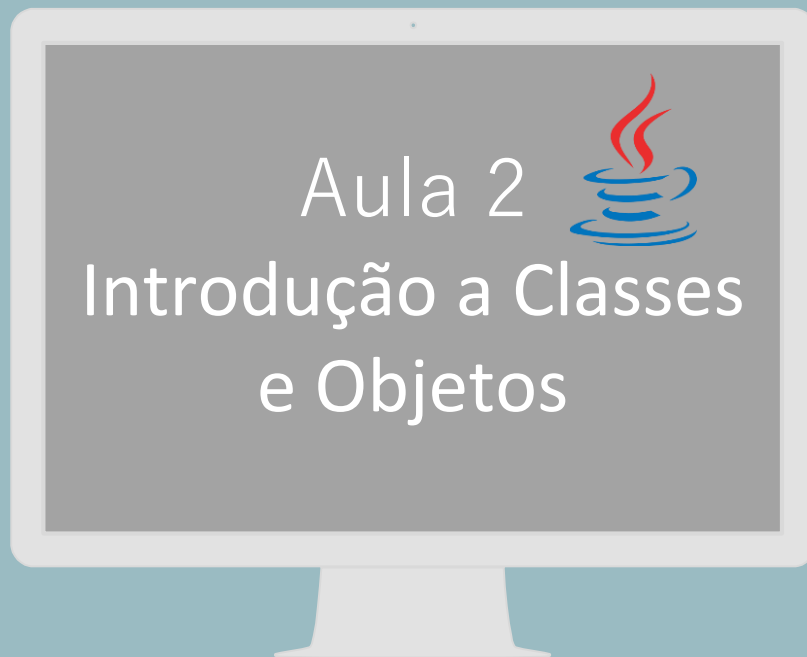




# CURSO DE PROGRAMAÇÃO EM JAVA



1.

# ENTRADA E SAÍDA

**Entrada e Saída de Dados em Java**

# Entrada e Saída 1

```
*Teste.java
1 package Teste;
2 import java.util.*;
3
4 public class Teste {
5     public static void main(String[] args){
6         Scanner teclado = new Scanner(System.in);
7         teclado.useLocale(Locale.ENGLISH);
8
9         //Entrada
10        int idade = teclado.nextInt();
11        float peso = teclado.nextFloat();
12        String nome = teclado.next();
13        teclado.close();
14
15        //Saída
16        System.out.println(nome);
17        System.out.print(nome + " tem " + idade + " anos\n");
18        System.out.printf("O %s pesa %f kilos\n", nome, peso);
19    }
20 }
21
```

# Entrada e Saída 2

```
Teste.java ✕
1 package Teste;
2 import java.util.*;
3
4 public class Teste {
5     public static void main(String[] args){
6         Scanner teclado = new Scanner(System.in);
7         teclado.useLocale(Locale.ENGLISH);
8
9         //Entrada
10        int idade = Integer.parseInt(teclado.nextLine());
11        float peso = Float.parseFloat(teclado.nextLine());
12        String nome = teclado.nextLine();
13        teclado.close();
14
15        //Saída
16        System.out.println(nome);
17        System.out.print(nome + " tem " + idade + " anos\n");
18        System.out.printf("O %s pesa %f kilos\n", nome, peso);
19    }
20 }
```

# Outras classes de Entrada e Saída

- ▷ Scanner;
- ▷ InputStream, OutputStream (DataInputStream, DataOutputStream);
- ▷ BufferedReader e BufferedWriter;
- ▷ Console;
- ▷ FileInputStream, FileReader;
- ▷ FileOutputStream, FileWriter;

# Exercício: Salário com Bônus

Faça um programa que leia o nome de um vendedor, o seu salário fixo e o total de vendas efetuadas por ele no mês (em dinheiro). Sabendo que este vendedor ganha 15% de comissão sobre suas vendas efetuadas, informar o total a receber no final do mês, com duas casas decimais.

Exemplos de Entrada	Exemplos de Saída
PEDRO 700.00 0.00	TOTAL = R\$ 700.00

# Exercício: Cálculo Simples

Neste problema, deve-se ler o código de uma peça 1, o número de peças 1, o valor unitário de cada peça 1, o código de uma peça 2, o número de peças 2 e o valor unitário de cada peça 2. Após, calcule e mostre o valor a ser pago.

