Introdução ao Java – Parte I

Programação Orientada a Objetos — QXD0007

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

1º semestre/2022

Tópicos desta Aula

- Programas e Paradigmas de Programação
- Compiladores, Interpretadores e Máquinas Virtuais
- Características do Java
- Tipos de dados nativos
- Variáveis, Contantes
- Castings
- Entrada e Saída

Introdução

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:

- O que faz um computador ser capaz de realizar estas atividades são os seus programas.

O que é um programa de computador?

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Computadores são ferramentas de uso comum hoje em dia – muitas atividades humanas se beneficiam do uso de computadores:

- O que faz um computador ser capaz de realizar 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.

Baixo nível × Alto Nível

- Nos primórdios da computação, era necessário usar código de máquina para programar, o que era chamado linguagem de **baixo nível**.
  - cartões perfurados, Assembly
  - propensão a erros
  - baixa produtividade, desmotivação para programar

Baixo nível × Alto Nível

- Nos primórdios da computação, era necessário usar código de máquina para programar, o que era chamado linguagem de **baixo nível**.
  - cartões perfurados, Assembly
  - propensão a erros
  - baixa produtividade, desmotivação para programar

- Posteriormente, surgiram as linguagens de **alto nível**. São chamadas de alto nível porque disponibilizam comandos bem próximos de uma linguagem natural: as **palavras-chave** da linguagem fornecem uma maior clareza de como se deve orquestrar o que um computador deve fazer.

Baixo nível × Alto Nível

- Nos primórdios da computação, era necessário usar código de máquina para programar, o que era chamado linguagem de **baixo nível**.
  - cartões perfurados, Assembly
  - propensão a erros
  - baixa produtividade, desmotivação para programar

- Posteriormente, surgiram as linguagens de **alto nível**. São chamadas de alto nível porque disponibilizam comandos bem próximos de uma linguagem natural: as **palavras-chave** da linguagem fornecem uma maior clareza de como se deve orquestrar o que um computador deve fazer.
- Com as linguagens de alto nível surgiu o conceito de **compilador** e **tempo de compilação**.

Linguagens de alto nível

- O primeiro compilador foi escrito por Grace Hopper, em 1952, para a linguagem de programação A-0.
- A primeira linguagem de programação de alto nível amplamente usada foi Fortran, criada em 1954.

Grace Hopper em 1984

Linguagens de alto nível

- O primeiro compilador foi escrito por Grace Hopper, em 1952, para a linguagem de programação A-0.
- A primeira linguagem de programação de alto nível amplamente usada foi Fortran, criada em 1954.

Grace Hopper em 1984

- COBOL (1959) foi uma linguagem de ampla aceitação para uso comercial.
- ALGOL (1960) teve grande influência no projeto de muitas linguagens posteriores.
- Lisp (1958) tornou-se amplamente utilizada na pesquisa em Inteligência Artificial, assim como o Prolog (1972).
- Simula 67 (1984) introduz o conceito de classes e Smalltalk (1989) se torna a primeira linguagem de programação que oferecia suporte completo à programação orientada a objetos.

Linguagens de alto nível

- Diversas linguagens de programação surgiram desde então. Entre estas incluem-se Pascal, C, C++, Java, C#, VB.NET, Object Pascal, Objective-C, PHP, Python, Ruby e JavaScript.

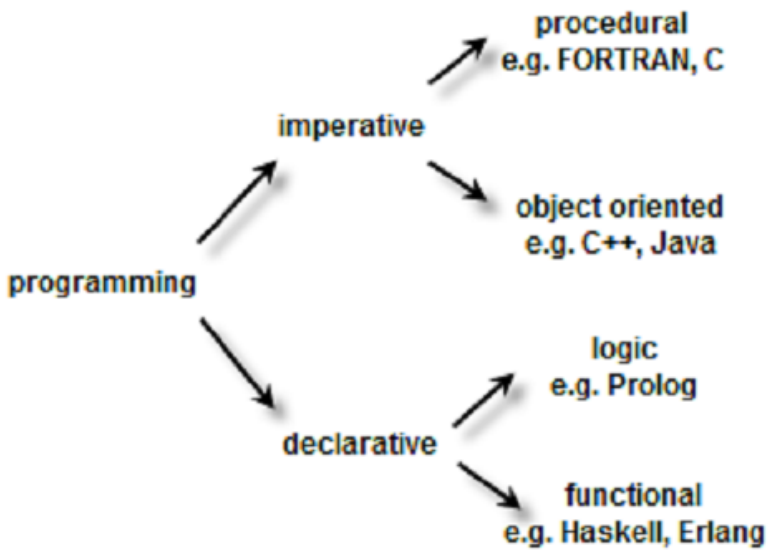
- Hoje, existem diversas linguagens de programação e cada uma possui peculiaridades de acordo com o paradigma de programação que esta implementa.

Paradigmas de Programação

- Um paradigma de programação é um estilo de programação, pode ser entendido como um um modelo ou conjunto de regras que podem ser usadas para solucionar problemas usando uma determinada linguagem de programação.

Paradigmas de Programação

- Um paradigma de programação é um estilo de programação, pode ser entendido como um um modelo ou conjunto de regras que podem ser usadas para solucionar problemas usando uma determinada linguagem de programação.
- Os paradigmas dividem-se em dois grandes grupos:
  - Programação imperativa (o quê)
  - Programação declarativa (como).



Programação Imperativa

- A programação imperativa foi definida por John Von Neumann e preconiza que se deve dar ordens ao computador, ou seja, definir de modo formal e sequencial os passos a serem cumpridos, e assim codificar o algoritmo.

Programação Imperativa

- A programação imperativa foi definida por John Von Neumann e preconiza que se deve dar ordens ao computador, ou seja, definir de modo formal e sequencial os passos a serem cumpridos, e assim codificar o algoritmo.

- **Paradigma Estruturado:** segue a programação imperativa e tornou-se o mais usado para ensinar programação a estudantes.

  - Nesse paradigma, existe uma transição direta entre a descrição do algoritmo e a programação nas linguagens que seguem esse paradigma.
  - Linguagens procedurais notáveis: C e Pascal.

Programação Imperativa

- A programação imperativa foi definida por John Von Neumann e preconiza que se deve dar ordens ao computador, ou seja, definir de modo formal e sequencial os passos a serem cumpridos, e assim codificar o algoritmo.

- **Paradigma Estruturado:** segue a programação imperativa e tornou-se o mais usado para ensinar programação a estudantes.

  - Nesse paradigma, existe uma transição direta entre a descrição do algoritmo e a programação nas linguagens que seguem esse paradigma.
  - Linguagens procedurais notáveis: C e Pascal.

Com o **aumento do poder computacional** e **novas necessidades na automação de processos**, aumentou a demanda por novos paradigmas.

Paradigma Orientado a Objetos

- O Paradigma Orientado a Objetos (POO) tem como principal característica uma melhor e maior expressividade das necessidades do nosso dia-a-dia.

