Introdução à linguagem Java

QXD0007 – Programação Orientada a Objetos

Atílio G. Luiz

20 de fevereiro de 2019

Universidade Federal do Ceará – Campus Quixadá

O que é um programa de computador?

- Computadores são ferramentas de uso comum hoje em dia – muitas atividades humanas se beneficiam do uso de computadores:
  - bancos, escolas, universidades, empresas
  - cálculos complexos que exijam rapidez e confiabilidade
  - armazenamento e busca de informações em grandes volumes
  - entretenimento, com jogos e multimídia

O que é um programa de computador?

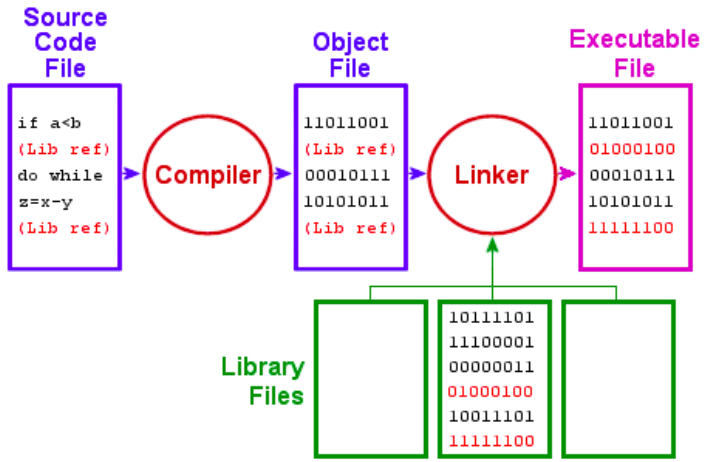
- Computadores são ferramentas de uso comum hoje em dia – muitas atividades humanas se beneficiam do uso de computadores:
  - bancos, escolas, universidades, empresas
  - cálculos complexos que exijam rapidez e confiabilidade
  - armazenamento e busca de informações em grandes volumes
  - entretenimento, com jogos e multimídia
- O que faz um computador ser capaz de realizas estas atividades são os seus programas.
- **Um programa** é um conjunto de instruções que descrevem uma tarefa a ser realizada por um computador.

O que é um programa de computador?

- Computadores são ferramentas de uso comum hoje em dia – muitas atividades humanas se beneficiam do uso de computadores:
  - bancos, escolas, universidades, empresas
  - cálculos complexos que exijam rapidez e confiabilidade
  - armazenamento e busca de informações em grandes volumes
  - entretenimento, com jogos e multimídia
- O que faz um computador ser capaz de realizas estas atividades são os seus programas.
- **Um programa** é um conjunto de instruções que descrevem uma tarefa a ser realizada por um computador.
- Programas são escritos em linguagens de programação, que possuem regras específicas e bem determinadas.

Linguagens compiladas

- Em termos de compilação, existem basicamente dois tipos de linguagem: linguagens **compiladas** e linguagens **interpretadas**.
- Linguagens compiladas compilam diretamente para o código da máquina. C e C ++ são exemplos de linguagens compiladas.



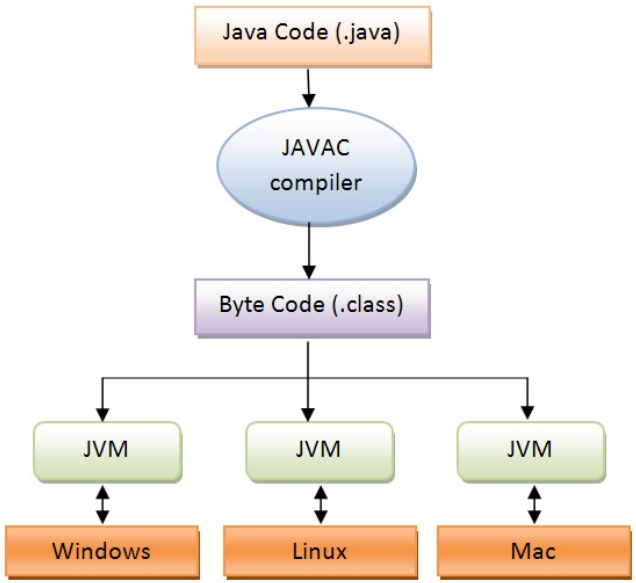
- Código compilado nos dá a melhor velocidade, pois é executado diretamente pelo processador.
- Porém, não temos controle sobre ele. Se for malicioso, pode causar estragos ao ser executado pelo processador.

Linguagens interpretadas

- Um programa conhecido como **Interpretador** lê uma linha do código, verifica se está correto e depois o executa.
- Se encontrar um erro, o programa é interrompido.
- Exemplos de tais linguagens são BASIC, Shell Scripting, Perl, Python, Ruby e Java.
- Linguagens interpretadas demoram mais para executar.
- Porém, o interpretador sabe qual código será executado e pode verificar se esse código pode ou não executar algumas operações maliciosas. Se isso acontecer, ele irá pará-lo ali mesmo.