Exemplos de Entrada	Exemplos de Saída
12 1 5.30 16 2 5.10	VALOR A PAGAR: R\$ 15.50

# 2.

## NOÇÕES INICIAIS

**Classes, Objetos, Variáveis de Instância, Estado e Comportamento**



## Situação

Suponha uma situação em que um programa deva representar os alunos da Unifesp com identificação composta pelo nome e pelo número de matrícula.



# Forma estruturada

```
String nome = "Fernando Gomes";  
int RA = 101132;
```

```
String nome0 = "Fernando Gomes";  
int RA0 = 101132;
```

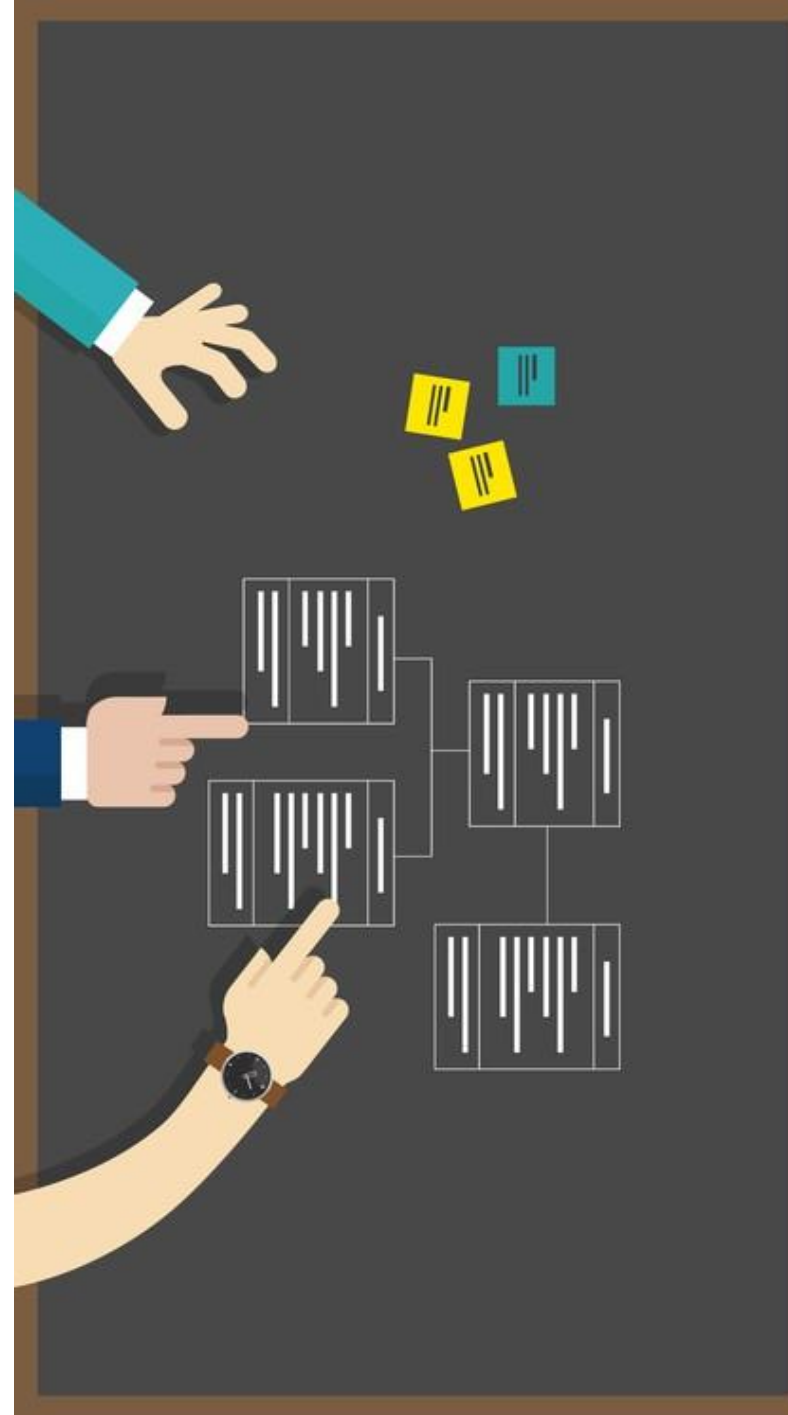
```
String nome1 = "Ricardo Silva";  
int RA1 = 101456;
```