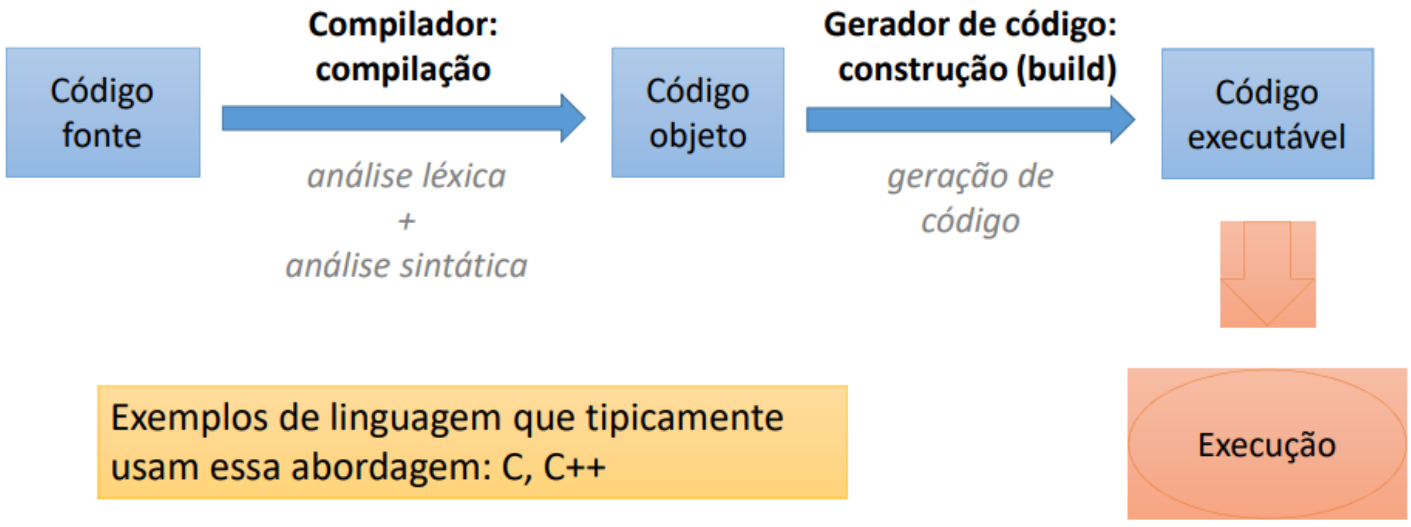
- Possibilita criar unidades de código mais próximas da forma como pensamos e agimos, assim facilitando o processo de transformação das necessidades diárias para uma linguagem orientada a objetos.

- **Java** foi a linguagem de programação que estabeleceu a POO como um paradigma de sucesso no desenvolvimento de software.

Compilação × Interpretação

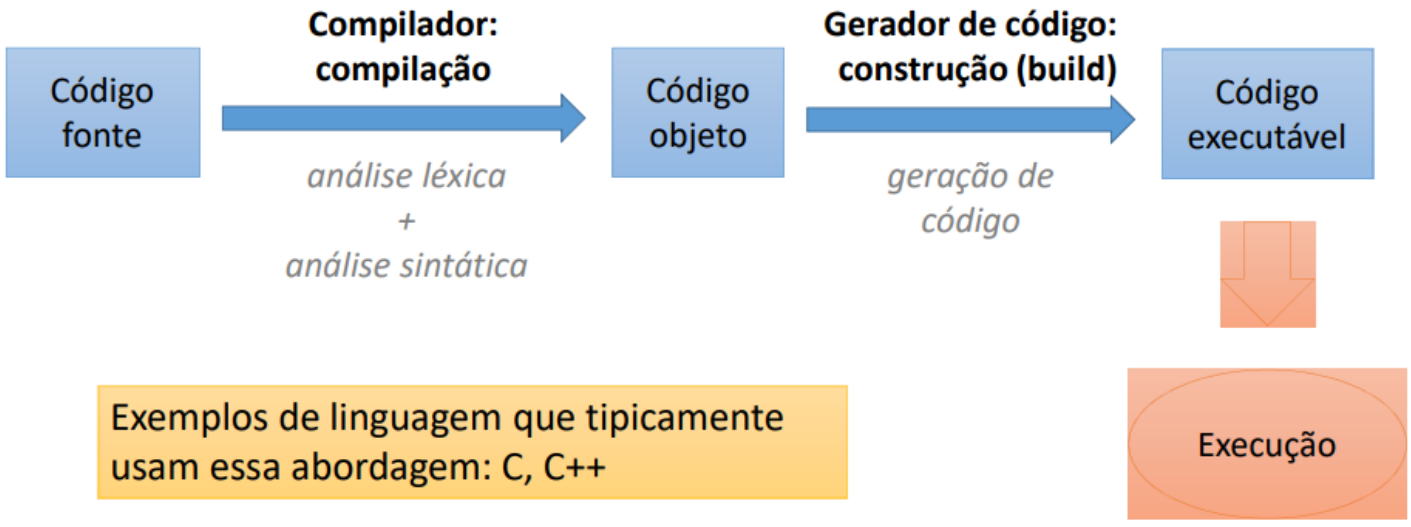
Compilação

- Linguagens compiladas compilam diretamente para o código da máquina. C e C ++ são exemplos de linguagens compiladas.



Compilação

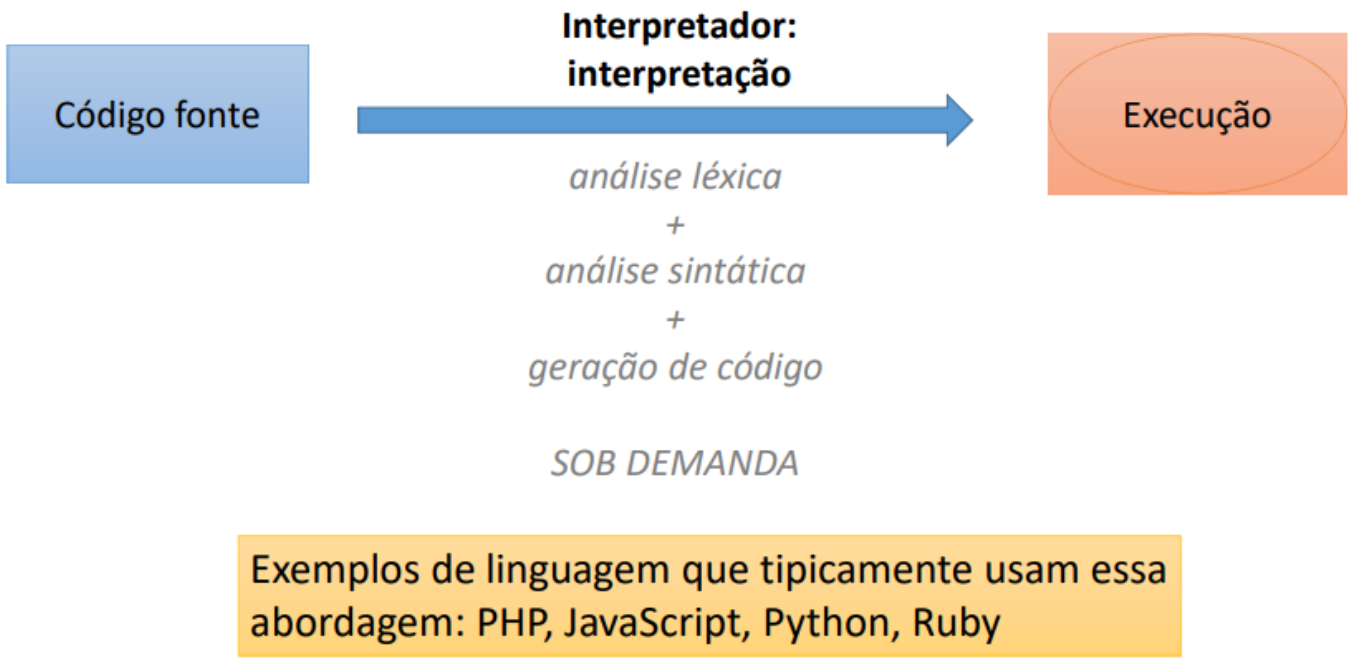
- Linguagens compiladas compilam diretamente para o código da máquina. C e C ++ são exemplos de linguagens compiladas.



