

# TIPOS PRIMITIVOS E MANIPULAÇÃO DE DADOS

Comentando seu código:

1. `//` - para comentar em apenas uma linha; tudo o que tiver depois dele será comentado;
2. `/*`  
`*/`  
- tudo o que tiver entre será o comentário; comentário de várias linhas;

Obs: por questões estéticas, os programadores utilizam `*` em todos os comentários, exemplo:

```
/*
 * Nos comentários de muitas linhas,
 * podemos escrever o quanto quiser
 * até indicarmos o final do comentário
 */
```

3. `/**`  
`*/`  
- são comentários para documentação; posso utilizar marcas específicas, como por exemplo dizer o autor do código, a versão do software e a primeira versão criada, como no exemplo:

```
/**
 * Exemplo para o CursoemVideo de Java
 * @author Gustavo Guanabara
 * @version 1.0
 * @since 2015-01-01
 */
```

Obs: para saber mais, procurar JAVADOCS

---

Tipos primitivos:

Lembrando de algoritmos:

```
var
  idade : Inteiro
  sal : Real
  letra : Caractere
  casado : Logico
inicio
  idade <- 3
  sal <- 1825.54
  letra <- "G"
  casado <- falso
```

Como incluir essas 4 declarações em Java:

- 1- `int idade = 3;`  
`float sal = 1825.54f;`  
`char letra = 'G';`  
`boolean casado = false;`

Obs: a variável real é um ponto flutuante, então no Java é considerado **float**

Obs: o pequeno "f" minúsculo indica que o número é **float**

Obs: o G nos algoritmos tem aspas dupla, já no **Java tem aspas simples**

Obs: o tipo **char** ele só ocupa uma letra! E não tem tipos que armazenem mais de uma letra, mas temos uma classe para isso...

- 2- `int idade = (int) 3;`  
`float sal = (float) 1825.54;`  
`char letra = (char) 'G';`  
`boolean casado = (boolean) false;`

- Técnica para especificar tipos de valores específicos: **typecast**
- O entre parênteses antes do 3 quer dizer: “considere esse 3 como inteiro”

3- **Integer** idade = new **Integer**(3);  
**Float** sal = new **Float**(1825.54);  
**Charater** letra = new **Charater**('G');  
**Boolean** casado = new **Boolean**(false);

- Técnica que utiliza classes (LEMBRANDO QUE COMEÇAM COM A PRIMEIRA LETRA MAIÚSCULA): **Wrapper Class**

Obs: nas duas primeiras técnicas elas são variáveis, e na terceira ela é um objeto

Sempre que eu utilizar o “new” dentro de uma declaração, esse “new” está criando um objeto, então eu preciso ter uma classe referenciando-o!

```
int idade = 3;
float sal = 1825.54f;
char letra = 'G';
boolean casado = false;

int idade = (int) 3;
float sal = (float) 1825.54;
char letra = (char) 'G';
boolean casado = (boolean) false;

Integer idade = new Integer(3);
Float sal = new Float(1825.54);
Character letra = new Character('G');
Boolean casado = new Boolean(false);
```

- Existe uma tabela com cada um dos tipos suportados pelo Java, inclusive com suas classes em invólucro

Família	Tipo Primitivo	Classe Invólucro	Tamanho	Exemplo
Lógico	boolean	Boolean	1 bit	true
Literais	char	Character	1 byte	'A'
	-	String	1 byte/cada	"JAVA"
Inteiros	byte	Byte	1 byte	127
	short	Short	2 bytes	32 767
	int	Integer	4 bytes	2 147 483
	long	Long	8 bytes	2 <sup>63</sup>
Reais	float	Float	4 bytes	3.4e+38
	double	Double	8 bytes	1.8e+308

OBS: o **tipo char** aceita apenas uma letra. Se eu quero guardar um nome, por exemplo, eu uso a **classe invólucro String**. Não existe **tipo string** no Java, mas sim uma **classe invólucro**!

OBS: 127 é o maior número inteiro guardado pelo **tipo primitivo byte**, assim como 32767 pelo **tipo primitivo short** e assim por diante

Mas qual a necessidade de tantos tipos primitivos? PARA ECONOMIZAR MEMÓRIA!

Ex: em um relógio, não adianta ter um **tipo int** que ocupa 7 bytes para guardar um número, pois a memória do dispositivo pode ser pequena!

Saída de dados:

- É quando eu pego um dado que está na memória do computador e mostro de alguma maneira

```
float nota = 8.5f;
System.out.print("Sua nota é" + nota);
System.out.println("Sua nota é" + nota);
```

Obs: o **System.out.print** é pra escrever a nota na tela!

Obs: o símbolo de + significa soma ou concatenação

- O **ln** serve para pular uma linha

Exemplo:

1- float nota = 8.5f;

System.out.print("A nota é " + nota);

```
run:
A nota é 8.5CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

2- float nota = 8.5f;

System.out.println("A nota é " + nota);

```
run:
A nota é 8.5
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Outros tipos para mostrar na tela:

```
System.out.printf("Sua nota é %.2f", nota);
System.out.format("Sua nota é %.2f", nota);
```

Obs: o **f** vem de formatado.

Obs: basicamente, a nota vai se encaixar no **%f**. Se eu quero que a nota tenha 2 casas decimais, eu coloco o número .2 entre o **%f**, ou seja, **%.2f**

3.1- float nota = 8.5f;

System.out.printf("A nota é %.2f", nota);

```
run:
A nota é 8,50CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

3.2- float nota = 8.5f;

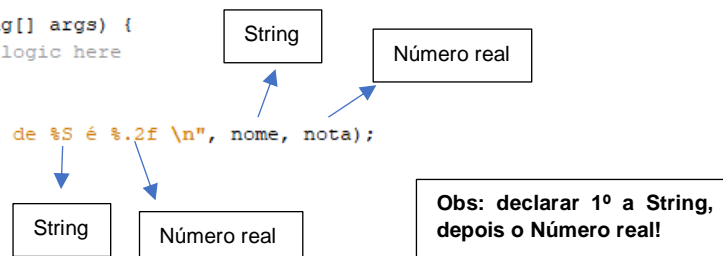
System.out.printf("A nota é %.2f \n", nota);

```
run:
A nota é 8,50
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Obs: o **ln** serve para quebrar linha

Exemplo: se eu quero mostrar a nota de Gustavo:

```
public static void main(String[] args) {
    // TODO code application logic here
    String nome = "Gustavo";
    float nota = 8.5f;
    System.out.printf("A nota de %S é %.2f \n", nome, nota);
}
```



Obs: o **format** tem a mesma sintaxe que o **printf**!

4- float nota = 8.5f;

System.out.format("A nota é %.2f \n", nota);

```
run:
A nota é 8,50
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## Entrada de dados:

- É todo dado que está fora e eu quero colocar para dentro do computador, normalmente utilizando o teclado
- O Java não vem com comando nenhum de entrada de dados!

Obs: o `Java.lang` não tem nenhum comando específico para a entrada de dados, então precisamos usar uma outra classe: **import Java.util.Scanner** (COMANDO PARA IMPORTAR A CLASSE!)

- Para ativar essa classe, vou precisar criar um objeto, então vou utilizar:

```
Scanner teclado = new Scanner(System.in);
```

Nome de objeto  
qualquer

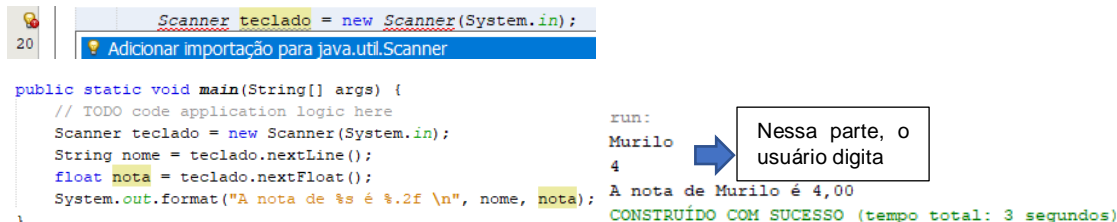
O entre parênteses eu vou  
passar o que ele vai escanear,  
o que ele vai ficar monitorando

Obs: se o **System.out** era o comando de saída, o comando de entrada é **System.in**

## NA PRÁTICA:

Ex: ler o nome e a nota e permitir com que o usuário digite isso

Lembrando: adicionar Scanner à importação!



The screenshot shows a Java IDE with the following code:

```
import java.util.Scanner;

public static void main(String[] args) {
    // TODO code application logic here
    Scanner teclado = new Scanner(System.in);
    String nome = teclado.nextLine();
    float nota = teclado.nextFloat();
    System.out.format("A nota de %s é %.2f \n", nome, nota);
}
```

The output window shows the following text:

```
run:
Murilo
4
A nota de Murilo é 4,00
CONSTRUÍDO COM SUCESSO (tempo total: 3 segundos)
```

Annotations in the image:

- A blue box with the text "Adicionar importação para java.util.Scanner" is shown above the code.
- A blue arrow points from the `Scanner` class name in the code to a box containing the text "Nessa parte, o usuário digita".

Obs: é importante dar mais interatividade na tela, e para isso:

```
public static void main(String[] args) {
    // TODO code application logic here
    Scanner teclado = new Scanner(System.in);
    System.out.print("Digite o nome do aluno: ");
    String nome = teclado.nextLine();
    System.out.print("Digite a nota do aluno: ");
    float nota = teclado.nextFloat();
    System.out.format("A nota de %s é %.2f \n", nome, nota);
}
```

Annotations in the image:

- Two blue arrows point from the `System.out.print` statements in the code to a box containing the text "Comandos adicionados".

## Na prática:

```
run:
Digite o nome do aluno: Murilo
Digite a nota do aluno: 10
A nota de Murilo é 10,00
CONSTRUÍDO COM SUCESSO (tempo total: 13 segundos)
```

- Existem vários métodos para ler valores de tipos diferentes!

Ex: seu eu quiser que meu objeto (no exemplo, teclado), para ler um número inteiro, eu coloco:

```
int idade = teclado.nextInt();
float salario = teclado.nextFloat();
String nome = teclado.nextLine();
```

Annotations in the image:

- A blue arrow points from the `nextInt()` method in the code to a box containing the text "Ou nextDouble(), nextShort(), nextByte()".

## Incompatibilidades números <-> string:

### Exemplo:

```
int idade = 30;
String valor = idade;
```

Vai dar incompatibilidade pois **int** não pode ser convertido para **String**

Maneira correta:

```
int idade = 30;  
String valor = idade;  
String valor = (String) idade;  
String valor = Integer.toString(idade);
```

E se for ao contrário?

```
String valor = "30";  
int idade = valor;  
int idade = (int) valor;  
int idade = Integer.parseInt(valor);
```

Obs: parse significa converter, então eu vou fazer com que o **valor** seja convertido para inteiro

Obs: tudo isso que fiz em valores inteiros também pode ser para valores reais!

Ex:

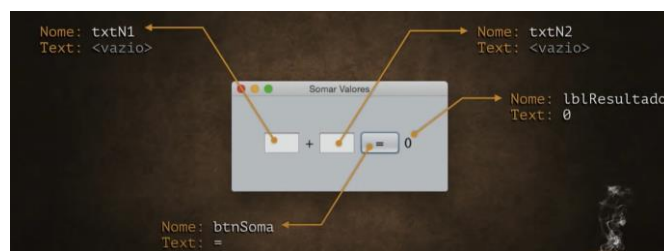
```
String valor = "30.5";  
float idade = Float.parseFloat(valor);  
System.out.println(idade);
```

- Funciona para todas as classes invólucros!

---

Agora com o Swing:

Lembrando sempre de colocar nomes nos objetos em que vou utilizar, como por exemplo:



Para esse exemplo, vou utilizar:

- 2 campos de texto
- 2 labels (1 como o símbolo de + e outro como lblResultado)
- 1 botão

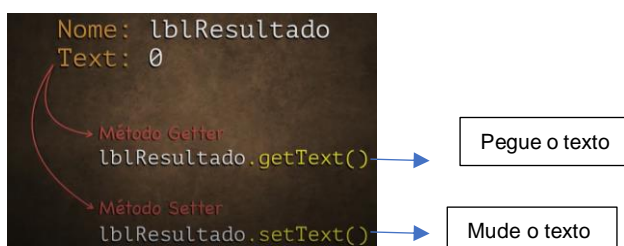
Após isso, devo criar o evento

Como tirar dado dentro de uma caixa de texto: **MÉTODOS ACESSORES**


**Método Getter:** pegar o valor que está dentro

**Método Setter:** colocar um valor lá dentro

Exemplo:

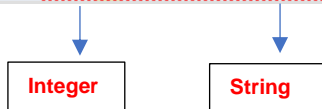


Então:



```
int n1 = txtN1.getText();  
int n2 = txtN2.getText();
```

Obs:



Então: utilizar as classes invólucros! Como exemplo o Integer.parseInt, ficando assim:

```
int n1 = Integer.parseInt(txtN1.getText());  
int n2 = Integer.parseInt(txtN2.getText());
```

Código do evento:

```
private void btnSomaActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    int n1 = Integer.parseInt(txtN1.getText());  
    int n2 = Integer.parseInt(txtN2.getText());  
    int s = n1 + n2;  
    lblSoma.setText(Integer.toString(s));  
}
```