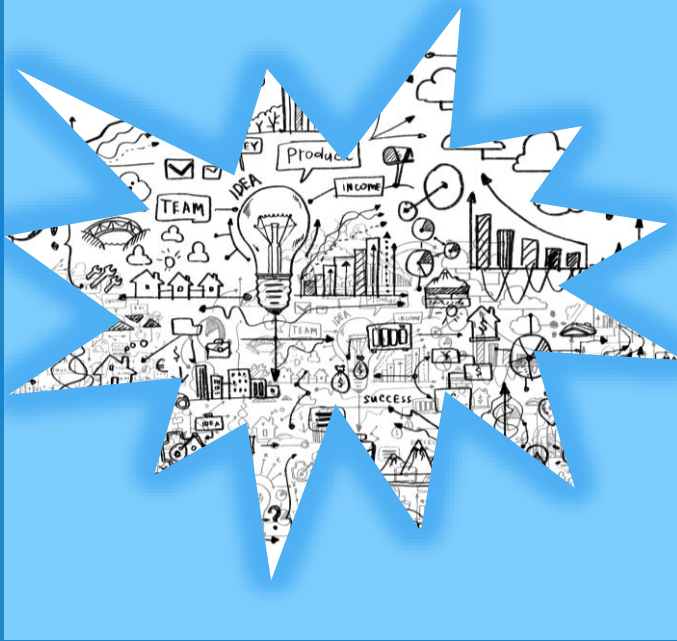
```
String[] nome = new String[5000];  
int[] RA = new int[5000];
```

```
nome[0] = "Fernando Gomes";  
RA[0] = 101132;
```

# Orientação a Objetos

No final dos anos setenta, com a popularização dos computadores pessoais e o aumento da presença da informática em nossa sociedade, pesquisadores começaram a notar a necessidade de representar os dados de forma mais fiel à forma como eles eram definidos em cada problema. Ou seja, uma identificação de estudante deveria ter um tipo de dado que representasse sua estrutura, tal qual ela era conceituada no mundo real (e não um conjunto de variáveis alocadas na memória). Surgiu então a **Orientação a Objetos**.





# Tipos de Dados

Um tipo de dado é um conjunto de valores e operações definidos naqueles valores. Os tipos de dados primitivos são implementados em Java em extensas bibliotecas de tipos por referência, que são adaptados para uma grande variedade de aplicações.

# Classe String

Você já deve ter usado um tipo de dado que não é primitivo – o tipo de dado *String*, cujos valores são sequências de caracteres. Para essa classe existem diversos **métodos** construídos em Java.

```
public class String
```

---

<code>String(String s)</code>	<i>create a string with the same value as s</i>
<code>int length()</code>	<i>number of characters</i>
<code>char charAt(int i)</code>	<i>the character at index i</i>
<code>String substring(int i, int j)</code>	<i>characters at indices i through (j-1)</i>
<code>boolean contains(String substring)</code>	<i>does this string contain substring?</i>
<code>boolean startsWith(String pre)</code>	<i>does this string start with pre?</i>
<code>boolean endsWith(String post)</code>	<i>does this string end with post?</i>
<code>int indexOf(String pattern)</code>	<i>index of first occurrence of pattern</i>
<code>int indexOf(String pattern, int i)</code>	<i>index of first occurrence of pattern after i</i>
<code>String concat(String t)</code>	<i>this string with t appended</i>
<code>int compareTo(String t)</code>	<i>string comparison</i>
<code>String toLowerCase()</code>	<i>this string, with lowercase letters</i>
<code>String toUpperCase()</code>	<i>this string, with uppercase letters</i>
<code>String replaceAll(String a, String b)</code>	<i>this string, with as replaced by bs</i>
<code>String[] split(String delimiter)</code>	<i>strings between occurrences of delimiter</i>
<code>boolean equals(Object t)</code>	<i>is this string's value the same as t's?</i>
<code>int hashCode()</code>	<i>an integer hash code</i>

# Tipos Agregados de Dados

A maioria das linguagens de programação suporta o conceito de variáveis tipadas, ou seja, uma variável pode ser do tipo int, double, char, etc. Embora essas linguagens possuam um bom número de tipos pré-definidos, seria mais interessante que o programador pudesse definir seus próprios tipos de dados.





# Tipos Agregados de Dados

Essa questão é resolvida através da implementação de **Tipos**

**Agregados de Dados** (em algumas linguagens, também conhecidos como Tipos Estruturados de Dados ou Registros). **Dados agregados** são tipos de dados definidos pelo programador no código-fonte de um sistema. Uma vez que um tipo de dado agregado tenha sido definido pelo programador, ele pode ser usado normalmente para declarar variáveis.