- Código compilado nos dá a melhor velocidade, pois é executado diretamente pelo processador.
- Quais as possíveis desvantagens?

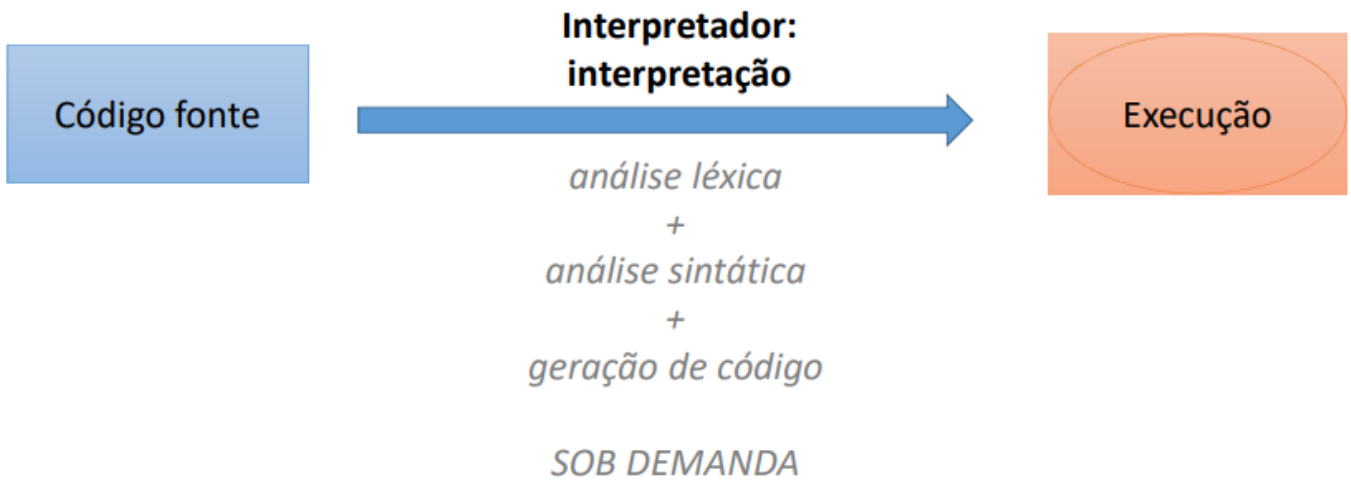
Interpretação

- 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.



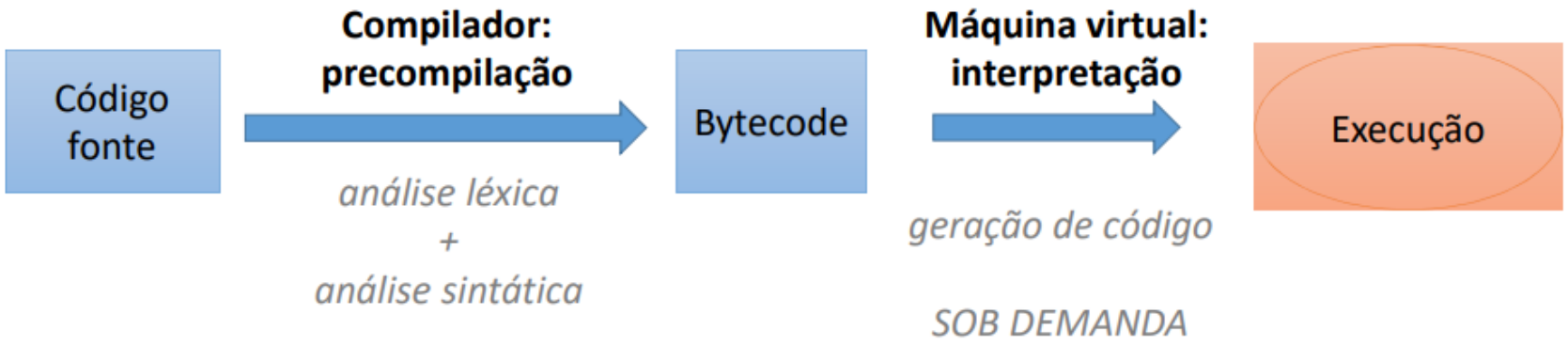
Interpretação

- 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.



- 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.

Abordagem híbrida



Exemplos de linguagem que tipicamente usam essa abordagem: Java (JVM), C# (Microsoft .NET Framework)

Vantagens

Compilação:

- velocidade do programa
- auxílio do computador antes da execução

Interpretação:

- flexibilidade de manutenção do aplicativo em produção
- expressividade da linguagem
- código fonte não precisa ser recompilado para rodar em plataformas diferentes