Máquina Virtual Java

- **Máquina virtual Java** (do inglês *Java Virtual Machine - JVM*) é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina.



- Programas escritos em Java podem funcionar em qualquer plataforma de hardware e software que possua uma versão da JVM, tornando assim essas aplicações independentes da plataforma onde funcionam.

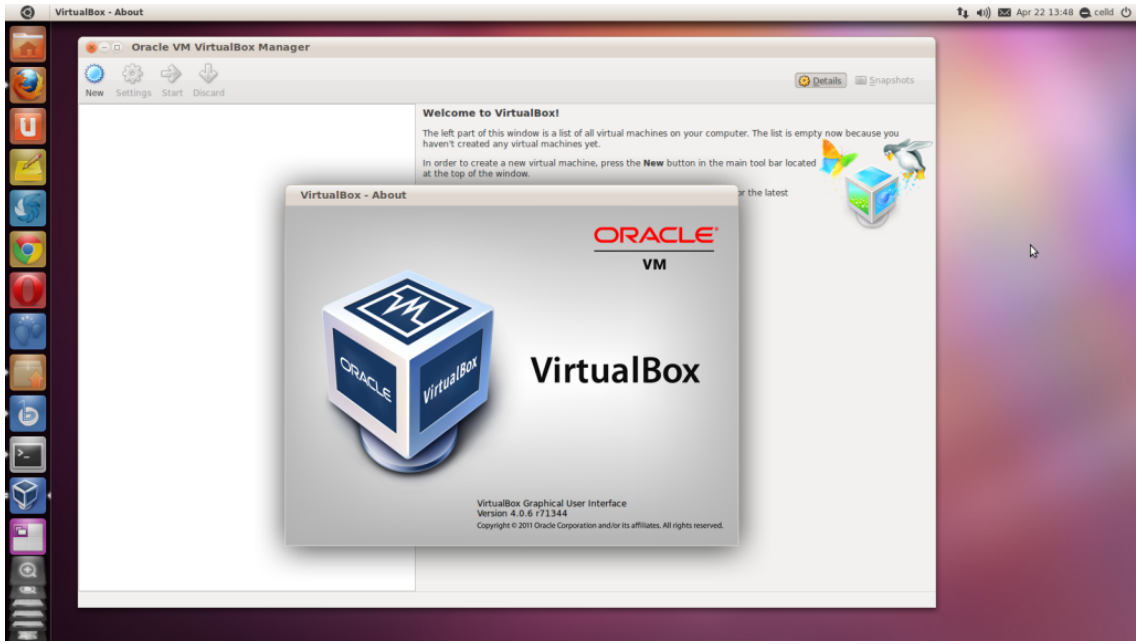
Máquina Virtual Java

A execução do **bytecode** pela JVM pode ser feita de duas maneiras:

- **Interpreter:** a JVM lê cada instrução do bytecode e a executa. A velocidade de execução do código aqui é melhor do que a de outras linguagens interpretadas, mas pior que a das linguagens compiladas.
- **Just In Time Compiler (JIT):** quando a JVM detecta um trecho do código que é executado muitas vezes, ela compila esse trecho do bytecode para código máquina e salva o código compilado para uso futuro. Isso aumenta significativamente a velocidade.
    - Em tempo de execução, o JIT tem informações sobre o processador atual sendo usado e pode aplicar algumas otimizações específicas para esse processador.

Máquinas Virtuais – conceito

- Máquina virtual possui um conceito bem mais amplo que o de um interpretador.
- Ela é responsável por gerenciar memória, threads, a pilha de execução, etc.
- Ela emula um ambiente de computação física, mas requisições de CPU, memória, disco rígido, rede e outros recursos de hardware serão todos geridos por uma "camada de virtualização" que traduz essas solicitações para o hardware presente na máquina.



Ponto forte da linguagem Java

O uso do Java é interessante em:

- aplicações de médio a grande porte, onde o time de desenvolvedores tem várias pessoas e sempre pode vir a mudar e crescer
- aplicações em que a legibilidade de código é importante
- onde há muita conectividade
- e onde há muitas plataformas (ambientes e sistemas operacionais) heterogêneas (Linux, Unix, OSX e Windows misturados)

Versões do Java

Version	Release date	End of Free Public Updates[5][6]	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018
Java SE 7	July 2011	April 2015	July 2022
Java SE 8 (LTS)	March 2014	**January 2019 for Oracle (commercial)** December 2020 for Oracle (personal use) At least September 2023 for AdoptOpenJDK	March 2025
Java SE 9	September 2017	March 2018	N/A
Java SE 10	March 2018	September 2018	N/A
Java SE 11 (LTS)	September 2018	N/A for Oracle At least September 2022 for AdoptOpenJDK	Vendor specific
Java SE 12	March 2019	N/A for Oracle September 2019 for OpenJDK	N/A

Legend: Old version | Older version, still supported | **Latest version** | Future release

Primeiro programa em Java: Olá Mundo

```
1
2 public class HelloWorld {
3     public static void main(String[] args) {
4         System.out.println("Olá Mundo!");
5     }
6 }
```

