

# Introdução ao Java

---

Marco A L Barbosa  
malbarbo.pro.br

Departamento de Informática  
Universidade Estadual de Maringá



# Conteúdo

Introdução

Olá mundo

Variáveis

Tipos de dados

Campos e métodos estáticos

Documentação

Operadores

Controle de fluxo

Referências

# Introdução

# Introdução

- Por enquanto vamos aprender parte da sintaxe e dos tipos de dados, sem se preocupar com os aspectos de programação orientado a objetos
- Vamos usar algumas construções sem entender precisamente como funcionam, ao longo das aulas estas construções serão esclarecidas
- Vamos usar comparações com a linguagem C para “aproveitar” o conhecimento de vocês
- Vamos começar com um editor de texto e compilação no terminal, depois usaremos um IDE

**Olá mundo**

# Olá mundo

```
public class Ola {
    public static void main(String[] args) {
        // - args é um arranjo indexado a partir de 0
        //   que contém os argumentos passados na linha de comando
        // - args também tem um campo length com a quantidade
        //   de elementos no arranjo
        if (args.length == 1) {
            System.out.println("Olá " + args[0] + "!");
        } else if (args.length == 2) {
            System.out.printf("Olá %s e %s!\n", args[0], args[1]);
        } else {
            System.out.println("Olá mundo!");
        }
    }
}
```

Para compilar um arquivo Java, invocamos o compilador `javac` e passamos como argumento o nome do arquivo. As dependências (no mesmo projeto) são compiladas automaticamente.

```
$ javac Ola.java
```

## Olá mundo

Para executar um programa, invocamos a máquina virtual com o comando `java` e passamos como argumento o nome da classe que contém o método `main`. Os argumentos para o programa são especificados após o nome da classe.

```
$ java Ola
```

```
Olá mundo!
```

```
$ java Ola João
```

```
Olá João!
```

```
$ java Ola João Maria
```

```
Olá João e Maria!
```

```
$ java Ola João Maria Pedro
```

```
Olá mundo!
```



- Por conversão, os nomes de classe são escritos em CamelCase, começando com maiúscula
- O código de cada classe pública deve ser escrito em um arquivo com o mesmo nome da classe e com extensão `.java`
- Um arquivo pode ter muitas declarações de classes, mas no máximo uma classe pública
- O compilador gera um arquivo `.class` para cada classe compilada

# Variáveis

- Nome do tipo seguido do nome da variável

```
long x = 20;  
boolean tag = true;
```

- Inferência de tipo (Java 10)

```
var x = 20;           // tipo inferido int  
var tag = false;    // tipo inferido boolean
```

- Por convenção nomes de variáveis são escritos em camelCase, começando com minúscula

# Tipos de dados

## Inteiros primitivos

Tipo	Bytes	Min	Max
byte	1	-128	127
short	2	-32768	32767
int	4	-2147483648	2147483647
long	8	-9223372036854775808	9223372036854775807

## Ponto flutuantes primitivos

Tipo	Bytes	Precisão
float	4	6 a 7 casas decimais
double	8	14 a 15 casas decimais

# Outros tipos primitivos

- `boolean`
  - Valores `true` ou `false`
  - O tamanho dependente da máquina virtual
- `char`
  - Um caractere Unicode de 16 bits: `'\u0000'` a `'\uffff'`
  - Exemplo

```
char x = 'a';
```

Strings não são tipos primitivos, mas são tratadas de forma especial pelo compilador.

- Literais (são armazenados em um *pool*)
- Concatenação com o operador +
- Exemplos

```
String s = "uva";  
String x = "sudo de " + s;  
String y = x + " de " + 500 + " ml";  
assert y == "suco de uva de 500 ml";
```



## Literais numéricos

- Literais numéricos sem ponto são do tipo `int` a menos que algum sufixo seja especificado
  - `l` ou `L` → `long`
  - `f` ou `F` → `float`
  - `d` ou `D` → `double`
- Literais numéricos com ponto são do tipo `double` a menos que algum sufixo seja especificado
  - `f` ou `F` → `double`

- `_` pode aparecer entre os dígitos de qualquer literal numérico
- Literais em hexadecimal são prefixados com `0x`
- Literais em binário são prefixados com `0b`

## Exemplos

```
System.out.println(23); // imprime 23
```

```
System.out.println(23f); // imprime 23.0
```

```
System.out.println(123_456.0); // imprime 123456.0
```

```
System.out.println(0xabc); // imprime 2748
```

```
System.out.println(0b1010); // imprime 10
```

# Conversões

- Conversões sem perda de magnitude podem ser feitas de forma implícita
  - byte para short, int, long, float e double
  - short para int, long, float e double
  - char para int, long, float e double
  - int para long, float e double
  - long para float e double
  - float para double
- Conversões com perda devem ser feitas de forma explícita

- Exemplos

```
int x = 120;
```

```
long y = x;
```

```
short z = (int) x;
```

## Tipos primitivos vs tipos referência

- Os tipos que não são primitivos são chamados de tipos referência
  - `int []`, `String`, `ArrayList`, etc
- Variáveis locais e parâmetros de tipos primitivos são sempre alocados na pilha
- As referências também são alocadas na pilha, mas os valores referenciados são sempre alocados no heap usando o operador `new`

```
// a referência para o arranjo é armazenada na pilha, mas  
// os 10 valores (e mais outros dados) são armazenados no heap  
int [] x = new int [10];  
// os valores alocados com `new` são desalocados automaticamente  
// pelo coletor de lixo
```

## Tipos primitivos vs referências

- Esta dicotomia entre tipos primitivos e tipos referências permite mais eficiência em tempo de execução, mas torna a linguagem menos ortogonal
- Para amenizar esta dualidade, cada tipo primitivo tem um tipo referência correspondente
  - Byte, Short, Integer, Long, Float, Double, Boolean, Character
  - Isto permite que as variáveis do tipo primitivo sejam armazenadas como variáveis do tipo referência

```
Integer x = new Integer(10);  
Object y = 20; // o compilador executa implicitamente  
               // new Integer(20)
```

Novos tipos referência podem definidos com `class` e `enum`

- Uma classe é composta por dois tipos de membros
  - Campos
  - Métodos (funções)
  - Por enquanto, vamos considerar as classes similares as estruturas em C
- Um `enum` é um tipo ordinal onde os valores possíveis do tipo são enumerados explicitamente



# Novos tipos

```
class Ponto {  
    int x;  
    int y;  
}
```

```
class Circulo {  
    Ponto centro;  
    int raio;  
    Cor cor;  
}
```

```
enum Cor {  
    VERMELHO,  
    VERDE,  
    AZUL  
}
```

## Novos tipos

```
public class Exemplo {  
    public static void main(String[] args) {  
        // - Todos os campos de uma classe são inicializados com  
        //     valores padrão  
        // - Não é possível alocar uma instância de uma classe na  
        //     pilha. O uso do new é um indicador que a alocação  
        //     ocorre no heap  
        Ponto p = new Ponto();  
        // x e y são inicializados com 0  
        p.x = 3;  
        p.y = 4;  
        Circulo c = new Circulo();  
        // centro e cor são inicializado com null e raio com 0  
        c.centro = p;  
        c.raio = 10;  
        c.cor = Cor.VERMELHO;  
    }  
}
```

# Campos e métodos estáticos

- Os campos podem ser de instâncias ou de classe
  - Cada instância da classe contém um valor para cada campo de instância, portanto, para acessar um campo de instância é necessário uma instância
  - Existe apenas um valor de cada campo estático. Os campos estáticos são similares as variáveis globais em C
  - Por padrão, os campos são de instância, para declarar um campo estático devemos utilizar o modificador `static`

# Campos estáticos

```
class Pessoa {  
    static long proximoId = 1;  
    long id;  
    String nome;  
}
```

# Campos estáticos

```
public class Exemplo {  
    public static void main(String[] args) {  
        // Ok  
        System.out.println(Pessoa.proximoId); // imprimi 1  
  
        // Erro: cada instância de Pessoa tem um valor  
        //       para id, de qual instância é o id  
        //       que estamos tentando acessar?  
        System.out.println(Pessoa.id);  
        // mensagem: non-static variable id cannot be  
        //           referenced from a static context  
  
        Pessoa jose = new Pessoa();  
        jose.nome = "José";  
        jose.id = Pessoa.proximoId;  
        Pessoa.proximoId += 1;  
  
        Pessoa pedro = new Pessoa();  
        pedro.nome = "Pedro";  
        pedro.id = Pessoa.proximoId;  
        Pessoa.proximoId += 1;  
        // pedro.proximoId += 1; // funciona, mas é confuso  
  
        // Ok: id da instância jose  
        System.out.println(jose.id); // imprimi 1  
        // Ok: id da instância pedro  
        System.out.println(pedro.id); // imprimi 2  
    }  
}
```

# Métodos

- As funções em Java são chamadas de métodos
- Todo método é definido dentro de uma classe
- Os métodos podem ser de instância ou de classe
  - Os métodos de instância precisam de uma instância da classe para serem executados. Eles podem acessar e modificar os campos da instância. Veremos mais adiante como eles funcionam
  - Os métodos de classe, também chamados de métodos estáticos, não precisam de uma instância da classe e portanto só podem acessar e modificar os campos estáticos da classe. Os métodos estáticos são invocados de maneira similares as funções em C

- Quando um método é definido, por padrão ele é de instância, para definir um método estático adicionamos o modificador `static`



# Métodos estáticos

```
class Ponto {
    int x;
    int y;

    static double distanciaOrigem(Ponto p) {
        return Math.hypot(p.x, p.y);
    }
}

public class MetodoEstatico {
    public static void main(String[] args) {
        Ponto p = new Ponto();
        p.x = 3;
        p.y = 4;
        double d = Ponto.distanciaOrigem(p);
        // p.distancia_origem(p); funciona, mas é confuso
        System.out.println(d); // imprimi 5.0
    }
}
```

# Passagem de parâmetros

- Todos os parâmetros em Java são passados por valor
- **Mas**, uma cópia de uma referência é similar a uma cópia de ponteiro, o que implica que a semântica da passagem de parâmetro de tipos referência é como a de passagem por referência!

# Passagem de parâmetros

```
import java.util.Arrays;

public class Parametros {
    public static void main(String[] args) {
        int a = 1;
        int[] b = {1, 2, 3};

        Parametros.mudaPrimitivo(a);
        System.out.println(a); // imprimi 1

        Parametros.mudaArray1(b);
        System.out.println(Arrays.toString(b)); // imprimi [10, 2, 3]

        Parametros.mudaArray2(b);
        System.out.println(Arrays.toString(b)); // imprimi [10, 2, 3]
    }

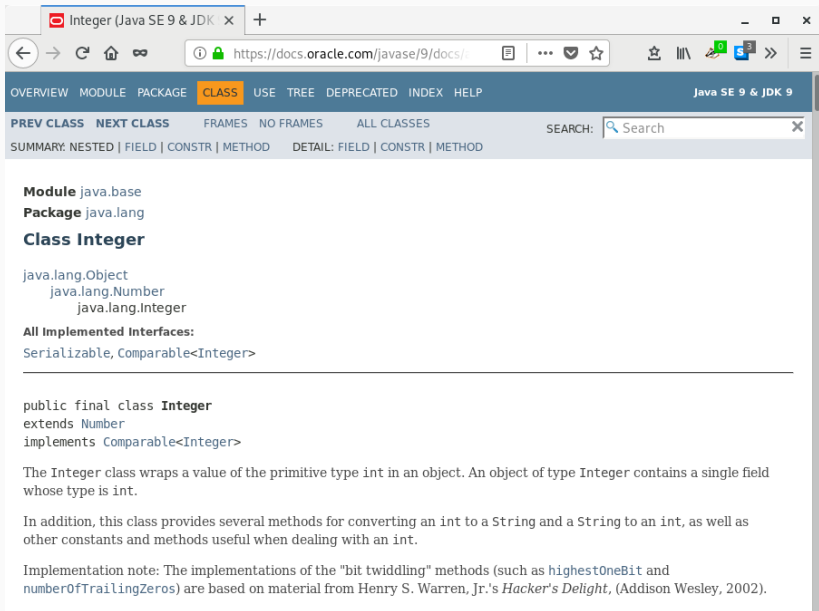
    static void mudaPrimitivo(int x) {
        x = 10;
    }

    static void mudaArray1(int[] x) {
        x[0] = 10;
    }

    static void mudaArray2(int[] x) {
        x = new int[3];
    }
}
```

# Documentação

- Um dos fatores que contribuíram para a popularidade do Java foi a extensa biblioteca padrão e sua documentação
- Embora os IDEs possam mostrar a documentação dos itens durante a escrita do código, o uso de um navegador para exploração das APIs é em geral mais apropriado
- A documentação para o Java 10 está disponível em <https://docs.oracle.com/javase/10>



The screenshot shows a web browser window displaying the Oracle Java SE 9 & JDK 9 documentation for the `Integer` class. The browser's address bar shows the URL `https://docs.oracle.com/javase/9/docs/`. The page navigation includes tabs for OVERVIEW, MODULE, PACKAGE, CLASS (selected), USE, TREE, DEPRECATED, INDEX, and HELP. Below the navigation, there are links for PREVIOUS CLASS, NEXT CLASS, FRAMES, NO FRAMES, and ALL CLASSES. A search bar is also present. The main content area shows the following information:

- Module** `java.base`
- Package** `java.lang`
- Class** `Integer`
- Class hierarchy: `java.lang.Object` → `java.lang.Number` → `java.lang.Integer`
- All Implemented Interfaces:** `Serializable`, `Comparable<Integer>`

---

`public final class Integer`  
`extends Number`  
`implements Comparable<Integer>`

The `Integer` class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

Implementation note: The implementations of the "bit twiddling" methods (such as `highestOneBit` and `numberOfTrailingZeros`) are based on material from Henry S. Warren, Jr.'s *Hacker's Delight*, (Addison Wesley, 2002).

The screenshot shows a web browser window displaying the Oracle Java SE 9 & JDK 9 documentation for the `Integer` class. The page is titled "Integer (Java SE 9 & JDK 9)". The navigation bar includes "OVERVIEW", "MODULE", "PACKAGE", "CLASS" (highlighted), "USE", "TREE", "DEPRECATED", "INDEX", and "HELP". Below the navigation bar, there are links for "PREV CLASS", "NEXT CLASS", "FRAMES", "NO FRAMES", and "ALL CLASSES". A search box is present with the text "SEARCH: Search". The main content area is titled "Field Summary" and contains a sub-section "Fields" with a table of static fields.

Modifier and Type	Field	Description
static int	<code>BYTES</code>	The number of bytes used to represent an int value in two's complement binary form.
static int	<code>MAX_VALUE</code>	A constant holding the maximum value an int can have, $2^{31}-1$ .
static int	<code>MIN_VALUE</code>	A constant holding the minimum value an int can have, $-2^{31}$ .
static int	<code>SIZE</code>	The number of bits used to represent an int value in two's complement binary form.
static Class<Integer> TYPE		The Class instance representing the primitive type int.

Below the field summary, there is a section titled "Constructor Summary" with a sub-section "Constructors" and a table with columns "Constructor" and "Description".

The screenshot shows the Oracle Java SE 9 & JDK 9 documentation page for the `Integer` class. The browser address bar shows `https://docs.oracle.com/javase/9/docs/`. The navigation bar includes links for OVERVIEW, MODULE, PACKAGE, CLASS (highlighted), USE, TREE, DEPRECATED, INDEX, and HELP. Below the navigation bar, there are links for PREVIOUS CLASS, NEXT CLASS, FRAMES, NO FRAMES, and ALL CLASSES, along with a search box. The main content area is divided into two sections: Constructor Summary and Method Summary.

## Constructor Summary

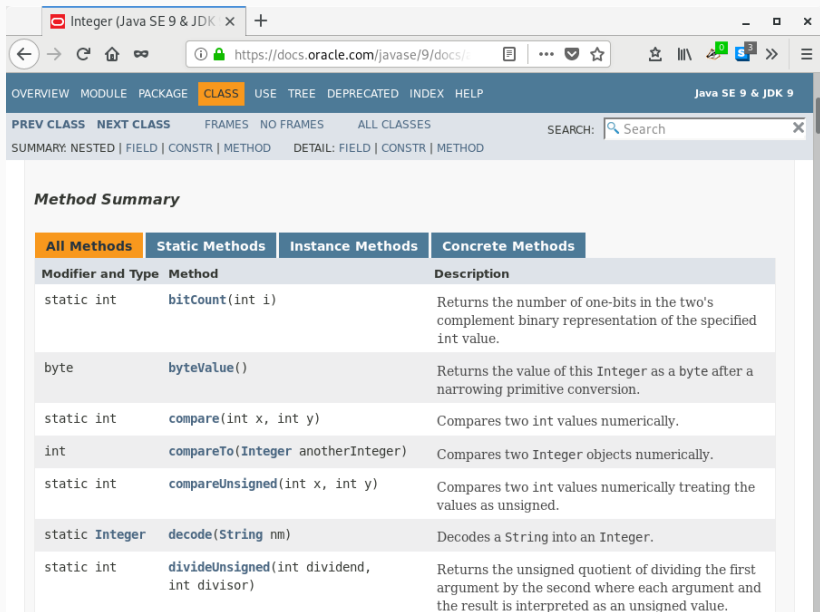
**Constructors**

Constructor	Description
<code>Integer</code> (int value)	<b>Deprecated.</b> It is rarely appropriate to use this constructor. The static factory <code>valueOf(int)</code> is generally a better choice, as it is likely to yield significantly better space and time performance.
<code>Integer</code> (String s)	<b>Deprecated.</b> It is rarely appropriate to use this constructor. Use <code>parseInt(String)</code> to convert a string to a int primitive, or use <code>valueOf(String)</code> to convert a string to an Integer object.

## Method Summary

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type	Method	Description	
static int	<code>bitCount(int i)</code>	Returns the number of one-bits in the two's	





The screenshot shows a web browser window displaying the Oracle Java SE 9 & JDK 9 documentation for the `Integer` class. The page title is "Integer (Java SE 9 & JDK 9)". The navigation bar includes "OVERVIEW", "MODULE", "PACKAGE", "CLASS" (highlighted), "USE", "TREE", "DEPRECATED", "INDEX", and "HELP". Below the navigation bar, there are links for "PREV CLASS", "NEXT CLASS", "FRAMES", "NO FRAMES", and "ALL CLASSES". A search box is present with the text "SEARCH: Search". The main content area is titled "Method Summary" and contains a table with four columns: "All Methods", "Static Methods", "Instance Methods", and "Concrete Methods". The table lists several methods with their modifiers, names, and descriptions.

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type	Method		Description
static int	<code>bitCount(int i)</code>		Returns the number of one-bits in the two's complement binary representation of the specified int value.
byte	<code>byteValue()</code>		Returns the value of this Integer as a byte after a narrowing primitive conversion.
static int	<code>compare(int x, int y)</code>		Compares two int values numerically.
int	<code>compareTo(Integer anotherInteger)</code>		Compares two Integer objects numerically.
static int	<code>compareUnsigned(int x, int y)</code>		Compares two int values numerically treating the values as unsigned.
static Integer	<code>decode(String nm)</code>		Decodes a String into an Integer.
static int	<code>divideUnsigned(int dividend, int divisor)</code>		Returns the unsigned quotient of dividing the first argument by the second where each argument and the result is interpreted as an unsigned value.

The screenshot shows a web browser window with the URL `https://docs.oracle.com/javase/9/docs/`. The page title is "Integer (Java SE 9 & JDK 9)". The navigation bar includes "OVERVIEW", "MODULE", "PACKAGE", "CLASS" (highlighted), "USE", "TREE", "DEPRECATED", "INDEX", and "HELP". Below the navigation bar, there are links for "PREV CLASS", "NEXT CLASS", "FRAMES", "NO FRAMES", and "ALL CLASSES". A search box is present with the text "SEARCH: Search". The main content area is titled "parseInt" and contains the following information:

```
public static int parseInt(String s)
    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('`\u002D`') to indicate a negative value or an ASCII plus sign '+' ('`\u002B`') to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

**Parameters:**  
s - a String containing the int representation to be parsed

**Returns:**  
the integer value represented by the argument in decimal.

**Throws:**  
`NumberFormatException` - if the string does not contain a parsable integer.

Below this, the "parseUnsignedInt" method is partially visible:

```
public static int parseUnsignedInt(String s,
    int radix)
    throws NumberFormatException
```

# Operadores

# Operadores

- Os operadores em Java são praticamente os mesmos do C
- Atenção especial aos operadores == e !=
  - Para os valores de tipos primitivos testa se os operados são iguais ou diferentes
  - Para os valores de tipos referência testa se os operados são iguais ou diferentes, ou seja testa se as referências são iguais ou diferentes, não compara os valores referenciados! Use o método equals para comparar os valores referenciados

```
Integer x = new Integer(10);  
Integer y = new Integer(10);  
System.out.println(x == y); // imprime false  
System.out.println(x != y); // imprime true  
System.out.println(x.equals(y)); // imprime true
```

# Precedência dos operadores

Operador	Precedência
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %= &= ^=  = <<= >>= >>>=

# Controle de fluxo

# if

```
if (condição) {  
    sentenças  
}
```

```
if (condição) {  
    sentenças  
} else {  
    sentenças  
}
```

condição é uma expressão booleana

## switch

```
switch (expressão) {  
    case exp1: sentenças [break;]  
    case exp2: sentenças [break;]  
    ...  
    default: sentenças [break;]  
}
```

Funcionamento semelhante ao da linguagem C

expressão pode ser um tipo inteiro, enumerado ou String



# while

```
while (condição) {  
    sentenças  
}
```

```
do {  
    sentenças  
} while (condição);
```

condição é uma expressão booleana

## for

```
for (inicialização; condição; incremento) {  
    sentenças  
}
```

Equivalente a:

```
inicialização;  
while (condição) {  
    sentenças;  
    incremento;  
}
```

Funcionamento semelhante ao da linguagem C

# for

```
for (tipo nome : exp) {  
    sentenças  
}
```

```
int[] valores = {5, 2, 7, 10};  
for (int x : valores) {  
    ...  
}
```

## Referências

- Learning the Java Language
- Java 10 API