Vantagens

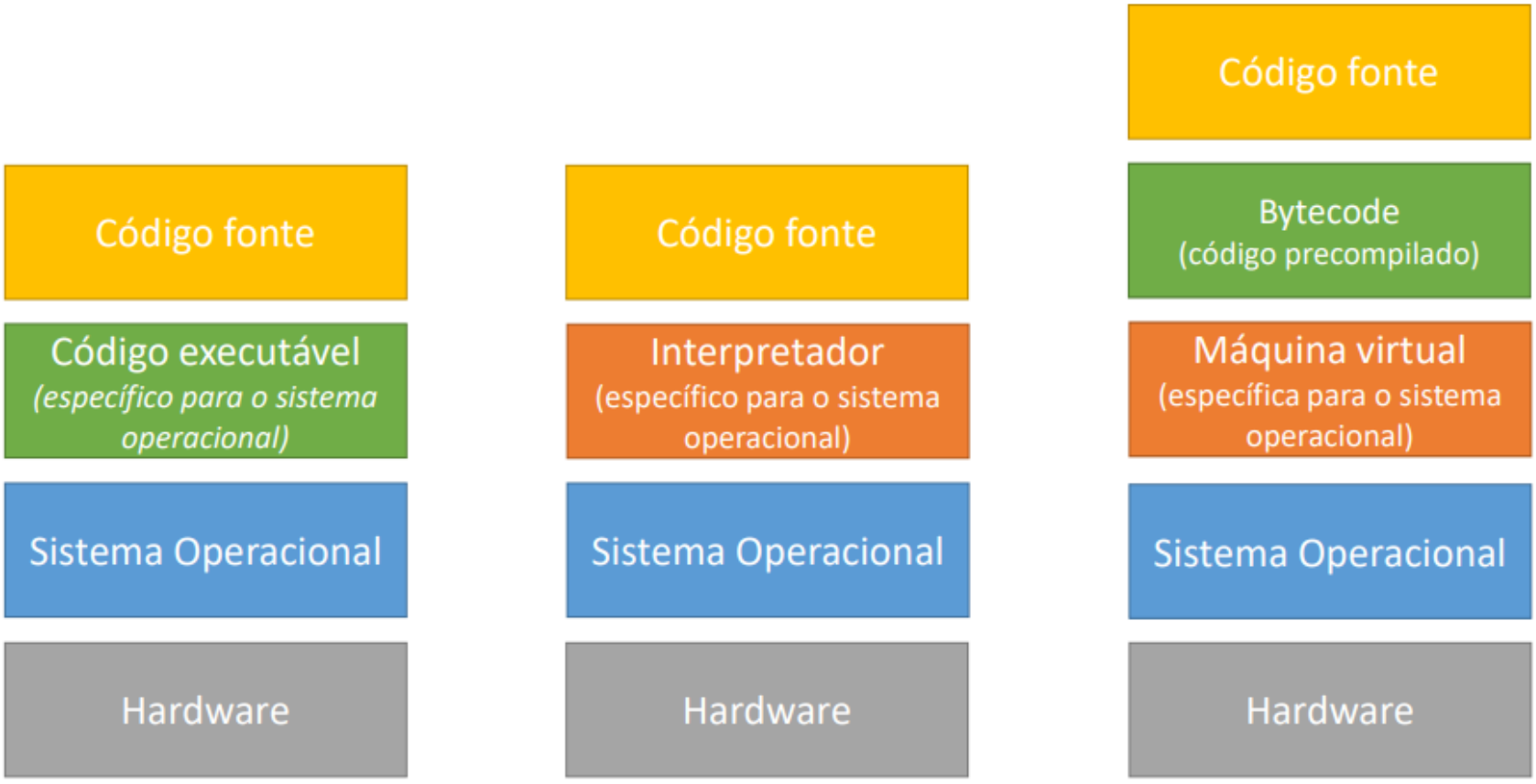
Compilação:

- velocidade do programa
- auxílio do computador antes da execução ⇐ Abordagem híbrida

Interpretação:

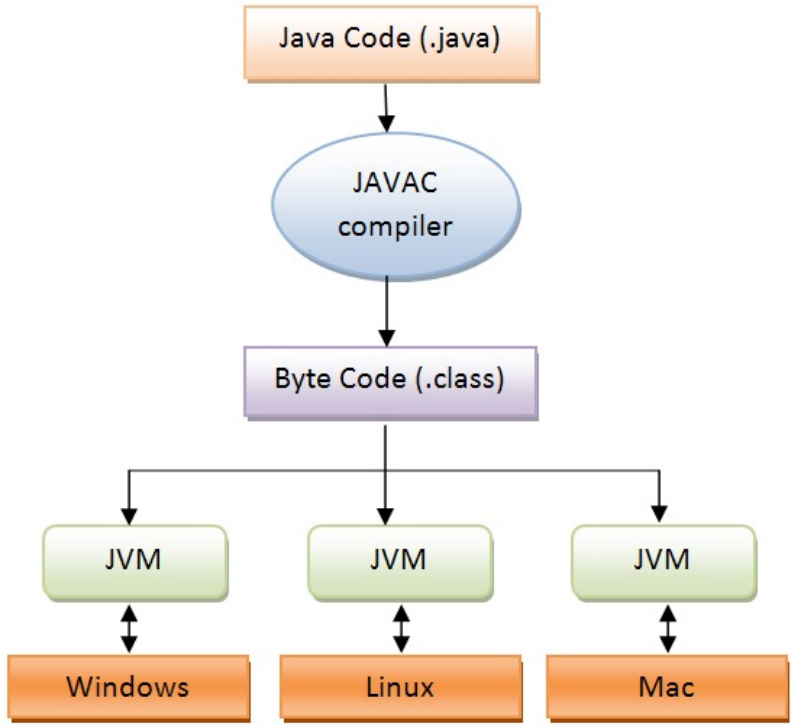
- flexibilidade de manutenção do aplicativo em produção ⇐ Abordagem híbrida
- expressividade da linguagem
- código fonte não precisa ser recompilado para rodar em plataformas diferentes ⇐ Abordagem híbrida

Comparação


Código fonte
Código executável
(específico para o sistema operacional)
Sistema Operacional
Hardware
C / C++
Código fonte
Interpretador
(específico para o sistema operacional)
Sistema Operacional
Hardware
PHP, Python, JavaScript
Código fonte
Bytecode
(código precompilado)
Máquina virtual
(específica para o sistema operacional)
Sistema Operacional
Hardware
Java, C#

Máquina Virtual Java

- **Máquina virtual Java** (do inglês *Java Virtual Machine - JVM*)



- 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.

Iniciando no Java

Nosso Primeiro Código Java

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 // Primeiro programa em Java
2 public class PrimeiroPrograma {
3     public static void main(String[] args) {
4         System.out.println("O primeiro de muitos");
5     }
6 }
```

Nosso Primeiro Código Java

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 // Primeiro programa em Java
2 public class PrimeiroPrograma {
3     public static void main(String[] args) {
4         System.out.println("O primeiro de muitos");
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`
- Para ver como é o bytecode:
  `$ javap -c HelloWorld`

Regras e Convenções do Java


```
1 // Primeiro programa em Java
2 public class PrimeiroPrograma {
3     public static void main(String[] args) {
4         System.out.println("O primeiro de muitos");
5     }
6 }
```

1. O nome de uma classe sempre se inicia com letra maiúscula e, quando necessário, as palavras seguintes também têm seu *case* alterado.
   - Exemplo: PrimeiroPrograma

Regras e Convenções do Java

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```java
// Primeiro programa em Java
public class PrimeiroPrograma {
    public static void main(String[] args) {
        System.out.println("O primeiro de muitos");
    }
}
```

1. O nome de uma classe sempre se inicia com letra maiúscula e, quando necessário, as palavras seguintes também têm seu *case* alterado.
   - Exemplo: PrimeiroPrograma

2. Um arquivo `.java` possui a definição de uma classe. É uma prática recomendada nomear a classe e o arquivo da mesma forma.