Em Java, os tipos agregados de dados são definidos através da palavra reservada **class**.

**Atenção:** o conceito de classes é bem mais amplo que simplesmente um tipo abstrato de dados. As características em detalhes serão apresentadas em detalhes ao longo do curso.



# Forma estruturada

```
//Tipo de dado definido pelo programador
class Estudante{
    String nome;
    int RA;
}
```

```
//Variáveis declaradas
Estudante estudante1;
Estudante estudante2;

Estudante[] Estudantes = new Estudante[8000];
```

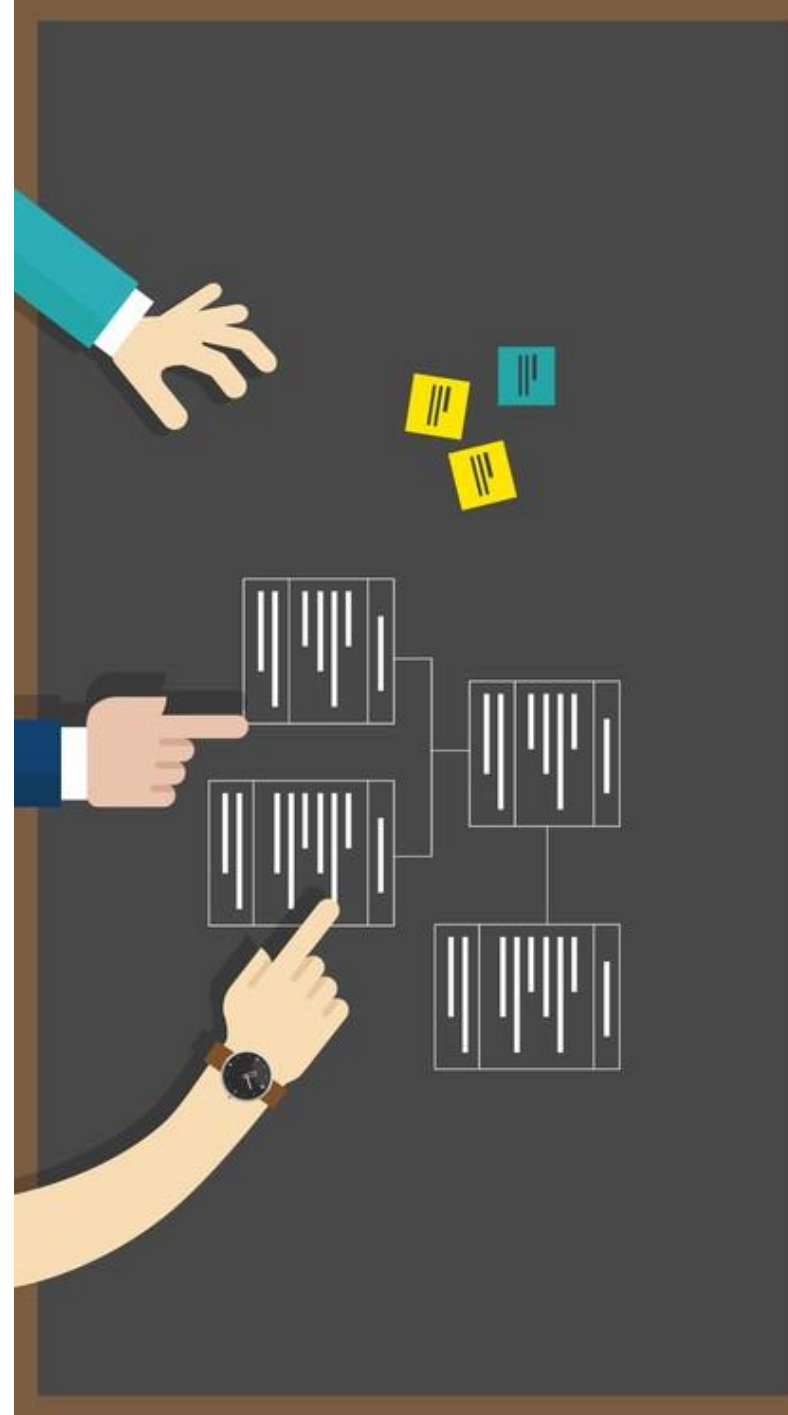
```
//Acesso aos dados
estudante1.nome = "Fernando Gomes";
Estudantes[0].nome = "Fernando Gomes";
```

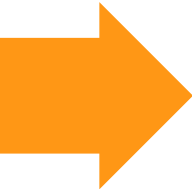
# Orientação a Objetos

Na terminologia de orientação a objetos chamamos os elementos que compõem uma classe de

**membros** dessa classe. No nosso exemplo, as variáveis *nome* e *RA* são chamadas de membros da classe *Estudante*.

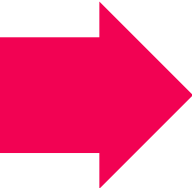
Além disso, quando criamos uma instância de um tipo agregado de dados, chamamos essa instância de **objeto**.





