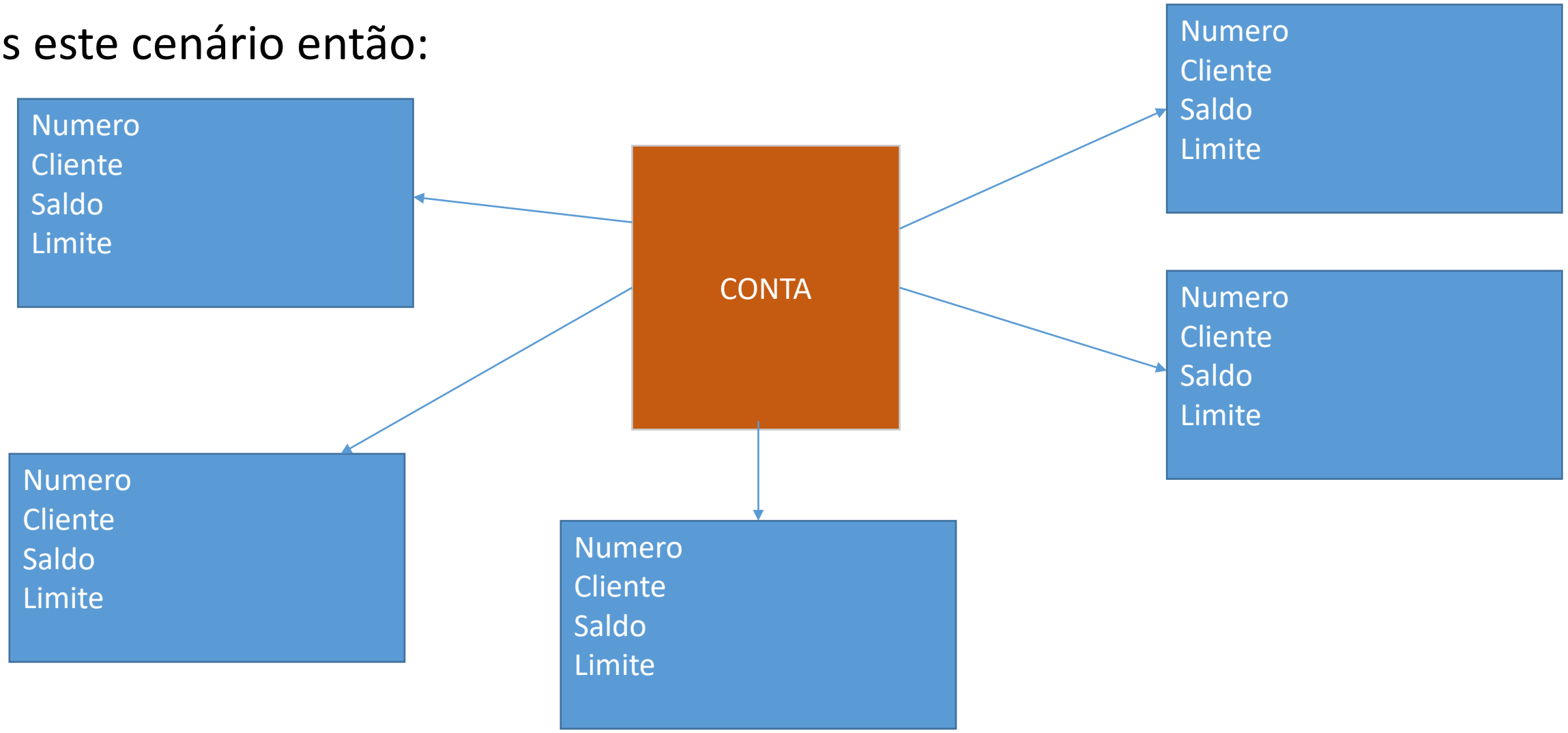
Vamos analisar o seguinte cenário

O que gostaríamos de "pedir à conta“ quando nós solicitamos em um sistema de banco?

- Saca uma quantidade X
- Deposita uma quantidade X
- Imprime o nome do titular da conta
- Ver o saldo atual
- Transfere um dinheiro de X para uma outra conta Y
- Ver o tipo de conta

Programação Orientada a Objetos

Temos este cenário então:



Programação Orientada a Objetos

OBJETOS SÃO ACESSADOS POR REFERÊNCIAS

Quando declaramos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma maneira de acessá-lo, chamada de **referência**.

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

        Conta conta1;
        conta1 = new Conta();
        Conta conta2;
        conta2 = new Conta();

}
```

Programação Orientada a Objetos

OBJETOS SÃO ACESSADOS POR REFERÊNCIAS

O que temos então é “Tenho uma referência **conta1** a um objeto do tipo **Conta**”.

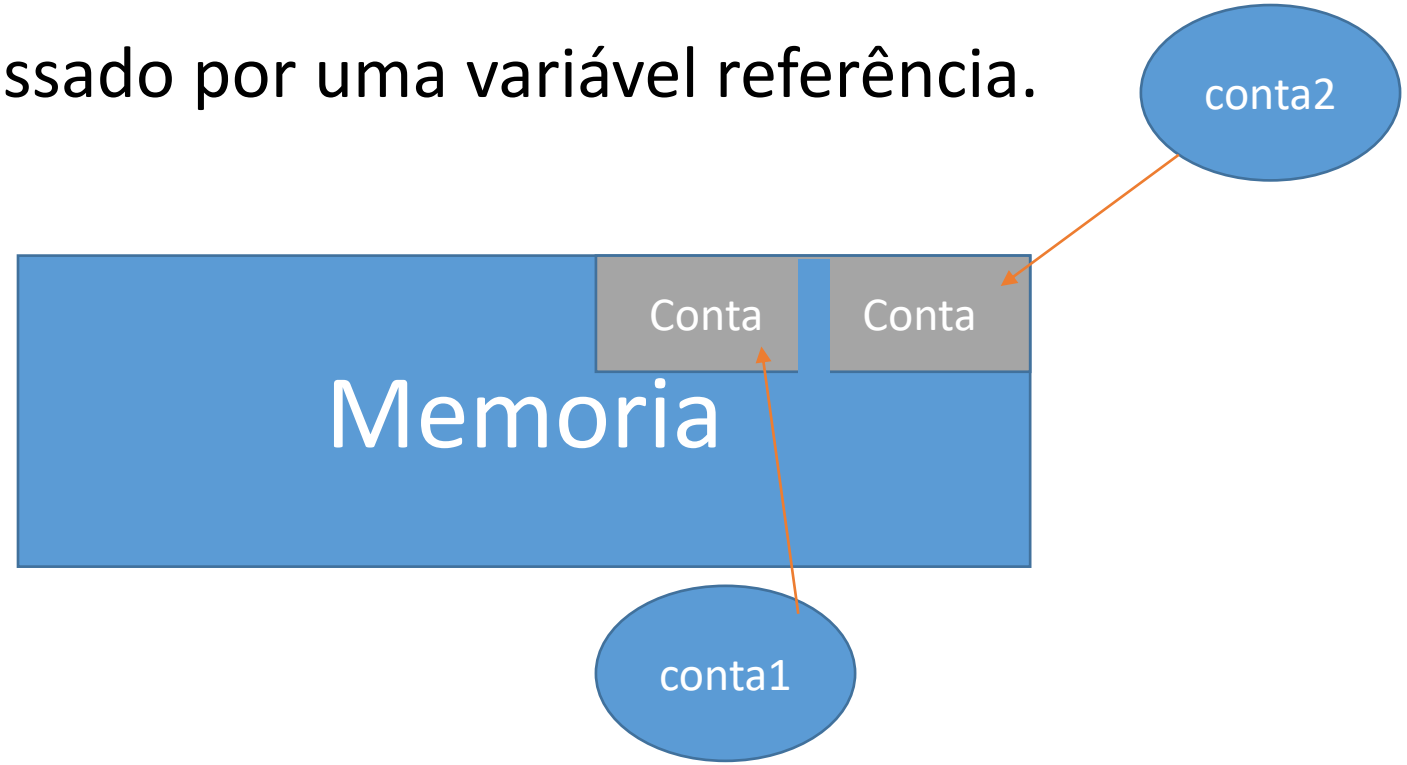
```
public static void main(String[] args) {

        Conta conta1;
        conta1 = new Conta();
        Conta conta2;
        conta2 = new Conta();

}
```

Programação Orientada a Objetos

OBJETOS SÃO ACESSADOS POR REFERÊNCIAS

Ou seja todo objeto Java é acessado por uma variável referência.

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

        Conta conta1;
        conta1 = new Conta();
        Conta conta2;
        conta2 = new Conta();

}
```



Programação Orientada a Objetos

OBJETOS SÃO ACESSADOS POR REFERÊNCIAS

Ou seja todo objeto Java é acessado por uma variável referência.

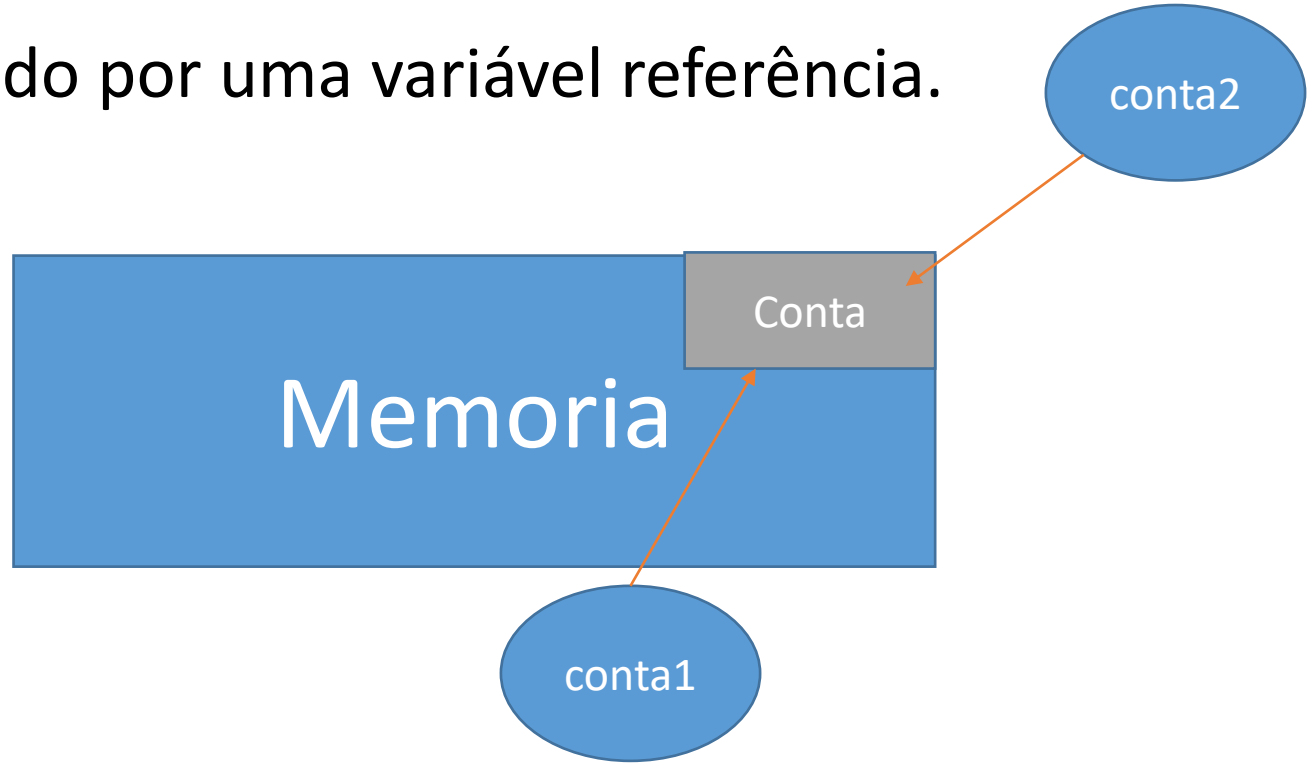
```
public static void main(String[] args) {

Conta conta1 = new Conta();
        conta1.deposita(100);
        Conta conta2 = conta1;
        conta2.deposita(200);
System.out.println(conta1.saldo);
System.out.println(conta2.saldo);
}
```

Programação Orientada a Objetos

OBJETOS SÃO ACESSADOS POR REFERÊNCIAS

Ou seja todo objeto Java é acessado por uma variável referência.

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

Conta conta1 = new Conta();
        conta1.deposita(100);
        Conta conta2 = conta1;
        conta2.deposita(200);
System.out.println(conta1.saldo);
System.out.println(conta2.saldo);
}
```



Programação Orientada a Objetos

Vamos analisar o seguinte cenário

O que gostaríamos de "pedir à conta“ quando nós solicitamos em um sistema de banco?

- Transfere um dinheiro de X para uma outra conta Y : poderíamos verificar se a conta possui a quantidade a ser transferida disponível.

Programação Orientada a Objetos

Analisando nossa classe Conta

Vamos aumentar nossa classe Conta e adicionar **nome**, **sobrenome** e **cpf** do titular da conta, então vamos criar a nossa classe ***cliente*** que será uma composição da nossa classe ***conta***.

```
public static void main(String[] args) {
        Conta minhaConta = new Conta();
        Cliente conta = new Cliente();
        minhaConta.titular = conta;
}
```

Programação Orientada a Objetos

Analisando nossa classe Conta

Podemos realmente navegar sobre toda essa estrutura de informação, sempre usando o “**ponto**”:

```
public static void main(String[] args) {
        Conta minhaConta = new Conta();
        minhaConta.titular.nome = "Diogenes";
}
```

Programação Orientada a Objetos

Encapsulamento

“Objetos do mundo real encapsulam em si os próprios atributos, quer sejam descritivos, partes componentes ou funções.” (Meyer, 1997)

Programação Orientada a Objetos

Encapsulamento

É o mecanismo que une o código e os dados que ele manipula. Outra forma de pensar sobre o encapsulamento é que ele é um escudo protetor que evita que os dados sejam acessados pelo código fora desse escudo.

Programação Orientada a Objetos

Encapsulamento

Tecnicamente no encapsulamento, as variáveis ou dados de uma classe são ocultados de qualquer outra classe e podem ser acessados somente por meio de qualquer função de membro de sua própria classe na qual é declarada.

Programação Orientada a Objetos

Encapsulamento

Como no encapsulamento, os dados em uma classe são ocultados de outras classes usando o conceito de ocultação de dados que é obtido tornando os membros ou métodos de uma classe privados, e a classe é exposta ao usuário final ou ao mundo sem fornecer quaisquer detalhes por trás da implementação usando o conceito de abstração, por isso também é conhecido como uma **combinação de ocultação de dados e abstração** .

.

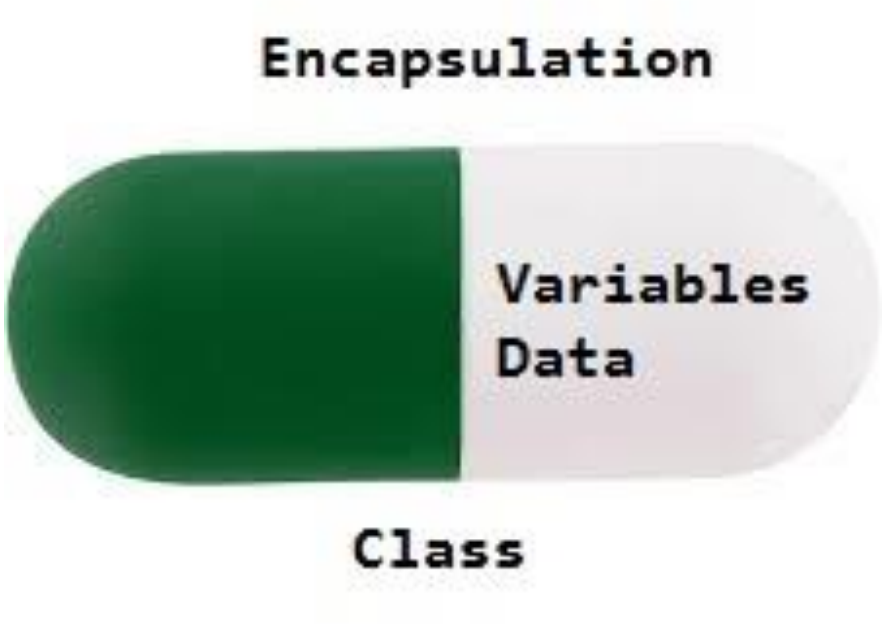
Programação Orientada a Objetos

Encapsulamento

- O encapsulamento pode ser alcançado declarando todas as variáveis na classe como privadas e escrevendo métodos públicos na classe para definir e obter os valores das variáveis.

Programação Orientada a Objetos

Encapsulamento



UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Encapsulamento

```
public class Cliente {

    private String  nome;
    private String  endereco;
    private String  cpf;
    private int idade;

    public  void    mudaCPF(String  cpf) {
        if  (this.idade <=  60) {
                        validaCPF(cpf);
        }
        this.cpf    =   cpf;
    }
    private void    validaCPF(String    cpf) {
        //regras de negocio para podermos validar o CPF...
    }
}
```

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Encapsulamento

Níveis de Acesso:

- Privada: não visível a nenhuma classe externa; visível apenas dentro da classe.

- Pública: completamente visível internamente e para outras classes e elementos externos.

- Protegida: não visível a classes e elementos externos; visível apenas dentro da classe e para seus herdeiros.

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Encapsulamento

Níveis de Acesso:

Atributos Privados e Métodos Públicos:

- Na Programação orientada a objetos uma classe possui, em geral, os atributos privados e os métodos podem ser públicos, tornando o objeto como uma caixa preta onde só aparece o suficiente para que o programa possa utilizá-lo.

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Encapsulamento Níveis de Acesso

Atributos Privados e Métodos Públicos:

- Privada (private): visível exclusivamente dentro da classe.

- Pública (public): completamente visível dentro e fora da classe.

- Protegida (protected): visível apenas dentro da classe, pelos seus herdeiros e pelas classes no mesmo pacote.

- Pacote (padrão): visível apenas dentro da classe e pelas classes que estão no mesmo pacote.

Programação Orientada a Objetos

Encapsulamento Níveis de Acesso

Atributo final :

- Atributo que recebe o prefixo final;
- Só pode receber o valor uma vez na declaração ou no construtor;
- Depois que recebe o valor não pode ser modificado;
- Usualmente associado ao ***static*** para a criação de uma constante;

Programação Orientada a Objetos

Encapsulamento Níveis de Acesso

GETTERS E SETTERS

- Precisamos então arranjar uma maneira de fazer esse acesso. Sempre que precisamos arrumar uma maneira de fazer alguma coisa com um objeto, utilizamos de métodos!

Programação Orientada a Objetos

Encapsulamento Níveis de Acesso

GETTERS E SETTERS

```
public class Conta2 {

    private double  saldo;

    public  double  pegaSaldo() {
        return this.saldo;
    }


}
```

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Encapsulamento Níveis de Acesso

GETTERS E SETTERS

```
public class Conta2 {
    private String  titular;
    private double  saldo;
    private double  limite;
    public  double  getSaldo() {
        return this.saldo + this.limite;
    }
    public String getTitular() {
        return this.titular;
    }
    public void setTitular(String titular) {
        this.titular = titular;
    }
    void deposita(double quantidade){
        this.saldo  +=  quantidade;
    }
}
```

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Encapsulamento Níveis de Acesso

GETTERS E SETTERS

```
public class PrincipalConta {

    public static void main(String[] args) {


        Conta2  minhaConta  =   new Conta2();
        minhaConta.deposita(1000);
        System.out.println("Saldo:" + minhaConta.getSaldo());

    }
```

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Encapsulamento Níveis de Acesso

```java
public class Pessoa {
    private String nome;
    private int idade;
    private String endereco;
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    public String getEndereco() {
        return endereco;
    }
    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }
}
```

GETTERS E SETTERS

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Encapsulamento Níveis de Acesso

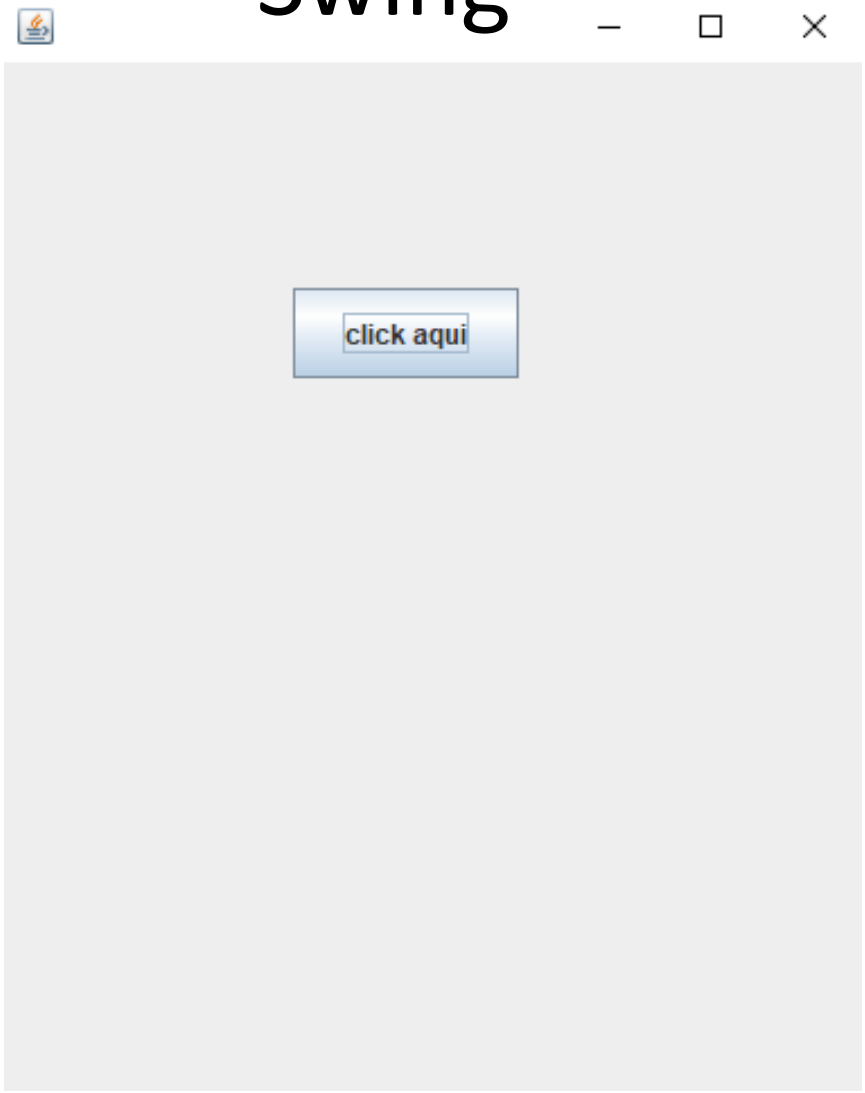
GETTERS E SETTERS

```
public class PrincipalConta {

    public static void main(String[] args) {


        Pessoa objPessoa = new Pessoa();
        objPessoa.setNome("Diogenes Carvalho Matias");
        objPessoa.setIdade(35);
        objPessoa.setEndereco("Avenida Gal Manoela Rabelo");

        System.out.println("Meu nome é"+objPessoa.getNome());
        System.out.println("Minha idade é :"+objPessoa.getIdade());
        System.out.println("Meu endereço é"+objPessoa.getEndereco());

    }
```

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Swing



UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Swing

```java
import javax.swing.*;
public class EstudoSwing {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        JFrame janela=new JFrame();//criamos nosso objeto JFrame

        JButton botao01=new JButton("click aqui");//criamos nosso objeto JButton
        botao01.setBounds(130,100,100, 40);//posição :x , posição :y, Largura: width,Altura: height

        janela.add(botao01);//Adicionando nosso botao em nossa janela

        janela.setSize(400,500);//Largura 400 por  500 de altura
        janela.setLayout(null);//sem usar gerenciadores de layout
        janela.setVisible(true);//tornando nossa janela visível

    }

}
```

Programação Orientada a Objetos

Swing

```java
import javax.swing.*;

public class Janela {

    JFrame janela;
    Janela(){
        janela=new JFrame();

    JButton botao=new JButton("click aqui");
    botao.setBounds(130,100,100, 40);

    janela.add(botao);

    janela.setSize(400,500);
    janela.setLayout(null);
    janela.setVisible(true);

    }
}
```

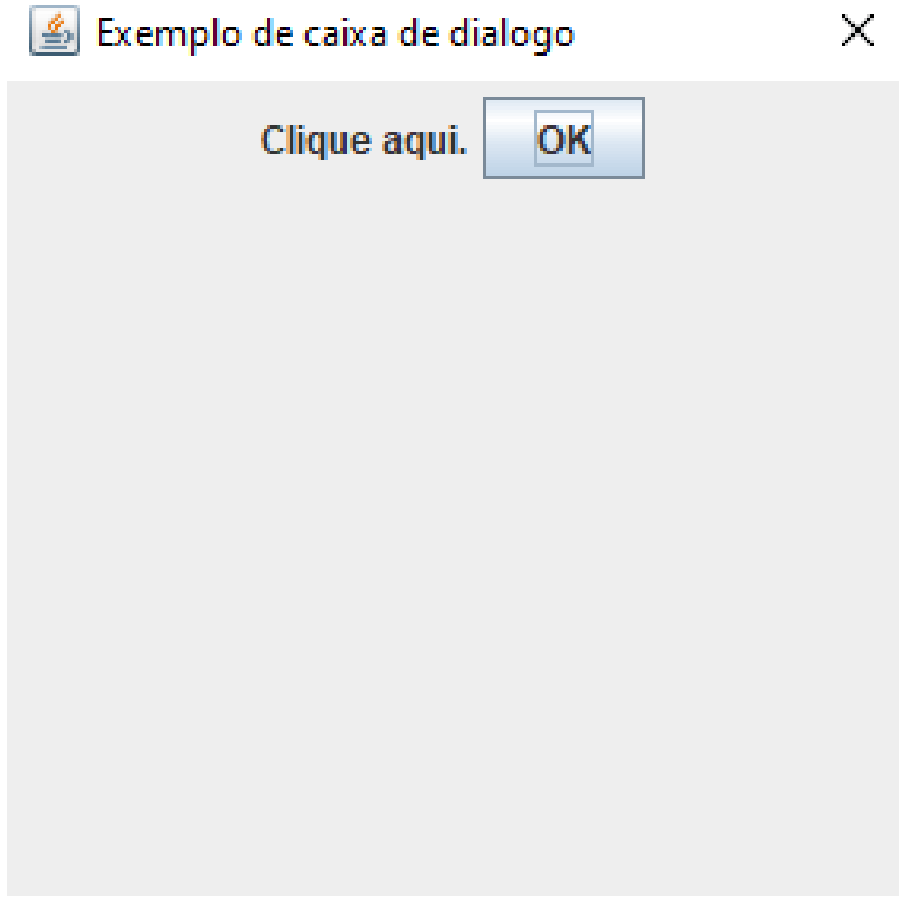
UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos

Swing

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

    new Janela();

}
```

Programação Orientada a Objetos
Swing



Programação Orientada a Objetos
Swing

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class CaixaDialogo {
     private static JDialog dialogo;
     CaixaDialogo() {
            JFrame janela= new JFrame();
            dialogo  = new JDialog(janela , "Exemplo de caixa de dialogo", true);
            dialogo.setLayout( new FlowLayout() );
            JButton botao = new JButton ("OK");
            botao.addActionListener ( new ActionListener()
            {
                public void actionPerformed( ActionEvent e )
                {
                    JOptionPane.showMessageDialog(janela,"Obrigado por continuar estudando.");
                    CaixaDialogo.dialogo.setVisible(false);
                }
            });
            dialogo.add( new JLabel ("Clique aqui."));
            dialogo.add(botao);
            dialogo.setSize(300,300);
            dialogo.setVisible(true);
     }
}
```

UNIFG
LAUREATE INTERNATIONAL UNIVERSITIES

Programação Orientada a Objetos
Swing

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

    new CaixaDialogo();


}
```

Duvidas???