Regras e Convenções do Java

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 // Primeiro programa em Java
2 public class PrimeiroPrograma {
3     public static void main(String[] args) {
4         System.out.println("O primeiro de muitos");
5     }
6 }
```

1. O nome de uma classe sempre se inicia com letra maiúscula e, quando necessário, as palavras seguintes também têm seu *case* alterado.
   - Exemplo: PrimeiroPrograma

2. Um arquivo `.java` possui a definição de uma classe. É uma prática recomendada nomear a classe e o arquivo da mesma forma.

3. Java é case sensitive, ou seja, leva em consideração o *case* (caixa) em que as instruções são escritas.

Comentários

1. Há três maneiras de escrever comentários em um programa Java
   - O mais comum é o comentário de linha: //
   - Quando comentários mais longos são necessários, podemos usar // para cada linha de comentário ou podemos usar os delimitadores /* e */
   - Um terceiro tipo de comentário pode ser usado para gerar documentação automaticamente. Este comentário usa um /** para iniciar e um */ para finalizar.

```
1  /**
2   * Este eh o primeiro programa de exemplo
3   * @author Atilio Gomes
4   */
5  public class PrimeiroPrograma { // Classe publica
6      /*
7      Toda classe em Java
8      possui um metodo chamado main
9      */
10     public static void main(String[] args) {
11         System.out.println("O primeiro de muitos");
12     }
13 }
```

Tipos Nativos e Variáveis

Tipos de dados nativos

- Java é uma linguagem **fortemente tipada**: todas as variáveis tem um tipo específico e o tipo da variável é definido no código e, portanto, conhecido/checado em tempo de compilação.

- Existem 8 tipos nativos em Java.

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 nativos

Descrição	Tipo	Tamanho	Valores
tipos numéricos inteiros	byte	8 bits	-128 a 127
	short	16 bits	-32768 a 32767
	int	32 bits	2147483648 a 2147483647
	long	64 bits	9223372036854775808 a 9223372036854775807
tipos numéricos com ponto flutuante	float	32 bits	±3.40282347E+38F (7 dígitos decimais)
	double	64 bits	±1.7976931348623157E+308 (15 dígitos decimais)
um caractere Unicode	char	16 bits	'\u0000' a '\uFFFF'
valor verdade	boolean	1 bit	{false, true}

- **String**: cadeia de caracteres

Veja: `https://unicode-table.com`

Tipos de dados nativos

Um bit pode armazenar 2 valores possíveis (0 ou 1)

Cada bit $=$ 2 possibilidades

8 bits:

$2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 = 2^8 = 256$ possibilidades

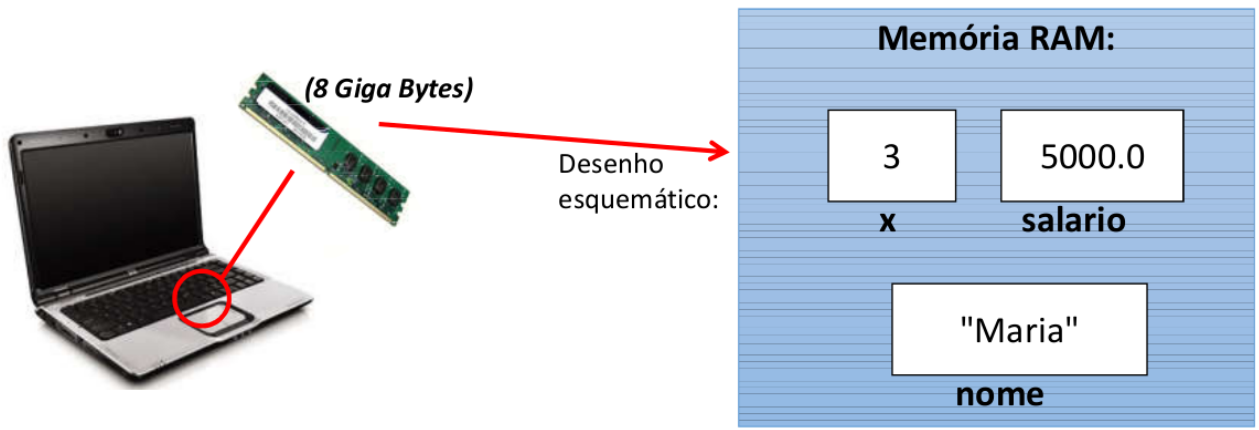
Variáveis

- Um programa de computador em execução lida com dados.

- Esses dados são armazenados em variáveis.

Variáveis

- Um programa de computador em execução lida com dados.
- Esses dados são armazenados em variáveis.
- Uma variável é uma região na memória do computador, utilizada para armazenar dados.
  - Essa região é acessada no nosso programa por meio de um identificador (nome dado à variável).



Declaração de Variáveis

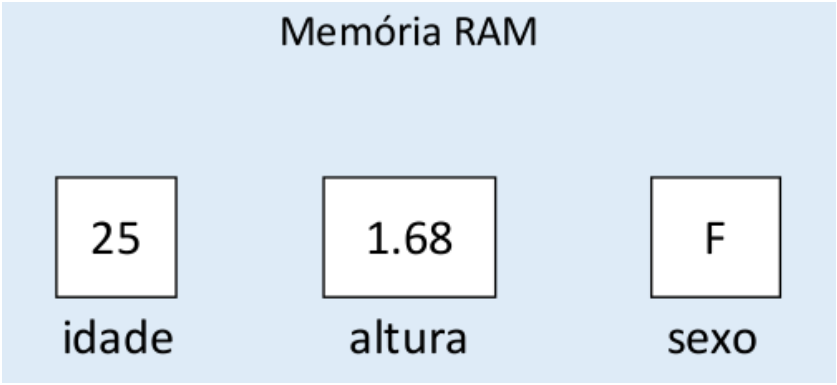
Sintaxe:

$$\langle \text{ tipo } \rangle \ \langle \text{ nome } \rangle \underbrace{= \langle \text{ valor inicial } \rangle}_{\text{opcional}};$$

Declaração de Variáveis

Sintaxe:

$$\langle \text{ tipo } \rangle \ \langle \text{ nome } \rangle \underbrace{= \langle \text{ valor inicial } \rangle}_{\text{opcional}};$$



Exemplos:

```
int idade = 25;
double altura = 1.68;
char sexo = 'F';
```

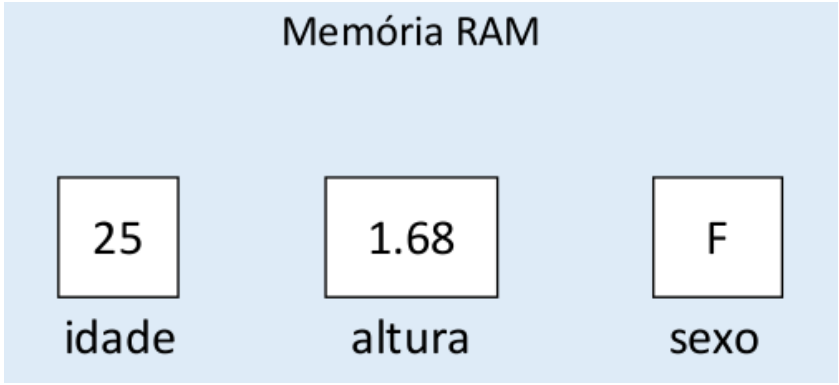
Declaração de Variáveis

Sintaxe:

$$\langle \text{ tipo } \rangle \ \langle \text{ nome } \rangle \underbrace{= \langle \text{ valor inicial } \rangle}_{\text{opcional}};$$

Exemplos:

```
int idade = 25;
double altura = 1.68;
char sexo = 'F';
```



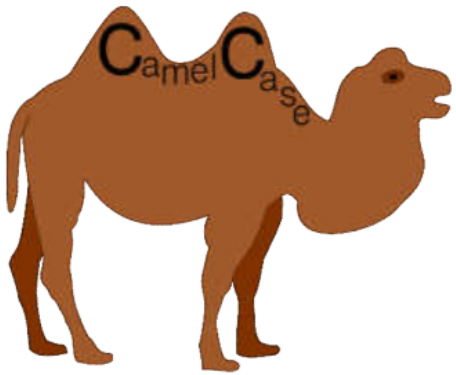
Uma variável possui:

- Nome (ou identificador)
- Tipo
- Valor
- Endereço

Convenções no nome de variáveis

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Não deve começar com dígito: use uma letra ou _
- Não pode ter espaço em branco
- Não usar acentos
- Nomes de variáveis são *case sensitive*
- Não usar palavras reservadas do Java
- Sugestão: use o padrão camel case
- Sugestão: iniciar com letra, que seja minúscula



Errado:

```
int 5minutos;
int salário;
int salário do funcionário;
```

Correto:

```
int _5minutos;
int salario;
int salarioDoFuncionario;
```

Exemplo – Variáveis


```
1  public class Programa04 {
2      public static void main(String[] args) {
3          int diasDaSemana;
4          diasDaSemana = 7;
5          System.out.println("Dias: " + diasDaSemana);
6
7          int meses = 12;
8          System.out.println("Meses: " + meses);
9      }
10 }
```

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.

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.

```
1  public class Programa05 {
2    public static void main(String[] args) {
3      final double CM_POR_POLEGADA = 2.54; // uma constante
4      double larguraPapel = 8.5;
5      double alturaPapel = 11;
6      System.out.println("Tamanho do papel em centimetros: "
7          + larguraPapel * CM_POR_POLEGADA + " por "
8          + alturaPapel * CM_POR_POLEGADA);
9    }
10 }
```

- É uma prática comum em Java nomear constantes em letras maiúsculas.

Operadores Aritméticos

Expressões aritméticas

- São expressões que, quando calculadas, resultam em um valor numérico.

Operadores aritméticos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

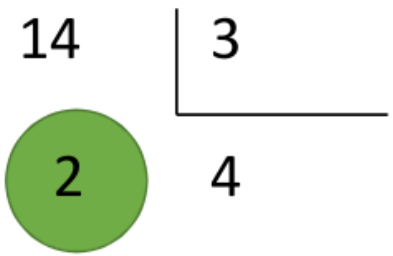
Operador	Significado
$+$	**adição**
$-$	**subtração**
$*$	**multiplicação**
$/$	**divisão**
$\%$	**resto da divisão (mod)**

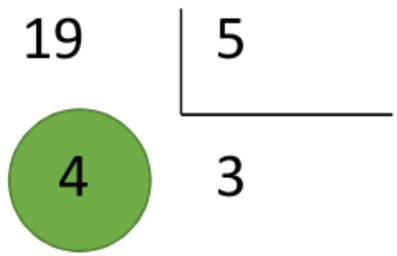
Precedência:

- 1º Lugar: $*$, $/$, $\%$
- 2º Lugar: $+$, $-$

Exemplos com o operador módulo %

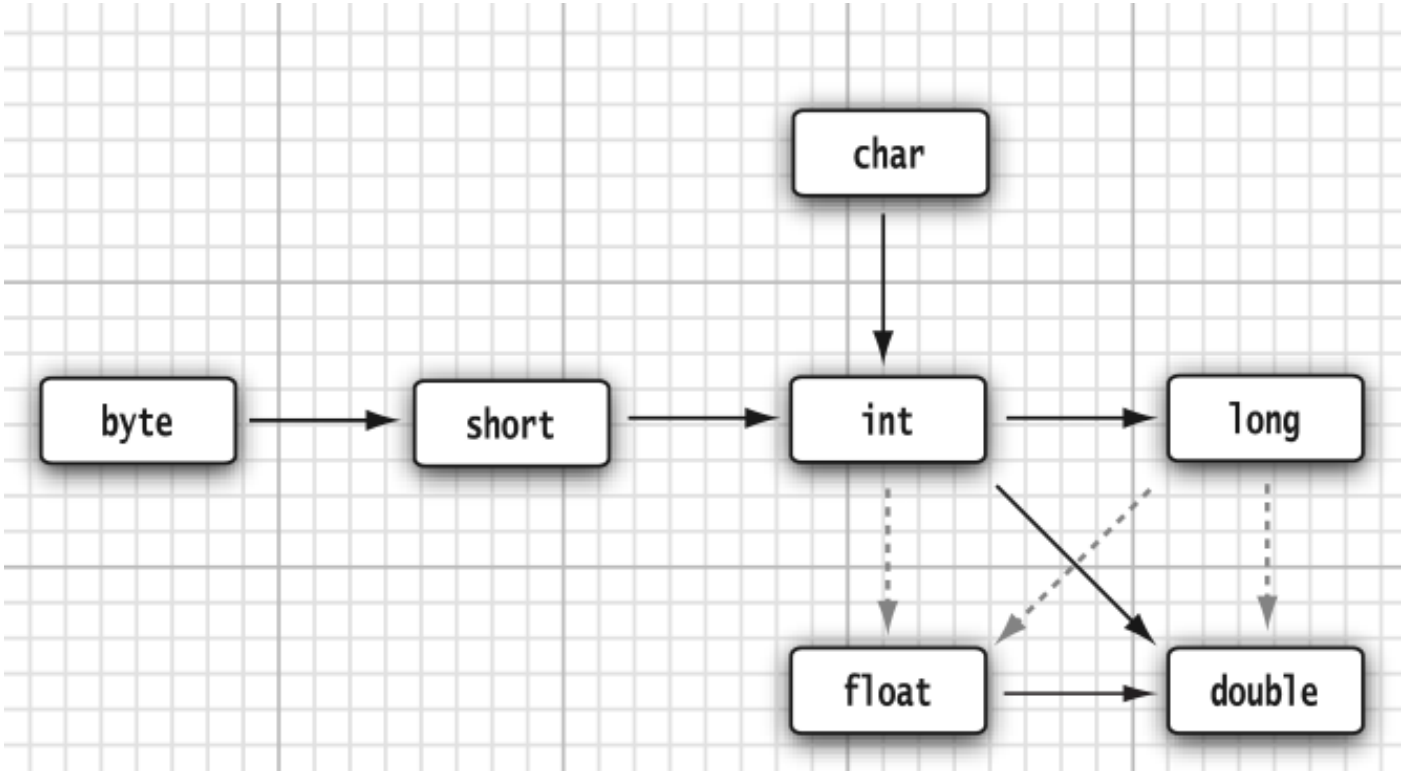
- $14 \,\%\, 3 \;=\; 2$
- $19 \,\%\, 5 \;=\; 4$





Conversões de Tipo

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.

Tipos de dados nativos

Descrição	Tipo	Tamanho	Valores
tipos numéricos inteiros	byte	8 bits	-128 a 127
	short	16 bits	-32768 a 32767
	int	32 bits	2147483648 a 2147483647
	long	64 bits	9223372036854775808 a 9223372036854775807
tipos numéricos com ponto flutuante	float	32 bits	±3.40282347E+38F (7 dígitos decimais)
	double	64 bits	±1.79769313486231570E+308 (15 dígitos decimais)
um caractere Unicode	char	16 bits	'\u0000' a '\uFFFF'
valor verdade	boolean	1 bit	{false, true}

Conversões entre tipos numéricos


