

INTRODUÇÃO A JAVA

Professor: Katyusco de Farias Santos

Primeiro programa em Java

- Criar o arquivo abaixo no bloco de notas, com nome MeuPrograma.java

```
class MeuPrograma {  
    public static void main(String[] args) {  
        System.out.println("Minha primeira aplicação Java!");  
    }  
}
```

A instrução `System.out.println(...)` imprime texto no terminal!

Declaração de variáveis

- Em Java, uma variável é declarada da seguinte forma:

tipoDaVariável **nomeDaVariável**;

- Por exemplo,

int idade;

- A variável idade é do tipo inteiro, e só pode armazenar valores deste tipo.

```
int idade;  
idade = 15;
```

```
int idade = 15;
```

Declaração de variáveis

□ Tipos primitivos em Java.

| Tipo | Tamanho | Descrição |
|----------------|---------|---|
| boolean | 1 bit | Pode assumir os valores true ou false |
| byte | 1 byte | Inteiro de 8 bits. Valores entre -128 e 127 |
| short | 2 bytes | Inteiro de 16 bits. Valores entre -32.768 e 32.767 |
| char | 2 bytes | Caractere em notação Unicode (16 bits) |
| int | 4 bytes | Int. de 32 bits. Valores entre 2.147.483.648 e 2.147.483.647 |
| float | 4 bytes | Ponto flutuante de 32 bits. Valores entre 1.40239846e-46 e 3.40282347e+38 |
| long | 8 bytes | Int. de 64 bits. Valores entre -2^{63} e $2^{63}-1$ |
| double | 8 bytes | Ponto flut. de 64 bits. Val. entre 4.94065645841246544e-324 e 1.7976931348623157e+308 |

Atribuição de valores

- Em Algoritmo Estruturado:

numero \leftarrow 8

- Em Pascal:

numero := 8;

- Em Java:

numero = 8;

Comentários

- Para adicionar um comentário em Java, utilizar as seguintes notações:

- `//` comentário para uma linha de comando

- `/*`

comentário para um bloco de linhas

`*/`


```
/* comentário daqui,  
ate aqui */
```

```
// uma linha de comentário sobre a idade  
int idade;
```

Casting de tipos

- Em Java, atribuir o valor de uma variável de um *tipo A* a uma variável de *tipo B*, pode gerar erro:

```
double valor = 2.8;  
int x = valor;    // Erro em tempo de compilação  
int i = 2.8;    // Também ocorre erro, pois 2.8 é real
```



```
double valor = 8;  
int x = valor;    // Erro em tempo de compilação
```



```
int x = 8;  
double valor = x;    // Funciona normalmente
```



Casting de tipos

- Para evitar erros em tempo de compilação na atribuição de uma variável *tipo A* para uma variável *tipo B*, faz-se o **casting**:
 - ▣ Processo explícito de converter um variável *tipo A* em variável *tipo B*:
 - Muito cuidado ao usar o casting!
 - ▣ Exemplo:

```
tipoB outraVariavel = valor_do_tipoB;  
tipoA nomeVariavel = (tipoA) outraVariavel;  
  
double valor = 2.88912;  
int x = (int) valor; // a variável x terá valor 2
```


Casting de tipos

- Verifique se há erros, e informa quais, nos exemplos abaixo:
 - ▣ Como corrigir os erros abaixo?

```
long valorLong = 5000;  
int valorInt = valorLong;
```

```
float valorF = 0.8;  
double valorD = valorF;
```

```
float valorF = 8;  
double valorD = 1;  
float soma = valorF + valorD;
```

```
byte valorB = 2;  
int valorI = valorB;
```

Casting de tipos

- Verifique se há erros, e informa quais, nos exemplos abaixo:
 - ▣ Como corrigir os erros abaixo?

```
long valorLong = 5000;  
int valorInt = valorLong;
```



```
float valorF = 0.8;  
double valorD = valorF;
```



```
float valorF = 8;  
double valorD = 1;  
float soma = valorF + valorD;
```




```
byte valorB = 2;  
int valorI = valorB;
```




Casting de tipos

- Verifique se há erros, e informa quais, nos exemplos abaixo:
 - ▣ Como corrigir os erros abaixo?


```
long valorLong = 5000;  
int valorInt = (int)valorLong;
```




```
float valorF = 0.8f;  
double valorD = valorF;
```



```
float valorF = 8;  
double valorD = 1;  
float soma = valorF + (float)valorD;
```



```
byte valorB = 2;  
int valorI = valorB;
```



Casting de tipos

□ Castings possíveis em Java.

| Para: | byte | short | char | int | long | float | double |
|--------|--------|-----------|-----------|-----------|-----------|-----------|-----------|
| De: | byte | short | char | int | long | float | double |
| byte | ---- | Implícito | Implícito | Implícito | Implícito | Implícito | Implícito |
| short | (byte) | ---- | Implícito | Implícito | Implícito | Implícito | Implícito |
| char | (byte) | (short) | ---- | Implícito | Implícito | Implícito | Implícito |
| int | (byte) | (short) | (char) | ---- | Implícito | Implícito | Implícito |
| long | (byte) | (short) | (char) | (int) | ---- | Implícito | Implícito |
| float | (byte) | (short) | (char) | (int) | (long) | ---- | Implícito |
| double | (byte) | (short) | (char) | (int) | (long) | (float) | ---- |

Fonte: Apostila online da Caelum

Decisão If-Else

- A decisão **if-else** é útil para executar instruções que dependem de uma certa condição;

- Sintaxe do **if-else**:

```
if (condicaoBooleana) {  
    // código Java da condição obtida  
} else {  
    // código Java da condição não obtida  
}
```

- O termo **condicaoBooleana** é uma expressão que só poderá retornar **true** ou **false**;

Decisão If-Else

- Para concatenar expressões booleanas, basta usar os operadores lógicos a seguir:
 - “E” lógico **equivale** a `&&`;
 - “OU” lógico **equivale** a `| |`.
- Para negar uma expressão booleana, basta adicionar o sinal de exclamação (!) à frente da expressão.
 - NÃO lógico **equivale** a `!`

Decisão If-Else

□ Exemplos.

```
int idade = 20;
if (idade < 21) {
    System.out.println("Não pode entrar");
} else {
    System.out.println("Pode entrar");
}
```

```
int idade = 20;
boolean amigoDoDono = false;
if (idade < 21 && amigoDoDono == false) {
    System.out.println("Não pode entrar");
} else {
    System.out.println("Pode entrar");
}
```

Decisão Switch-Case

- A estrutura **switch-case** identifica o valor de uma variável (**switch**), e a testa em uma lista de valores possíveis (**case**).
- Sintaxe do **switch-case**.

```
switch (variável) {  
    case valor1 : // comandos Java para valor1  
    case valor2 : // comandos Java para valor2  
    case valor3 : // comandos Java para valor3  
    default : // se nenhum dos valores anteriores ocorreu  
}
```


Decisão Switch-Case

- Os tipos da **variável** do **switch** só podem ser:
 - ▣ Byte;
 - ▣ Short;
 - ▣ Char;
 - ▣ Int;
 - ▣ Strings – cadeia de caracteres;
 - ▣ Enumeradores.

- Ao final de bloco de comandos do case, deverá ser colocado o **break**:
 - ▣ Para evitar que o **case** seguinte seja executado.

Decisão Switch-Case

□ Exemplo.

```
int nota = 8;
switch (nota) {
    case 10 : System.out.println("Excelente");
              break;
    case 9  : System.out.println("Muito bom");
              break;
    case 8  : System.out.println("Bom");
              break;
    case 7  : System.out.println("Ná média");
              break;
    default : System.out.println("Nota baixa");
}
```

Repetição com While

- O **while** é utilizado para repetir um trecho de código até que uma condição seja atingida:
 - ▣ Laço (loop).
- Sintaxe do **while**.

```
while (condicaoBooleana) {  
    // comandos a serem executados  
}
```

- O termo **condicaoBooleana** é uma expressão que só poderá retornar **true** ou **false**:
 - ▣ Podem ser usados o **&&** e **||** para concatenação de expressões.

Repetição com While

□ Exemplo.

```
int idade = 15;
while (idade < 18) {
    System.out.println(idade);
    idade = idade + 1;
}
```

```
int i = 0;
while (i != 10) {
    System.out.println(i);
    i = i + 1;
}
```

Repetição com For

- Outro comando de loop é o **for**;
 - ▣ Repetir determinado trecho de código até uma condição ser atingida.
- Sintaxe do **for**.

```
for(inicialização; condição; incremento) {  
    // comandos a serem executados  
}
```

- Exemplo.

```
for(int i = 0; i < 10; i++) {  
    // comandos a serem executados  
}
```

Simples Leitura a Partir do Teclado

- A partir do Java 1.5 ou J2SE 5 está disponível a classe **Scanner**, do pacote **java.util**, que implementa as operações de entrada de dados pelo teclado no console.
- Primeiro: importar o pacote
 - ▣ `import java.util.Scanner; // importando o pacote`
- Segundo: Instanciar e criar um objeto Scanner
 - ▣ `Scanner ler = new Scanner(System.in);`
- Por fim: Declarar variáveis e ler seus valores de acordo com seu tipo

```
int n;  
n = ler.nextInt( );
```

```
float f;  
f = ler.nextFloat( );
```

```
String f;  
f = ler.next( );
```

Simples Leitura a Partir do Teclado

```
import java.util.Scanner; // importar o pacote java.util.Scanner
class LendoDoTeclado {
    public static void main(String args[]){
        Scanner ler = new Scanner(System.in);
        int n;
        System.out.println("Informe um número inteiro: ");
        n = ler.nextInt();
        double salario;
        System.out.println("Informe um número double: ");
        salario = ler.nextDouble();    // A vírgula é o separador aceito
        String palavra;
        System.out.println("Informe uma palavra simples: ");
        palavra = ler.next();          // Ler uma palavra simple (sem espaço)
        System.out.println("inteiro lido: " + n);
        System.out.println("double lido: " + salario);
        System.out.println("palavra simples lida: " + palavra);
    }
}
```

Métodos em Java

- Dentro das classes, é possível criar funções:
 - ▣ Bloco de comandos que poderão ser acessados repetidas vezes através de uma chamada;
 - ▣ Melhora manutenção e reutilização de código.
- Em Java, as funções são conhecidas como **métodos**.
- Funções x Procedimentos:
 - ▣ Em Java, os métodos **devem** ter um tipo de retorno:
 - Mesmo que defina que não há retorno, com o tipo **void**.
 - ▣ Um procedimento em Java é uma função com tipo de retorno **void**.

Métodos em Java

□ Sintaxe de um Método.

```
tipoVariavel nomeMetodo (tipoVariavel p1, tipoVariavel p2) {  
    // comandos a serem executados  
}
```

□ Exemplos.

```
double quadrado(double numero) {  
    return (numero*numero);  
}  
  
int soma(int n1, int n2) {  
    return (n1 + n2);  
}  
  
void impSoma(int n1, int n2) {  
    System.out.println(n1+n2);  
}
```

Métodos em Java

- Os métodos podem receber argumentos (também chamados de parâmetros):
 - ▣ São variáveis utilizadas quando a função inicia sua execução;
 - ▣ As variáveis de um argumento são temporárias:
 - Ao ser concluída a execução da função, as variáveis deixam de existir;
 - O escopo das variáveis de um método reside apenas no método.

Métodos em Java

- O tipo de retorno *void* significa que nenhuma informação será devolvida ao trecho de código que está ativando o método.

```
class Teste{  
    public static void main(String args[]){  
        int x = impSoma(3, 5);  
    }  
  
    static void impSoma(int n1, int n2){  
        System.out.println(n1+n2);  
    }  
}
```



Métodos em Java

- Não se retorna valores quando o tipo de retorno de um método é *void*.



```
class Teste{
    public static void main(String args[]){
        impSoma(3, 5);
    }

    static void impSoma(int n1, int n2){
        return (n1+n2);
    }
}
```

Métodos em Java

- Não se retorna valores quando o tipo de retorno de um método é *void*.



```
class Teste{
    public static void main(String args[]){
        impSoma(3, 5);
    }

    static void impSoma(int n1, int n2){
        System.out.println(n1+n2);
    }
}
```

Métodos em Java

- ❑ Não é permitido colocar valor padrão para argumentos nos métodos Java.

```
void impSoma(int n1 = 10) {  
    System.out.println(n1);  
}
```



```
void impSoma(double n = 20.5) {  
    System.out.println(n1);  
}
```



Métodos em Java

- Em vez disso, é possível ter dois métodos com mesmo nome, mas mudando o argumento:
 - ▣ Sobrecarga de métodos.
- A sobrecarga de métodos deve obedecer às seguintes diretrizes:
 - ▣ O método *sobrecarregado* deve ter mesmo tipo de retorno;
 - ▣ Os parâmetros podem ser iguais em quantidade, mas tipo deve ser diferente.
 - ▣ Ou o número de parâmetros pode ser diferente.



```
static void impSoma(int n1) {  
    System.out.println(n1);  
}
```

```
static void impSoma() {  
    System.out.println(10);  
}
```

Métodos em Java

- Para melhor organização do código, é possível separar métodos em diversas classes.

```
class Teste{  
    public static void main(String args[]){  
        int soma;  
        soma = Calculadora.soma(3, 5);  
    }  
}
```

- O que é `Calculadora.soma()`?
- O que é `Calculadora`?
- Onde está implementado `Calculadora.soma()`?

Métodos em Java

- Para melhor organização do código, é possível separar métodos em diversas classes.

```
class Teste{  
    public static void main(String args[]){  
        int soma;  
        soma = Calculadora.soma(3, 5);  
    }  
}
```

- O que é `Calculadora.soma()`?
 - ▣ `soma()` é um método **de classe** (ou **estático**) da classe `Calculadora`:
- O que é `Calculadora`?
 - ▣ `Calculadora` é uma outra classe.
- Onde está implementado `Calculadora.soma()`?
 - ▣ A classe `Calculadora` está na mesma pasta que `Teste`.

Métodos em Java

- E se a classe Calculadora estiver dentro da pasta teste?
 - ▣ Afinal, em um grande projeto Java, não é viável ter todas as classes na mesma pasta;
 - ▣ Resposta: uso de pacotes em Java.

```
import teste.Calculadora;

class Teste{
    public static void main(String args[]){
        int soma;
        soma = Calculadora.soma(3, 5);
    }
}
```

Métodos em Java



- Como compilar classes em pastas separadas?
- Como reusá-las em outras classes?
- O que é método de classe ou estático?