Uma **variável** representa a unidade básica de armazenamento temporário de dados e compõe-se de um tipo, um identificador e um escopo. Seu objetivo é armazenar um dado de determinado tipo primitivo para que possa ser recuperado e aplicado em operações posteriores.

```
< tipo > < identificador > = < valor >;
```



As **constantes** são unidades básicas de armazenamento de dados que não devem sofrer alterações ao longo da execução do aplicativo. O uso de constantes é menos frequente que o uso de variáveis. No entanto, há situações em que elas são requeridas.

```
final < tipo > < identificador > = < valor >;
```

# Termos básicos



## **Tipo Agregado de Dados**

É um tipo de dado definido pelo programador. Agregado pelo fato de ser definido a partir da agregação de um ou mais tipos primitivos de dados em Java.



## **Classe**

A tradução em linguagem orientada a objetos dos tipos agregados de dados. Além de agregar tipos primitivos de dados, uma classe provê outras funcionalidades.



## **Objeto**

Uma instância de uma classe. Podemos considerar uma classe como um gabarito, um modelo a partir do qual criamos objetos. Podemos declarar, por exemplo, uma classe que representa carteiras de estudantes. A classe representa um modelo de carteira de estudante. Já uma instância dessa classe é um conjunto de valores aplicados a esse modelo.



## **Membro**

Um membro de uma classe é um dos elementos que formam o modelo representado pela classe. No nosso exemplo, as variáveis Nome e RA são membros da classe Estudante.

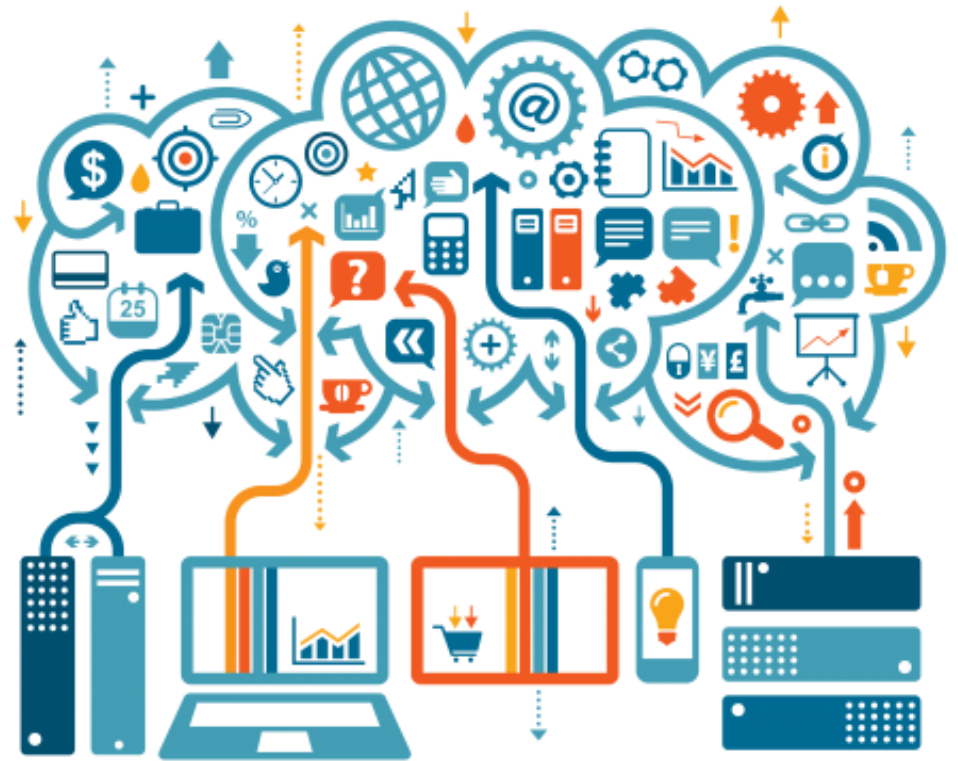


## **Referência**

Em Java, uma variável definida a partir de uma classe não contém as informações sobre um objeto dessa classe. Ao invés disso, a variável contém o endereço de memória no qual se encontram esses valores. Tais variáveis são chamadas de referências a um objeto.

# 3. MÉTODOS

**Definição, Declaração e Instanciação**

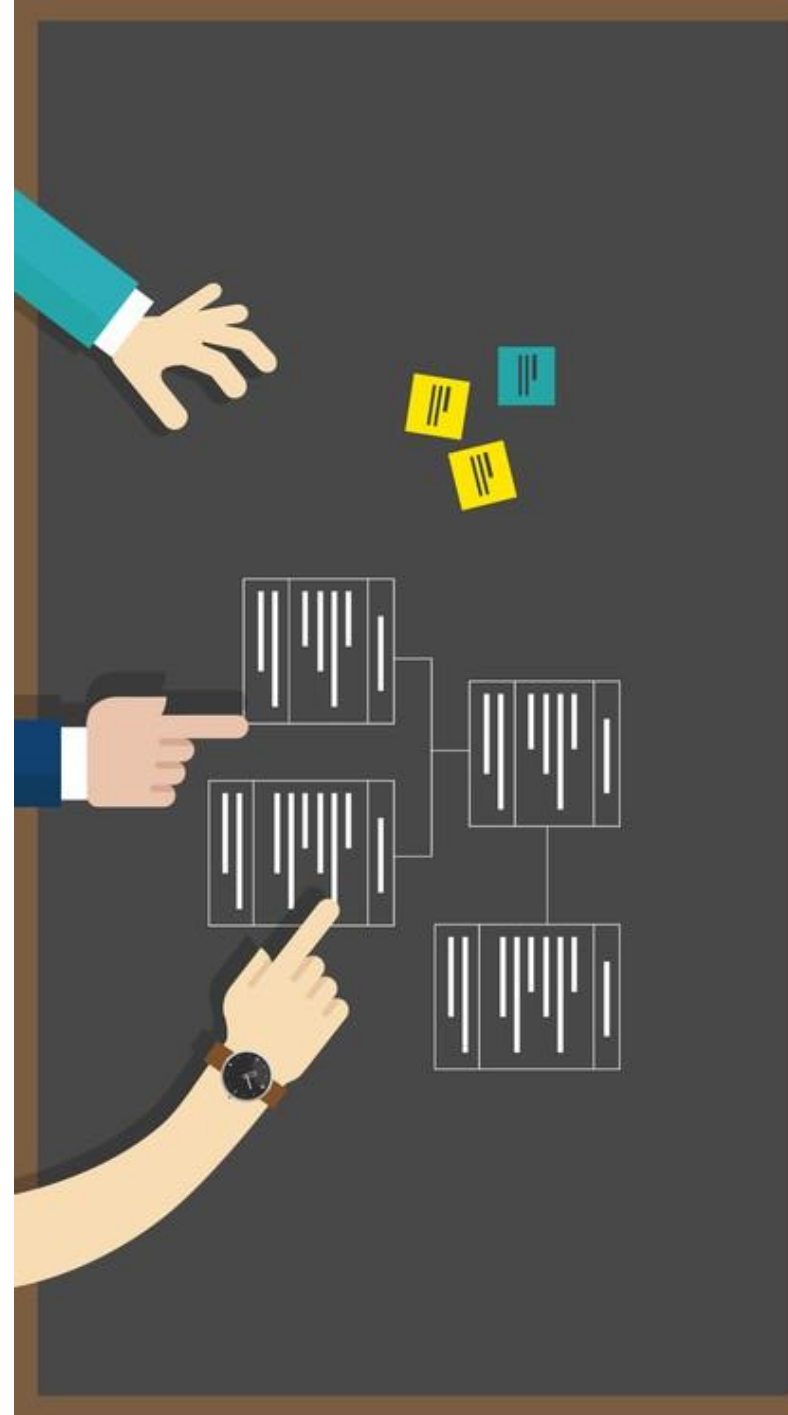


# Abstração de dados

Uma das vantagens da programação orientada a objetos é a capacidade de representar um objeto ou comportamento esperado por esse objeto em um único trecho de código, a conhecida classe.

# Tipos Abstratos de Dados

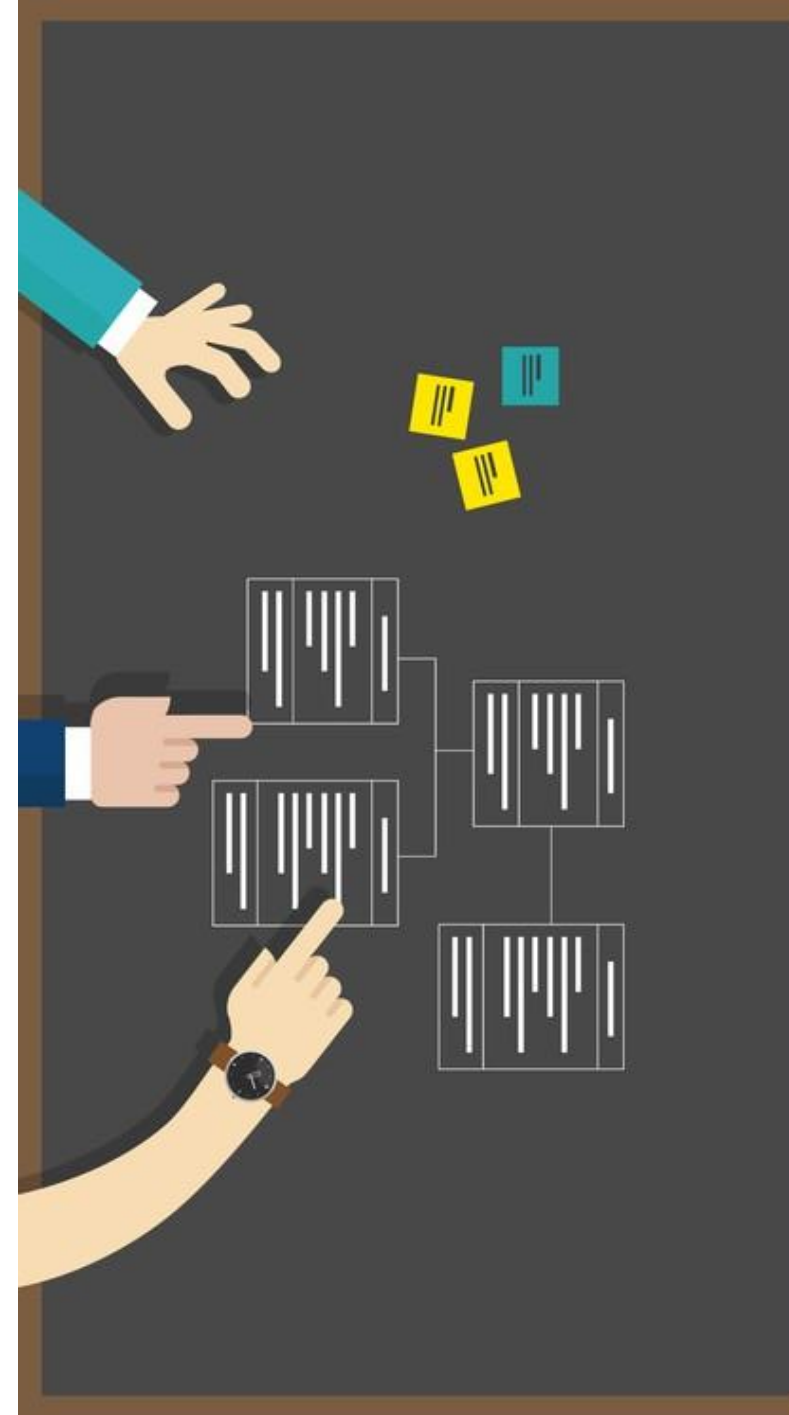
Quando definimos um tipo abstrato de dados, podemos também definir um conjunto de operações que podem incidir sobre esse tipo de dado. Este não é um conceito novo. Quando uma linguagem de programação define um tipo primitivo, tal como um inteiro, também é definido um conjunto de operações que pode ser aplicado a dados desse tipo, como **adição**, **subtração**, **multiplicação**, **divisão**, etc.



# Tipos Abstratos de Dados

Algumas linguagens de programação, incluindo Java, permitem uma estreita associação entre a declaração de um tipo de dados e a declaração das operações que incidem sobre as variáveis desse tipo. Essa associação normalmente é descrita como um **tipo abstrato de dados**.

Em Java você pode criar um tipo abstrato de dados através da implementação de **métodos**.





# Métodos

```
public class Avaliacao{
    public float[] trabalhos = new float[4];
    public float[] provas = new float[2];

    public void AtualizarNotaTrabalho(int IDtrabalho, int nota){
        trabalhos[IDtrabalho] = nota;
    }

    public void AtualizarNotaProva(int IDprova, int nota){
        provas[IDprova] = nota;
    }

    public static void main (String[] args){
        Avaliacao AvaliacaoMarcelo = new Avaliacao();
        AvaliacaoMarcelo.AtualizarNotaTrabalho(0, 8);
        AvaliacaoJoao.AtualizarNotaProva(1, 6);
    }
}
```

4.

# Encapsulamento

**Modificadores de acesso e Métodos Acessores**

# Encapsulamento

O encapsulamento vem da ideia de se encapsular o código para torná-lo o mais modular e reaproveitável possível, fazendo com que seja mais fácil alterar uma classe sem que haja retrabalho. Além disso, o encapsulamento provê o controle de acesso aos membros e métodos de uma determinada classe, controlando e protegendo os dados contidos nas instâncias da aplicação.



# Encapsulamento

O nível de proteção de uma variável ou método de uma classe é implementado através dos **modificadores de acesso** (indicados na declaração do atributo/método). A linguagem Java possui diversos modificadores de acesso, sendo os mais extremos o **public** (os membros da classe podem ser usados e acessados por todas as outras classes) e o **private** (os membros declarados não podem ser usados por nenhuma outra classe).





# Problema

Se o acesso aos atributos de uma classe ficam protegidos do meio externo, como podemos ler e modificar os valores dos membros de uma classe?

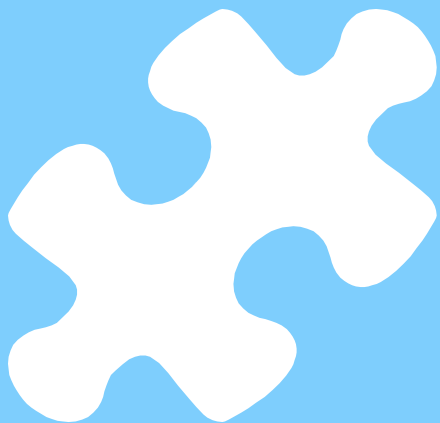
# Solução! Métodos Acessores

Métodos assessores permitem o acesso controlado aos atributos de uma classe, e normalmente são de dois tipos:

- ▷ *Get*: método que retorna o valor do atributo.
- ▷ *Set*: método que altera o valor de um atributo

# Métodos Acessores

```
public class Conta{  
    private double limite;  
    private double saldo;  
  
    public void setSaldo(double x) {  
        saldo = x;  
    }  
  
    public double getLimite() {  
        return limite;  
    }  
  
    public void setLimite(double y) {  
        limite = y;  
    }  
}
```



# DESAFIO

E aí, vamos praticar?



# Conversão de Tempo

Leia um valor inteiro, que é o tempo de duração em segundos de um determinado evento em uma fábrica, e informe-o no formato horas:minutos:segundos.

Exemplo de Entrada	Exemplo de Saída
556	0:9:16
1	0:0:1
140153	38:55:53

**Utilização de classes e métodos:**

**Classe Tempo com atributos:** horas, minutos, segundos.

**Métodos:** obterMinutos, obterHoras, imprimir

# Notas e Moedas

Leia um valor de ponto flutuante com duas casas decimais. Este valor representa um valor monetário. A seguir, calcule o menor número de notas e moedas possíveis no qual o valor pode ser decomposto. As notas consideradas são de 100, 50, 20, 10, 5, 2. As moedas possíveis são de 1, 0.50, 0.25, 0.10, 0.05 e 0.01. A seguir mostre a relação de notas necessárias.

Exemplo de Entrada	Exemplo de Saída
576.73	NOTAS: 5 nota(s) de R\$ 100.00 1 nota(s) de R\$ 50.00 1 nota(s) de R\$ 20.00 0 nota(s) de R\$ 10.00 1 nota(s) de R\$ 5.00 0 nota(s) de R\$ 2.00 MOEDAS: 1 moeda(s) de R\$ 1.00 1 moeda(s) de R\$ 0.50 0 moeda(s) de R\$ 0.25 2 moeda(s) de R\$ 0.10 0 moeda(s) de R\$ 0.05 3 moeda(s) de R\$ 0.01

**Utilização de classes e métodos:**

**Classes:** Nota, Moeda, Principal

# Obrigado!

## **Alguma pergunta?**

Você pode me contatar em:  
[ywassef@hotmail.com](mailto:ywassef@hotmail.com)