```
1 public class Programa06 {
2     public static void main(String[] args) {
3         int n = 123456789;
4         System.out.println(n);
5         float f = n;              // f = 1.234567892E8
6         System.out.println(f);
7     }
8 }
```

Conversões entre tipos numéricos

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1 public class Programa06 {
2     public static void main(String[] args) {
3         int n = 123456789;
4         System.out.println(n);
5         float f = n;              // f = 1.234567892E8
6         System.out.println(f);
7     }
8 }
```

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

```
1 public class Programa07 {
2     public static void main(String[] args) {
3         double pi = 3.1415;
4         int p = pi;     // erro de compilacao
5         float q = pi;   // erro de compilacao
6         System.out.println(q);
7         System.out.println(p);
8     }
9 }
```

Conversões incompatíveis

- 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.

Conversões incompatíveis

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

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

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

Conversões incompatíveis

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

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

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

- Casting **implícito**:
  byte $\rightarrow$ short $\rightarrow$ int $\rightarrow$ long $\rightarrow$ float $\rightarrow$ double
- Casting **explícito**:
  double $\rightarrow$ float $\rightarrow$ long $\rightarrow$ int $\rightarrow$ short $\rightarrow$ byte

Divisão de inteiros

- O que será impresso pelo programa abaixo?

```
1  import java.util.Locale;
2
3  class Divisao {
4      public static void main(String[] args) {
5          Locale.setDefault(Locale.US);
6          int a = 10;
7          int b = 4;
8          double res = (double) a / b;
9          System.out.printf("Resultado = %.2f%n", res);
10     }
11 }
```

Entrada de dados (via Terminal)

Entrada – Usando a classe Scanner

- O fluxo de entrada padrão do Java é dado pelo objeto System.in

Entrada – Usando a classe Scanner

- O fluxo de entrada padrão do Java é dado pelo objeto System.in

- Uma forma de ler da entrada padrão consiste em, inicialmente, construir um objeto Scanner e associar a ele o objeto System.in, assim:

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

  Depois, usar um dos vários métodos da classe Scanner para ler a entrada.

Entrada – Usando a classe Scanner

- O fluxo de entrada padrão do Java é dado pelo objeto System.in

- Uma forma de ler da entrada padrão consiste em, inicialmente, construir um objeto Scanner e associar a ele o objeto System.in, assim:

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

  Depois, usar um dos vários métodos da classe Scanner para ler a entrada.

- Para usar a classe Scanner, é preciso importar o pacote `java.util.Scanner`, logo no início do programa

```
import java.util.Scanner;
```

Entrada – Usando a classe Scanner

- O fluxo de entrada padrão do Java é dado pelo objeto System.in

- Uma forma de ler da entrada padrão consiste em, inicialmente, construir um objeto Scanner e associar a ele o objeto System.in, assim:

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

  Depois, usar um dos vários métodos da classe Scanner para ler a entrada.

- Para usar a classe Scanner, é preciso importar o pacote `java.util.Scanner`, logo no início do programa

```
import java.util.Scanner;
```

- Após finalizar a leitura dos dados, pode-se fechar o Scanner com o seguinte comando:

```
input.close();
```

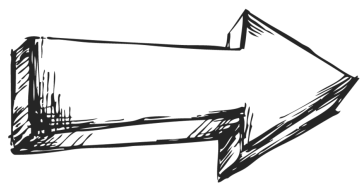
Ler uma palavra (texto sem espaços)

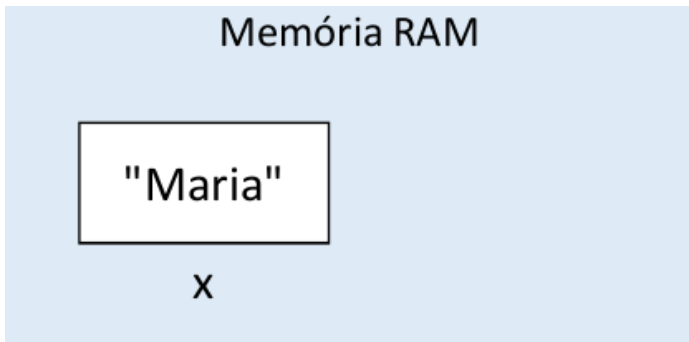
Universidade Federal do Ceará
Campus Quixadá

Suponha uma variável do tipo **String** declarada:

String x;

x = input.next();





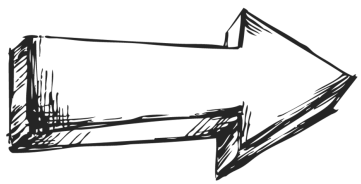
Ler um número inteiro

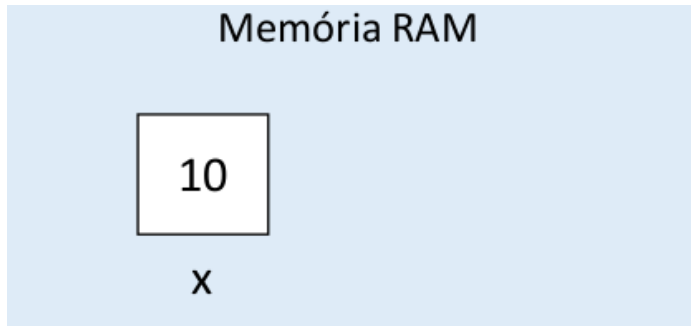
Universidade Federal do Ceará
Campus Quixadá

Suponha uma variável do tipo **int** declarada:

int x;

x = input.nextInt();





Ler um número com ponto flutuante

Suponha uma variável do tipo **double** declarada:

double x;

x = input.nextDouble(); $\Longleftarrow$ Localidade do sistema

Atenção:
Para considerar o separador de decimais como ponto, ANTES da declaração do Scanner, faça:
Locale.setDefault(Locale.US);

Ler um caractere

Suponha uma variável do tipo **char** declarada:

char x;

x = input.next().charAt(0);

Ler um texto até a quebra de linha


```java
import java.util.Scanner;

class Programa17 {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        String s1, s2, s3;

        s1 = input.nextLine();
        s2 = input.nextLine();
        s3 = input.nextLine();

        System.out.println("Dados Digitados:");
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);

        input.close();
    }
}
```

Quebra de linha pendente

Universidade Federal do Ceará
Campus Quixadá

Quando você usa um comando de leitura diferente do nextLine() e dá uma quebra de linha, essa quebra de linha fica "pendente" na entrada padrão.