- Salve o código acima como HelloWorld.java em algum diretório.
- Para compilar, você deve pedir para que o compilador de Java da Oracle, chamado **javac**, gere o bytecode correspondente ao seu código Java:
  `$ javac HelloWorld.java`
- Depois de compilar, um arquivo HelloWorld.class é gerado.
- Para executá-lo, digite:
  `$ java HelloWorld`

Comentários

Há três maneiras de escrever comentários em um programa Java:

- A forma mais comum é usando //. Por exemplo:
  `System.out.println("Ola pessoal"); // Imprime mensagem`

Comentários

Há três maneiras de escrever comentários em um programa Java:

- A forma mais comum é usando //. Por exemplo:
  `System.out.println("Ola pessoal"); // Imprime mensagem`

- Quando comentários mais longos são necessários, podemos usar // para cada linha de comentário ou podemos usar os delimitadores /* e */

```
2 public class HelloWorld {
3     /* Todo programa em Java
4       deve ter uma função main
5      */
6     public static void main(String[] args) {
7         System.out.println("Olá Mundo!"); // Imprime mensagem
8     }
9 }
```

Comentários

- Um terceiro tipo de comentário pode ser usado para gerar documentação automaticamente. Este comentário usa um /** para iniciar e um */ para finalizar.

Comentários

- Um terceiro tipo de comentário pode ser usado para gerar documentação automaticamente. Este comentário usa um /** para iniciar e um */ para finalizar.

```
2  public class HelloWorld {
3      /**
4       * Este é um exemplo de programa simples em Java
5       * Autor: A. G. Luiz
6       * Data: Fevereiro, 2019
7       */
8      public static void main(String[] args) {
9          System.out.println("Olá Mundo!"); // Imprime mensagem
10     }
11 }
```

Comentários

- Um terceiro tipo de comentário pode ser usado para gerar documentação automaticamente. Este comentário usa um /** para iniciar e um */ para finalizar.

```
2  public class HelloWorld {
3      /**
4       * Este é um exemplo de programa simples em Java
5       * Autor: A. G. Luiz
6       * Data: Fevereiro, 2019
7       */
8      public static void main(String[] args) {
9          System.out.println("Olá Mundo!"); // Imprime mensagem
10     }
11 }
```

- Para mais dicas de como escrever código legível e bem documentado, procure por *Java Code conventions*.

Tipos de dados primitivos

- Java é uma linguagem **fortemente tipada**.
- Existem 8 tipos primitivos em Java.

Quatro desses tipos suportam números inteiros:

Type	Storage Requirement	Range (Inclusive)
`int`	4 bytes	–2,147,483,648 to 2,147,483, 647 (just over 2 billion)
`short`	2 bytes	–32,768 to 32,767
`long`	8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`byte`	1 byte	–128 to 127

Tipos de dados primitivos

- Java é uma linguagem **fortemente tipada**.
- Existem 8 tipos primitivos em Java.

Quatro desses tipos suportam números inteiros:

Type	Storage Requirement	Range (Inclusive)
`int`	4 bytes	–2,147,483,648 to 2,147,483, 647 (just over 2 billion)
`short`	2 bytes	–32,768 to 32,767
`long`	8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`byte`	1 byte	–128 to 127

Diferentemente de outras linguagens, em Java, os intervalos dos tipos inteiros não dependem da máquina na qual você estará executando o código.

Tipos de dados primitivos

- **float** e **double**: suportam números de ponto flutuante:

Type	Storage Requirement	Range
`float`	4 bytes	Approximately ±3.40282347E+38F (6–7 significant decimal digits)
`double`	8 bytes	Approximately ±1.79769313486231570E+308 (15 significant decimal digits)

Tipos de dados primitivos

- **boolean** (1 bit): possui dois valores, false, e true e é usado para avaliar condições lógicas. Não é possível conversão entre inteiros e booleanos.

Tipos de dados primitivos

- **boolean** (1 bit): possui dois valores, false, e true e é usado para avaliar condições lógicas. Não é possível conversão entre inteiros e booleanos.
- **char** (2 bytes): esse tipo guarda um, e apenas um, caractere. Esse caractere deve estar entre aspas simples.

  Por exemplo, uma variável char não pode guardar o valor '', pois o vazio não é um caractere; assim como não pode guardar o valor 'casa'.

Tipos de dados primitivos

- **boolean** (1 bit): possui dois valores, false, e true e é usado para avaliar condições lógicas. Não é possível conversão entre inteiros e booleanos.
- **char** (2 bytes): esse tipo guarda um, e apenas um, caractere. Esse caractere deve estar entre aspas simples.

  Por exemplo, uma variável char não pode guardar o valor '', pois o vazio não é um caractere; assim como não pode guardar o valor 'casa'.
- uma variável char suporta todos os caracteres com valores hexadecimais entre `\u0000` e `\uffff`, isto é, com valores inteiros variando de 0 a 65535.

Palavras reservadas do Java

As palavras reservadas têm um significado predefinido na linguagem e, portanto, não podemos usá-las como variáveis, métodos, classes ou como qualquer outro identificador no programa.

Java Keywords

abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum
extends	final	finally	float	for
if	implements	import	instanceof	int
interface	long	native	new	package
private	protected	public	return	short
static	strictfp	super	switch	synchronized
this	throw	throws	transient	try
void	volatile	while		

Keywords that are not currently used

const	goto

Variáveis no Java

O que é uma variável?

- Uma região na memória do computador utilizada para armazenar valores.

Variáveis no Java

O que é uma variável?

- Uma região na memória do computador utilizada para armazenar valores.
- **Variáveis primitivas**: utilizadas para armazenar tipos primitivos.
- **Variáveis de referência**: usadas para referenciar objetos.
  (veremos isso mais adiante)

Variáveis

- Em Java, toda variável tem um tipo. Declaramos uma variável colocando o tipo primeiro, seguido pelo nome da variável.

```
double salario;
int diasFerias;
long populacaoTerra;
boolean flag1, flag2, flag3;
```

- Um nome de variável deve começar com uma letra e deve ser uma sequência de letras ou dígitos. Os termos "letra" e "dígito" são muito mais amplos em Java do que na maioria dos idiomas. Uma letra é definida como 'A' - 'Z', 'a' - 'z', '_', '$', ou qualquer caractere Unicode que indique uma letra em um idioma.
- Nomes de variáveis são *case sensitive*.
- Não podemos usar palavras reservadas do Java como um nome de variável.

Inicialização de variáveis

- Após declarar uma variável, devemos inicializá-la explicitamente por meio de uma instrução de atribuição. Não podemos usar o valor de uma variável não inicializada.

```
2 public class HelloWorld {
3     public static void main(String[] args) {
4         int diasSemana;
5         System.out.println(diasSemana);
6     }
7 }
```

Inicialização de variáveis

- Após declarar uma variável, devemos inicializá-la explicitamente por meio de uma instrução de atribuição. Não podemos usar o valor de uma variável não inicializada.

```
2 public class HelloWorld {
3     public static void main(String[] args) {
4         int diasSemana;
5         System.out.println(diasSemana);
6     }
7 }
```

- Inicializando a variável:

```
2 public class HelloWorld {
3     public static void main(String[] args) {
4         int diasSemana = 7;
5         System.out.println(diasSemana);
6     }
7 }
```

Variáveis com tipos primitivos

- O valor que uma variável de tipo primitivo guarda é o **real conteúdo** da variável.

- Quando utilizamos o operador de atribuição $=$, o valor do lado direito da atribuição é **copiado** para a variável.

```
int i = 5;  // i recebe uma cópia do valor 5
int j = i;  // j recebe uma cópia do valor i
i = i + 1;  // i vira 6, qual o valor de j?
```

Constantes

- Em Java, usamos a palavra-chave **final** antes do nome da variável para denotar uma constante.
- **final** indica que você pode atribuir à variável uma vez e, em seguida, seu valor é definido de uma vez por todas.

```
public class Constants {
    public static void main(String[] args) {
        final double CM_PER_INCH = 2.54;
        double paperWidth = 8.5;
        double paperHeight = 11;
        System.out.println("Paper size in centimeters: "
        + paperWidth * CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);
    }
}
```

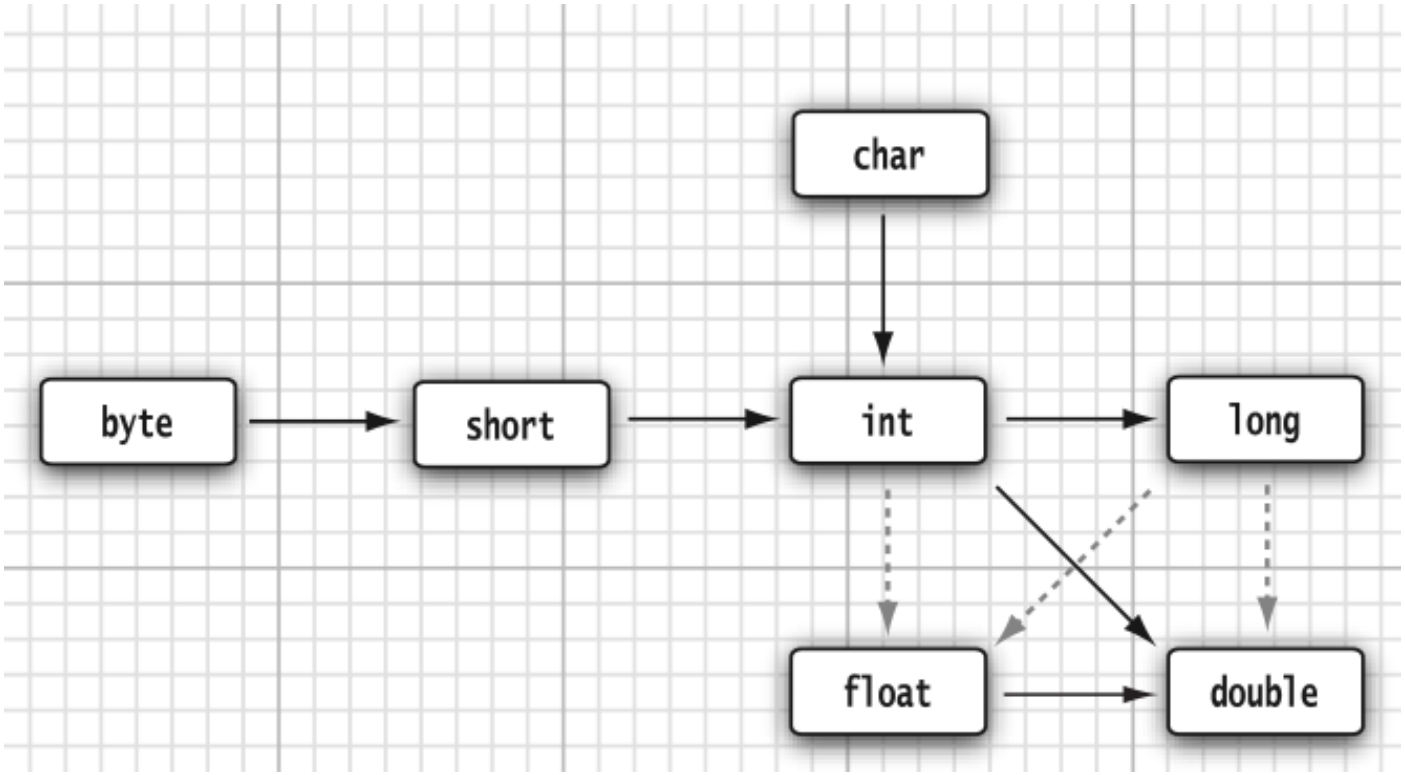
Constantes

- Em Java, usamos a palavra-chave **final** antes do nome da variável para denotar uma constante.
- **final** indica que você pode atribuir à variável uma vez e, em seguida, seu valor é definido de uma vez por todas.

```
public class Constants {
    public static void main(String[] args) {
        final double CM_PER_INCH = 2.54;
        double paperWidth = 8.5;
        double paperHeight = 11;
        System.out.println("Paper size in centimeters: "
        + paperWidth * CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);
    }
}
```

- É costume nomear constantes em letras maiúsculas.
- É comum em Java criar uma constante para que ela esteja disponível para vários métodos dentro de uma única classe. Estes são geralmente chamados de constantes de classe. Configuramos uma constante de classe com as palavras-chave **static final**.

Conversões entre tipos numéricos



- Setas sólidas denotam conversões sem perda de informações.
- Setas pontilhadas denotam conversões que podem perder precisão.
- Por exemplo, um inteiro grande como 123456789 tem mais dígitos do que o tipo float pode representar.

Conversões incompatíveis

- Alguns valores são incompatíveis se tentarmos fazer uma atribuição direta.

- Por exemplo, se tentarmos atribuir um valor do tipo `double` à uma variável do tipo `int`, o nosso código não compilará. Por exemplo:

```
double d = 3.1415;
int p = d;  // não compila
```

```
double d = 5;
int p = d;  // não compila
```

Casting

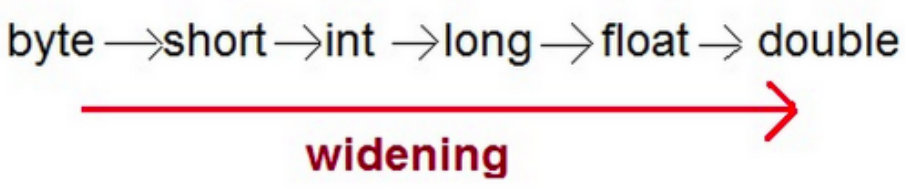
- Situação: às vezes precisamos que um número real seja arredondado e armazenado como um inteiro. Para fazer isso sem que haja erro de compilação, precisamos ordenar que o número real seja **moldado (casted)** como um número inteiro. Esse processo recebe o nome de **casting**.

Casting

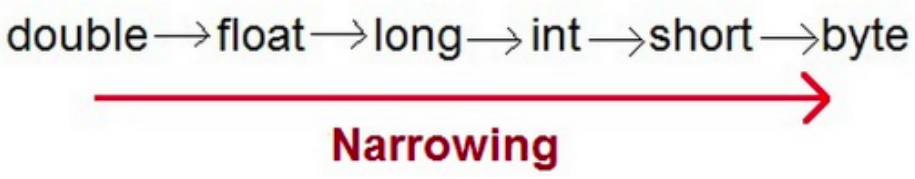
- A sintaxe de um *casting* consiste em fornecer o tipo resultante entre parênteses, seguido pelo nome da variável. Por exemplo:

```java
//Conversão do double 5.0 para float.
float a  = (float) 5.0;

//Conversão de double para int.
int b = (int) 5.1987;
```

- Casting **implícito**:



- Casting **explícito**:



Casting

- O tipo de dado **boolean** é o único tipo primitivo que não suporta casting.

DE \ PARA	***byte***	***short***	***char***	***int***	***long***	***float***	***double***
byte		Implícito	char	Implícito	Implícito	Implícito	Implícito
short	byte		char	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	

Operadores

- **Operadores aritméticos**
  - Utilizados para operações aritméticas
- **Operadores lógicos**
  - Utilizados para operações booleanas
- **Operadores relacionais**
  - Utilizados em processos de comparação

Operadores aritméticos

- Os operadores aritméticos usuais $+$, $-$, $*$, / são usados em Java para adição, subtração, multiplicação e divisão.
- O operador / indica divisão inteira se ambos os argumentos forem inteiros, e divisão de ponto flutuante caso contrário.
- O operador módulo, indicado por %, é o resto da divisão inteira.
- Por exemplo, $15/2 = 7$, $15\%2 = 1$ e $15,0/2 = 7,5$.
- A divisão de inteiros por 0 gera uma exceção, enquanto a divisão de ponto flutuante por 0 produz um resultado infinito (`Double.POSITIVE_INFINITY`, `Double.NEGATIVE_INFINITY`) ou `NaN`.

Operadores aritméticos

Os símbolos $++$ e $--$ são utilizados para incrementar ou decrementar em 1 o valor de uma variável numérica, podendo ser usados das seguintes formas:

- Primeiro incrementa a variável e depois retorna o seu valor.
  $++$<**variável**>**;**

- Primeiro devolve o valor da variável e depois incrementa o seu valor.
  <**variável**> $--$**;**

Quando se necessita realizar uma operação de uma variável com ela própria, acumulando seu valor, basta utilizar:
<**variável**> <**operador**> **=** <**operando**>
Por exemplo:
int num;
num $+=$ 5; *(corresponde a* **num = num + 5;***)*
num $/=$ 8; *(corresponde a* **num = num / 8;***)*

Operadores lógicos

- O símbolo && representa o operador lógico **E** (do inglês AND).
- Este operador retorna **true** somente se os dois operandos tiverem valor **true**.
- **boolean** valor = <operando1> && <operando2>;

p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

Operadores lógicos

- O símbolo || representa o operador lógico **OU** (do inglês OR).
- Este operador retorna **true** caso tenha pelo menos um operando com valor **true**.
- **boolean** valor = <operando1> || <operando2>;

p	q	p∨q
T	T	T
T	F	T
F	T	T
F	F	F

Operadores lógicos

- O símbolo ! representa o operador lógico **negação** (do inglês NOT).
- Este operador retorna **true** se o operando tiver o valor **false** e retorna **false** se o operando tiver o valor **true**.
- **boolean** valor = ! <operando>;

p	~p
T	F
F	T

Exemplo de uso dos operadores lógicos

```java
public class Teste {
    public static void main(String[] args) {
        System.out.println( true && true );
        System.out.println( !(true && true) );
        System.out.println( true || true );
        System.out.println( !(true || true) );
        System.out.println( !true || true );
    }
}
```

Resultado:

Exemplo de uso dos operadores lógicos

```java
public class Teste {
    public static void main(String[] args) {
        System.out.println( true && true );
        System.out.println( !(true && true) );
        System.out.println( true || true );
        System.out.println( !(true || true) );
        System.out.println( !true || true );
    }
}
```

Resultado:

true

Exemplo de uso dos operadores lógicos

```java
public class Teste {
    public static void main(String[] args) {
        System.out.println( true && true );
        System.out.println( !(true && true) );
        System.out.println( true || true );
        System.out.println( !(true || true) );
        System.out.println( !true || true );
    }
}
```

Resultado:

true

false

Exemplo de uso dos operadores lógicos

```java
public class Teste {
    public static void main(String[] args) {
        System.out.println( true && true );
        System.out.println( !(true && true) );
        System.out.println( true || true );
        System.out.println( !(true || true) );
        System.out.println( !true || true );
    }
}
```

Resultado:

true

false

true

Exemplo de uso dos operadores lógicos

```java
public class Teste {
    public static void main(String[] args) {
        System.out.println( true && true );
        System.out.println( !(true && true) );
        System.out.println( true || true );
        System.out.println( !(true || true) );
        System.out.println( !true || true );
    }
}
```

Resultado:

true

false

true

false

Exemplo de uso dos operadores lógicos

```java
public class Teste {
    public static void main(String[] args) {
        System.out.println( true && true );
        System.out.println( !(true && true) );
        System.out.println( true || true );
        System.out.println( !(true || true) );
        System.out.println( !true || true );
    }
}
```

Resultado:

true

false

true

false

true

Operadores relacionais

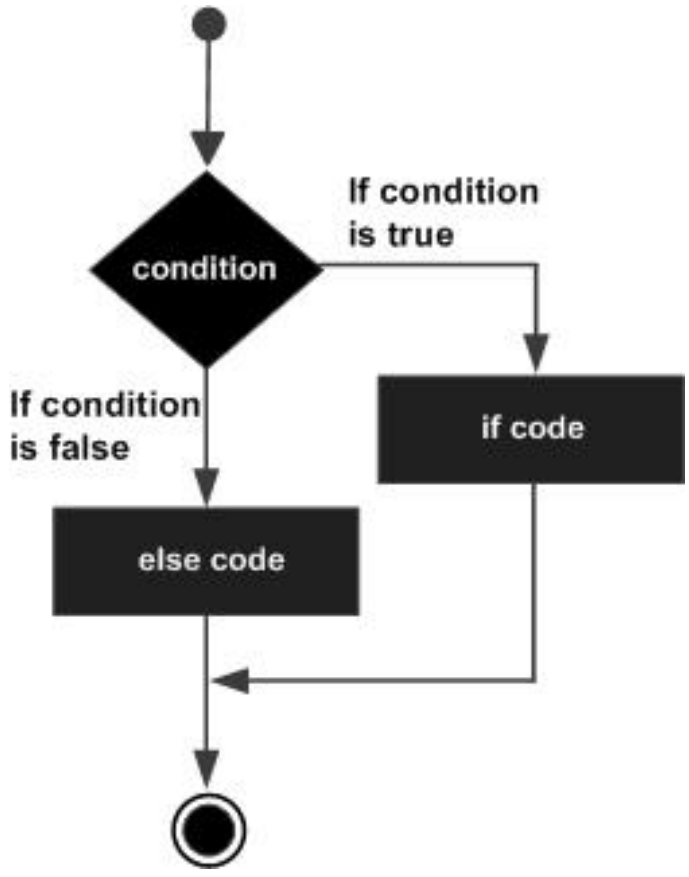
- $>$ (maior que)
- $<$ (menor que)
- $==$ (igualdade)
- $!=$ (diferença)
- $>=$ (maior ou igual que)
- $<=$ (menor ou igual que)

Estruturas condicionais: o IF e o ELSE

A estrutura de controle **if** é utilizada para executar um ou mais comandos apenas quando uma determinada condição for verdadeira **true**.

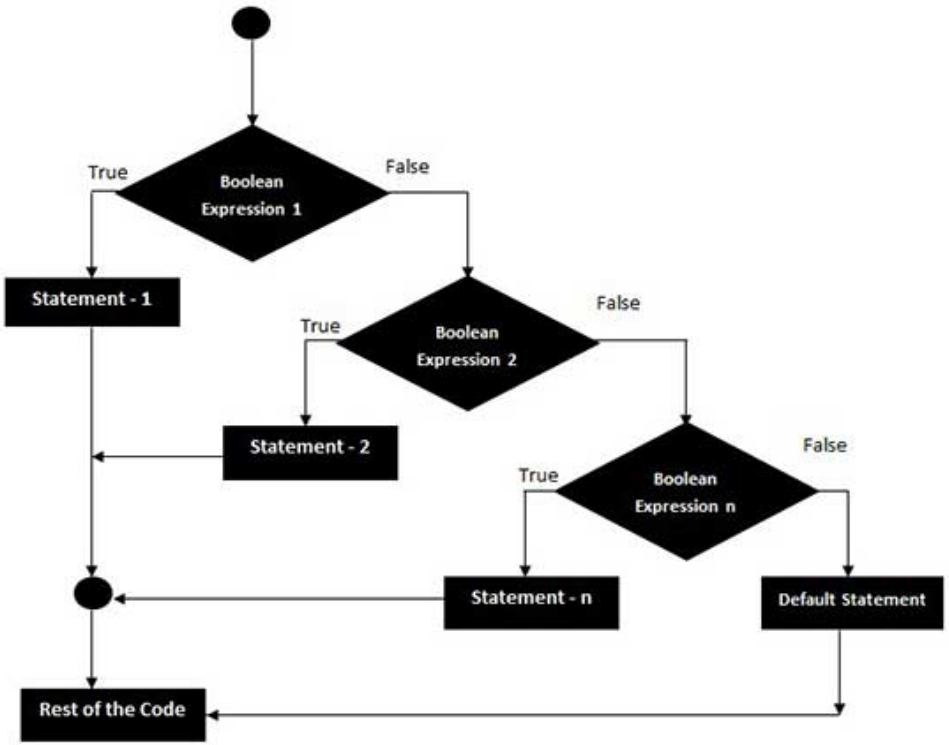
O **else** pode ou não acompanhar o **if**. O **else** é usado para executar um outro bloco de comandos caso a condição do **if** seja falsa (**false**). Exemplo:

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

        int idade = 15;

        if (idade < 18) {
            System.out.println("Não pode entrar");
        } else {
            System.out.println("Pode entrar");
        }
    }
}
```



Estruturas condicionais: o ELSE...IF

```
if(condição 1) {
  // Comandos executados caso a condição 1 verdadeira.
} else if(condição 2) {
 // Comandos executados caso a condicão 2 verdadeira.
} else if(condição 3) {
 // Comandos executados caso a condição 3 verdadeira.
} else {
  // Comandos executados caso nenhuma das condições for verdadeira.
}
```



Laços de repetição (Loops)

Comando **WHILE**

- O comando **while** repete um trecho de código enquanto uma determinada condição for verdadeira (**true**).

```
int idade = 15;

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

Laços de repetição (Loops)

Comando **FOR**

- O comando **for** repete um trecho de código enquanto uma determinada condição for verdadeira. Mas, além disso, o **for** também isola um espaço para inicialização de variáveis e para o modificador dessas variáveis.
- Isso faz com que fiquem mais legíveis as variáveis relacionadas ao loop.

```
for (int i = 0; i < 10; i = i + 1) {
    System.out.println("Olá");
}
```

Controlando loops — Comando BREAK

- Apesar de termos condições de parada nos nossos laços, em algum momento, podemos decidir parar o loop por algum motivo especial sem que o resto do laço seja executado.

```
for (int i = x; i < y; i++) {
    if (i % 9 == 0) {
        System.out.println("Achei um número divisível"+
        "por 19 entre x e y");
        break;
    }
}
```

Controlando loops — Comando CONTINUE

- É possível também obrigar o loop a executar o próximo laço.
- Para isso, usamos o comando **continue**.

```
for (int i = 0; i < 100; i++) {
    if (i > 50 && i < 60) {
        continue;
    }
    System.out.println(i);
}
```

String

- String é uma classe.
- Tipo de classe especial onde instância pode ser declarada como tipos simples:
  - String nome = “Pedro”;
- Comparação:
  - <string1>.equals(<string2>)
  - <string1>.equalsIgnoreCase(<string2>)

String – Funções de conversão de tipos

Úteis para converter uma string em um tipo nativo do Java.

- Short.parseShort(<argumento String>)
- Integer.parseInt(<argumento String>)
- Long.parseLong(<argumento String>)
- Float.parseFloat(<argumento String>)
- Double.parseDouble(<argumento String>)

Exemplo – Strings

```
public class ExemploString {

    public static void main(String args[]) {
        String nome1 = "Pedro";
        String nome2 = "Maria";
        String nome3 = "Pedro";

        if (nome1.equals(nome2))
            System.out.println(nome1+" e "+nome2+" nomes idênticos.");
        else
            System.out.println(nome1+" e "+nome2+" são nomes diferentes.");

        int a = Integer.parseInt("123");
        System.out.println("Integer: "+a);

        float b = Float.parseFloat("56.78");
        System.out.println("Float: "+b);

        double c = Double.parseDouble("34253.67");
        System.out.println("Double: "+c);
    }
}
```

Entrada de dados pelo console

- Classe Scanner
- Sequência de instruções:
  - Criação do objeto de entrada de dados:
    `Scanner <entrada> = new Scanner(System.in);`
  - Para cada leitura teclado:
    `String s = <entrada>.nextLine();`
  - Função `nextLine()` retorna String.

Exemplo — Entrada de dados pelo console

```java
//Demonstrando o funcionamento da classe Scanner do Java
import java.util.Scanner;

public class GetInput {

    public static void main(String args[]) {
        // criação do objeto de entrada de dados
        Scanner entrada = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String s = entrada.nextLine();
        System.out.println("Você digitou a string: "+s);

        System.out.print("Digite um número inteiro: ");
        int a = Integer.parseInt(entrada.nextLine());
        System.out.println("Você digitou o inteiro: "+a);

        System.out.print("Digite um número real: ");
        float b = Float.parseFloat(entrada.nextLine());
        System.out.println("Você digitou o número real: "+b);

        System.out.print("Digite outro número real: ");
        double c = Double.parseDouble(entrada.nextLine());
        System.out.println("Você digitou o número real: "+c);
    }
}
```

Vetores

- Declaração
  $< tipo > [\ ] < declaracao_1 >, \ldots, < declaracao_n >;$
  $< tipo > < declaracao_1 > [\ ], \ldots, < declaracao_n > [\ ];$

- Declaração *inline*
  Sintaxe: $< nome >=< inicializacao >$
  Chaves são usadas para inicializar cada dimensão.
  Exemplo: `int primos[] = 1, 2, 3, 5, 7;`

- Quando a inicialização não for inline o vetor precisa ser instanciado
  $< nome >= new < tipo > [< tamanho >]$
  Exemplo:

```
int primos[];
primos = new int[5];
```

Exemplo — Vetores

```java
public class ExemploVetor {

    public static void main(String args[]) {

        int[] numeros = {1, 2, 3, 4, 5};

        for (int i = 0; i < numeros.length; i++)
            System.out.println(numeros[i]);

        String str[] = {"laura", "marta", "joana"};

        for (int i = 0; i < str.length; i++)
            System.out.println(str[i]);

        double[] reais;
        reais = new double[4];
        reais[0] = 23.4;
        reais[1] = 56.768;
        reais[2] = 12.3;
        reais[3] = 90.7;

        for (int i = 0; i < reais.length; i++)
            System.out.println(reais[i]);
    }
}
```