A seguir, ao invocar o nextLine(), aquela quebra de linha pendente será absorvida pelo nextLine().

```
1  int x;
2  String s1, s2, s3;
3
4  x = input.nextInt();
5  s1 = input.nextLine();
6  s2 = input.nextLine();
7  s3 = input.nextLine();
8
9  System.out.println("Dados digitados:");
10 System.out.println(x);
11 System.out.println(s1);
12 System.out.println(s2);
13 System.out.println(s3);
```

Quebra de linha pendente

Quando você usa um comando de leitura diferente do nextLine() e dá uma quebra de linha, essa quebra de linha fica "pendente" na entrada padrão.

A seguir, ao invocar o nextLine(), aquela quebra de linha pendente será absorvida pelo nextLine().

```
1  int x;
2  String s1, s2, s3;
3
4  x = input.nextInt();
5  s1 = input.nextLine();
6  s2 = input.nextLine();
7  s3 = input.nextLine();
8
9  System.out.println("Dados digitados:");
10 System.out.println(x);
11 System.out.println(s1);
12 System.out.println(s2);
13 System.out.println(s3);
```

Solução: Faça um nextLine() extra antes de fazer o nextLine() de seu interesse.

Entrada – Exemplo


```
1  import java.util.Scanner;
2
3  public class Programa11 {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6
7          System.out.print("Qual eh o seu nome? ");
8          String nome = input.nextLine();
9
10         System.out.print("Quantos anos voce tem? ");
11         int idade = input.nextInt();
12
13         System.out.println("Ola, " + nome +
14             ". Proximo ano, voce tera " +
15             (idade + 1) + " anos");
16
17         input.close(); // fecha o Scanner
18     }
19 }
```

Entrada – Métodos mais usados

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Method	Data Type	Description
nextInt()	Int	It takes int type input value from the user.
nextFloat()	Float	It takes a float type input value from the user.
nextBoolean()	Boolean	It takes a boolean type input value from the user.
nextLine()	String	It takes a line as an input value from the user.
next()	String	It takes a word as an input value from the user.
nextByte()	Byte	It takes a byte type of input value from the user.
nextDouble()	Double	It takes a double type input value from the user.
nextShort()	Short	It takes a short type input value from the user.
nextLong()	Long	It takes a long type of input value from the user.

Mais informações sobre os métodos dessa classe podem ser encontrados na API do Java: `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Scanner.html`

Saída de dados (via Terminal)

Saída

- Em Java, podemos usar os seguintes comandos para escrever no terminal:
  - `System.out.print()`
    imprime uma string
  - `System.out.println()`
    imprime uma string e move o cursor para a próxima linha
  - `System.out.printf()`
    imprime uma string formatada (similar à do C/C++)

Para escrever na tela um texto qualquer

Sem quebra de linha ao final:

```
System.out.print("Bom dia!");
```

Com quebra de linha ao final:

```
System.out.println("Bom dia!");
```

Para escrever na tela um texto qualquer

Sem quebra de linha ao final:

System.out.print(“Bom dia!”);

Com quebra de linha ao final:

System.out.println(“Bom dia!”);

Podemos também concatenar vários elementos no mesmo comando de escrita

- Regra geral para **print** e **println**:
  - elemento1 + elemento2 + $\cdots$ + elementoN

Para escrever na tela um texto qualquer

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Sem quebra de linha ao final:

System.out.print(“Bom dia!”);

Com quebra de linha ao final:

System.out.println(“Bom dia!”);

Podemos também concatenar vários elementos no mesmo comando de escrita

- Regra geral para **print** e **println**:
  - elemento1 + elemento2 + $\cdots$ + elementoN
- System.out.println(“Resultado = ” + x + “metros”)

Imprimir conteúdo de uma variável com ponto flutuante

Suponha uma variável tipo **double** declarada e iniciada:

double x = 10.35784;

- System.out.println(x);
- System.out.printf("%.2f%n", x);
- System.out.printf("%.4f%n", x);

Atenção:
Para considerar o separador de decimais como ponto, ANTES da declaração do Scanner, faça:

Locale.setDefault(Locale.US);

Saída formatada — printf()

Sintaxe:

System.out.printf(“format-string” [,arg1, arg2, ...]);

Saída formatada — printf()

Sintaxe:
System.out.printf("format-string" [,arg1, arg2, ...]);

Format string: Composta de literais e especificadores de formato. Argumentos são requeridos somente se existem especificadores na string de formatação. São especificadores de formato: flags, largura, precisão e caracteres de conversão, na seguinte sequência:

% [flags] [largura] [.precisão] caractere-de-conversão

Os colchetes denotam parâmetros opcionais

Saída formatada — printf()

Sintaxe:
System.out.printf("format-string" [,arg1, arg2, ...]);

Format-String: % [flags] [largura] [.precisão] caractere-de-conversão

Caracteres de Conversão:

- d : inteiro decimal [byte, short, int, long]
- f : número de ponto flutuante [float, double]
- c : caractere
- s : String
- x : inteiro hexadecimal
- o : inteiro octal
- e : número de ponto flutuante em notação científica
- h : Hashcode
- n : nova linha
- b : booleano
- % : símbolo de porcentagem

Saída formatada — printf()

Universidade Federal do Ceará
Campus Quixadá

Sintaxe:
System.out.printf("format-string" [,arg1, arg2, ...]);

% [flags] [largura] [.precisão] caractere-de-conversão

Flags:
- $-$: alinhamento à esquerda (default é à direita)
- $+$: imprime um sinal de (+) mais ou (-) menos para um valor numérico
- 0 : preenche um número com zeros
- : espaço mostrará um sinal de menos se o número for negativo ou espaço se for positivo

Saída formatada — printf()

Sintaxe:
System.out.printf("format-string" [,arg1, arg2, ...]);

Format-String: % [flags] [largura] [.precisão] caractere-de-conversão

Largura:
Especifica a largura do campo em que o argumento será impresso e especifica o número mínimo de caracteres a serem escritos na saída.

Precisão:
Especifica o número de dígitos de precisão ao imprimir números de ponto flutuante. Números são arredondados para a precisão especificada.

Saída formatada — Exemplo com printf()

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  public class Programa12 {
2      public static void main(String[] args) {
3          double a = 10, b = 7;
4          double aux = a/b;
5          System.out.println(aux);
6          System.out.printf("aux: %f%n", aux);
7          System.out.printf("aux: %+f%n", aux);
8          System.out.printf("aux: %f%n", aux);
9          System.out.printf("aux: %010.2f%n", aux);
10         System.out.printf("aux: %10.2f%n", aux);
11     }
12 }
```

Imprime na tela:

1.4285714285714286
aux: 1,428571
aux: +1,428571
aux: 1,428571
aux: 0000001,43
aux: 1,43

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM