

---

---

# Introdução ao Java

Prof. Ítalo Assis

---

---

**Ajude a melhorar este material =]**

Encontrou um erro? Tem uma sugestão?

Envie e-mail para [italo.assis@ufersa.edu.br](mailto:italo.assis@ufersa.edu.br)

# Agenda

- Conceitos iniciais de programação em Java
- Estruturas de Controle e Decisão
- Estruturas de Repetição ou Iteração
- Funções e recursão

# Conceitos iniciais de programação em Java

# Java

- A **Sun Microsystems**, em 1991, financiou um **projeto de pesquisa** corporativa que resultou em uma linguagem de programação orientada a objetos chamada **Java**
- Um objetivo-chave do Java é ser capaz de escrever programas a serem **executados em uma grande variedade de sistemas computacionais** e dispositivos controlados por computador
- A web explodiu em popularidade em 1993 e a Sun viu o potencial de utilizar o Java para adicionar **conteúdo dinâmico**, como interatividade e animações, **às páginas da web**
- Ele é agora utilizado para desenvolver **aplicativos corporativos** de grande porte, aprimorar a funcionalidade de **servidores da web**, e desenvolver **aplicativos Android**
- A Sun Microsystems foi adquirida pela **Oracle** em 2010
- Muitos programadores Java tiram proveito das ricas coleções de classes existentes e métodos nas bibliotecas de classe Java, também conhecidas como **Java APIs**

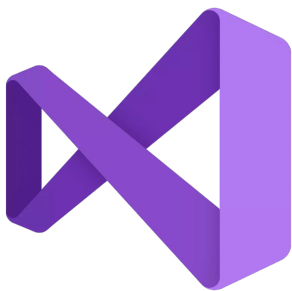
# Um ambiente de desenvolvimento Java típico



- Como instalar o Java?
  - [Java Downloads | Oracle](#)
  - [How to Install Java on Ubuntu 18.04 | Linuxize](#)
- O Java é distribuído em 3 diferentes edições: **Standard Edition (SE)**, Enterprise Edition (EE) e Micro Edition (ME).
- **OpenJDK** e **Oracle Java** são as duas principais implementações de Java, com quase nenhuma diferença entre elas, exceto que a Oracle Java tem alguns recursos comerciais adicionais
- Existem dois pacotes Java diferentes, o Java Runtime Environment (JRE) e o **Java Development Kit (JDK)**

# Um ambiente de desenvolvimento Java típico

- Normalmente, existem cinco passos para criar e executar um aplicativo Java: editar, compilar, carregar, verificar e executar.
- Fase 1: criando um programa
  - Você digita um programa Java (código-fonte) utilizando o editor e salva o programa
  - Arquivos de código-fonte Java recebem um nome que termina com a **extensão .java**



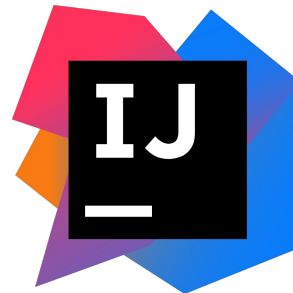
Visual Studio



Netbeans



Eclipse



IntelliJ

# Um ambiente de desenvolvimento Java típico

- Fase 2: compilando um programa Java em bytecodes
  - Utilize o compilador Java para compilar o programa. Por exemplo, para compilar um programa chamado *Programa.java*, você digitaria:
    - ***javac Programa.java***
  - Se o programa compilar, o compilador **produz um arquivo .class** (*Programa.class*)
    - IDEs tipicamente fornecem um item de menu, como *Build* ou *Make*, que chama o comando *javac* para você.
    - Se o compilador detectar erros, você precisa voltar para a Fase 1 e corrigí-los
    - **O compilador Java converte o código-fonte Java em bytecodes** que representam as tarefas a serem executadas



# Um ambiente de desenvolvimento Java típico

- **A Java Virtual Machine (JVM)** - uma parte do JDK e a base da plataforma Java - **executa bytecodes**
- A máquina virtual (virtual machine - VM) é um aplicativo de software que simula um computador, mas oculta o sistema operacional e o hardware subjacentes dos programas que interagem com ela
- Se a mesma máquina virtual é implementada em muitas plataformas de computador, os aplicativos escritos para ela podem ser utilizados em todas essas plataformas
- A JVM é uma das máquinas virtuais mais utilizadas

# Um ambiente de desenvolvimento Java típico

- Diferentemente das instruções em linguagem de máquina, que são dependentes de plataforma, instruções bytecode são independentes de plataforma.
- Portanto, **os bytecodes do Java são portáveis** - sem recompilar o código-fonte, as mesmas instruções em bytecodes podem ser executadas em qualquer plataforma contendo uma JVM que entende a versão do Java na qual os bytecodes foram compilados.
- A JVM é invocada pelo comando *java*. Por exemplo, para executar um aplicativo Java chamado *HelloWorld*, você digitaria:
  - ***java HelloWorld***

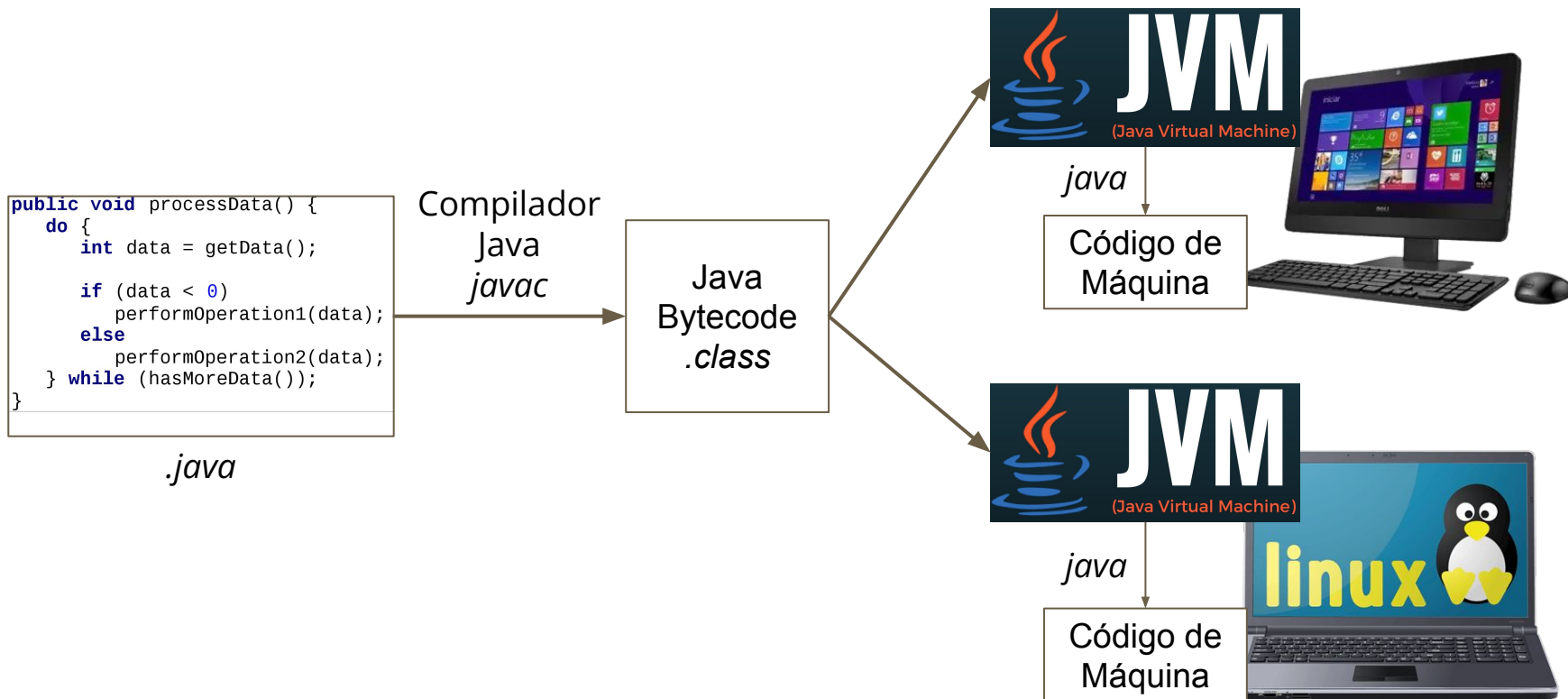
# Um ambiente de desenvolvimento Java típico

- Fase 3: carregando um programa na memória
  - O carregador de classe da JVM pega os arquivos *.class* que contêm os bytecodes do programa e os transfere para a memória primária
  - Ele também carrega qualquer um dos arquivos *.class* fornecidos pelo Java que seu programa usa
- Fase 4: verificação de bytecode
  - O verificador de bytecode examina seus bytecodes para assegurar que eles são válidos e não violam restrições de segurança do Java

# Um ambiente de desenvolvimento Java típico

- Fase 5: execução
  - A JVM executa os bytecodes do programa, realizando, assim, as ações especificadas por ele
  - A JVM traduz os bytecodes para a linguagem de máquina do computador

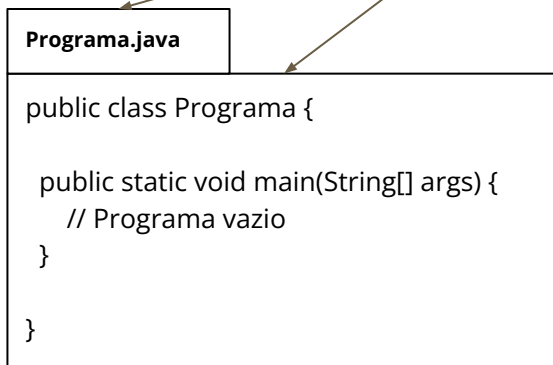
# Um ambiente de desenvolvimento Java típico



# Um ambiente de desenvolvimento Java típico

- Programa mínimo:

**Importante:** O nome da classe de ser o mesmo do arquivo



```
Programa.java

public class Programa {

    public static void main(String[] args) {
        // Programa vazio
    }

}
```

# Prática

- Compilar e executar um programa em Java

# Comentários

- Inserimos comentários para documentar programas e aprimorar sua legibilidade
- O compilador Java ignora os comentários
- Comentários de uma linha:

```
// Este eh um comentario de uma linha
```

- Comentários de várias linhas:

```
/*  
Este                eh                um                comentario  
De                  varias                linhas  
*/
```



# Variáveis e constantes

- Variável: espaço na memória designado para o armazenamento de um determinado valor
- Constantes: Espaço na memória para o armazenamento de um valor que não pode ser alterado durante o desenvolvimento do código
- Regras para nomes de variáveis e constantes:
  - Não pode iniciar com números
  - Não pode possuir caracteres especiais
  - Não pode possuir espaços em branco
- O padrão camelCase é uma convenção utilizada para variáveis e constantes em Java
- Nomes corretos: *distanciaPercorrida*, *notaFinal*, *situacaoCadastral*....
- Nomes incorretos: *1aresposta*, *soluç@o*, *x 1*...

# Variáveis e constantes

- Variáveis e constantes precisam ser declaradas
- Elas devem possuir um tipo associado
  - Os tipos do Java são divididos em primitivos e por referência
    - Entenderemos melhor os tipos por referência mais a frente no curso
    - Tipos primitivos: *boolean*, *byte*, *char*, *short*, *int*, *long*, *float* e *double*
  - As variáveis locais não são inicializadas por padrão

# Variáveis e constantes

**VariaveisConstantes.java**

```
public class VariaveisConstantes{
    public static void main(String[] args) {
        // variaveis
        int numero = 2, n2;
        float valor1, v1 = 3.68F;
        double valor2, v2 = 3.68;
        String palavra = "Orientacao a Objetos", palavra2;
        char letra = 'w', outraLetra;
        boolean resposta1 = true, resposta2 = false, resposta3;
        // constantes
        final double ACELERACAO_GRAVIDADE = 9.78;
        final String msg = "Bem vindo(a)!\n";
        // O algoritmo continua aqui
    }
}
```

# Entrada e saída de dados

- Comandos para exibir texto:
  - `System.out.print(...);`
  - `System.out.println(...);`
    - Salta uma linha no final
  - `System.out.printf(...);`
    - Exibe os dados formatados (similar ao comando em C)
- Exemplos:
  - `String nome = "UFERSA"; System.out.println(nome);`
  - `int espera = 15; System.out.printf("O tempo de espera é de %d minutos%n", espera);`

# Entrada e saída de dados

- Um **Scanner** permite a um programa ler os dados
- No exemplo, o **import** permite o uso dos métodos da API Java utilizados no programa
- O objeto de entrada padrão, **System.in**, permite que aplicativos leiam bytes de informações digitadas pelo usuário
- O Scanner traduz esses bytes em tipos (como ints) que podem ser utilizados em um programa

```
import java.util.Scanner;

public class NomeDaClasse {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String str = input.next();
        String str2 = input.nextLine();
        double numd = input.nextDouble();
        int numi = input.nextInt();
        float numf = input.nextFloat();
        System.out.printf("%s %g %d %f %n", str, numd, numi, numf);
        input.close();
    }
}
```

# Prática

- Escreva um programa que recebe o nome de uma pessoa e deseja a ela as boas vindas mencionando seu nome

# Expressões aritméticas

Operação(ões)	Operador(es)	Expressão algébrica	Expressão Java
Multiplicação	*	$bm$	<code>b * m</code>
Divisão	/	$x/y$ ou $x \div y$	<code>x / y</code>
Resto	%	$r \bmod s$	<code>r % s</code>
Adição	+	$f + 7$	<code>f + 7</code>
Subtração	-	$p - c$	<code>p - c</code>

- A divisão de inteiros produz um quociente inteiro. Por exemplo, a expressão `7 / 4` é avaliada como 1.

# Expressões aritméticas

Operador(es)	Operação(ões)	Precedência
*	Multiplicação	Avaliado primeiro. Se houver vários operadores desse tipo, eles são avaliados da esquerda para a direita.
/	Divisão	
%	Resto	
+	Adição	Avaliado em seguida. Se houver vários operadores desse tipo, eles são avaliados da esquerda para a direita.
-	Subtração	
=	Atribuição	Avaliado por último



# Expressões aritméticas

*Álgebra:*

$$z = pr \% q + w/x - y$$

*Java:*

`z = p * r % q + w / x - y;`

6

1

2

4

3

5

# Expressões aritméticas

- Os parênteses são utilizados para agrupar termos em expressões Java da mesma maneira como em expressões algébricas
  - Para multiplicar  $a$  por  $b + c$  escrevemos  $a * (b + c)$
- Se uma expressão contiver parênteses aninhados, como  $((a + b) * c)$ , a expressão no conjunto mais interno dentro dos parênteses é avaliada primeiro
- Parênteses tem precedência com relação as operações aritméticas

# Prática

Escreva um algoritmo que, tendo como dados de entrada dois pontos quaisquer no plano,  $(x1,y1)$  e  $(x2,y2)$ , escreva a distância entre eles.

# Estruturas de Controle e Decisão

# Operadores relacionais em Java

- Os operadores da tabela podem ser usados para comparar valores de tipos nativos numéricos
- Para comparar objetos do tipo *String*, podemos utilizar o método *equals*:  
*str1.equals(str2)*

==	igualdade
!=	diferença
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a

# Operadores lógicos em Java

- Valores do tipo *boolean* e resultados de operações ou métodos que retornem valores booleanos podem ser combinados entre si através dos operadores lógicos:

&&	operador de conjunção ( E )
	operador de disjunção ( OU )

- Exemplo: `((a >= b) && (a >= c))`
- Há também o operador de negação ! ( NÃO ) que inverte o valor de um booleano
  - `!(2 > 3)` retorna *true*

# Estruturas condicionais em Java

- *if ... else ...*
- Operador ternário
- *switch*

```
if (condição) {  
    // instruções a serem  
    // realizadas caso a  
    // condição seja VERDADEIRA  
} else {  
    // instruções a serem  
    // realizadas caso a  
    // condição seja FALSA  
}
```

```
int taxa = (salario > 3000 ? 20 : 15);
```

```
switch (número) {  
    case valor1:  
        // comandos caso número == valor1  
        break;  
    case valor2:  
        // comandos caso número == valor2  
        break;  
    [...]  
    default:  
        // comandos caso número seja  
        // diferente dos valores testados  
        break;  
}
```

# Prática

- Escreva um programa que leia uma data com dia, mês e ano como números inteiros e retorne a data no formato a seguir:
  - “[dia] de [mês] de [ano]”
  - “9 de agosto de 2021”
  - Utilize a estrutura *if...else*
- Informe também quantos dias tem o mês da data informada
  - Para simplificar, considere que fevereiro sempre tem 28 dias
  - Utilize a estrutura *switch*



# Estruturas de Repetição ou Iteração

# Contadores

- Variáveis que recebem um valor inicial e são modificadas a cada iteração de uma estrutura de repetição.
- Contadores podem ser variáveis de qualquer tipo numérico.
- Estas variáveis devem:
  - receber um valor inicial
  - ser alteradas a cada iteração do laço
  - ter seu valor verificado a cada iteração a fim de saber se um valor final foi alcançado.
- Por exemplo, para fazer uma estrutura que conte de um até dez, pode-se utilizar um contador que receberá o valor inicial de um, sendo acrescido de um em um até que o valor deste contador seja igual a dez, interrompendo então a repetição do laço.

# Contadores

- Valores dos contadores são alterados através da atribuição do resultado de uma operação à variável que representa o contador.
- Geralmente estas operações envolvem o próprio contador
  - *linha = linha + 1*
- O Java tem operadores especiais para a modificação de variáveis usando a própria variável

<b>++</b>	var++ ou ++var	var = var + 1
<b>+=</b>	var += val	var = var + val
<b>--</b>	var-- ou --var	var = var - 1
<b>-=</b>	var -= val	var = var - val
<b>*=</b>	var *= val	var = var * val
<b>/=</b>	var /= val	var = var / val

# Lembra do pré/pós incremento?

- Vamos observar os resultados das execuções dos códigos a seguir:

- ```
int a = 5;  
int b = 5 + a++;  
System.out.println("a = " + a + " b = " + b);
```

  - a = 6 b = 10

- ```
int c = 5;  
int d = 5 + ++c;  
System.out.println("c = " + c + " d = " + d);
```

  - c = 6 d = 11

# Estruturas de repetição

- Repete um comando ou bloco de comandos enquanto uma condição for verdadeira
- A condição deve ser um valor booleano ou expressão cujo resultado seja booleano
- As diferentes sintaxes utilizadas para representar laços são intercambiáveis

# Laço *while*

- O bloco ou comando associado ao laço será repetido enquanto o valor booleano avaliado pela instrução *while* a cada iteração for *true*.
- Se o argumento para a instrução *while* for inicialmente *false*, o comando ou bloco de comandos associado não será executado nem mesmo uma vez.

```
while (condição) instrucao_a_ser_repetida;
```

```
while (condição) {  
    // instruções a serem  
    // repetidas  
}
```

# Prática

- Escreva um programa em Java que imprima a velocidade em metros por segundo, milhas por horas e pés por segundo correspondentes às velocidades em quilômetros por hora, de zero a cinquenta, de meio em meio quilômetro por hora.
- A conversão das unidades de velocidade segue a lista abaixo.
  - 1 quilômetro por hora = 0.2778 metros por segundo
  - 1 quilômetro por hora = 0.6214 milhas por hora
  - 1 quilômetro por hora = 0.9113 pés por segundo

## Laço *do-while*

- Java oferece outro tipo de laço que é executado enquanto uma condição for verdadeira, mas garante que o bloco associado ao laço será executado ao menos uma vez.
- A condição é avaliada ao final do laço.

```
do {  
    // instruções a serem  
    // repetidas  
} while (condição);
```



# Prática

- Crie um programa para receber do usuário os dados de um cartão e verificar se os dados são válidos.
  - Os dados que o programa deve receber são nome, número, código e validade (mês e ano).
  - A validação consiste em verificar se a validade é maior que a data atual.
  - Caso não seja, deve-se solicitar novamente ao usuário a inserção da validade até que sejam informados dados válidos.
  - O programa deve exibir uma mensagem quando os dados forem validados.

# Laço *for*

- O Java tem uma estrutura especializada para a implementação de repetição controlada por contadores, que agrupa a inicialização, modificação e comparação da variável de controle em uma única instrução.

```
for (inicialização; verificação_de_condições; atualização) instrucao_a_ser_repetida;
```

```
for (inicialização; verificação_de_condições; atualização) {  
    // instruções a serem  
    // repetidas  
}
```

```
for (int i=0; i<10; i++)
```

# Prática

- Escreva um programa que receba um número  $n$  e calcule e exiba o fatorial de cada número de zero a  $n$ .

# Comandos *break* e *continue*

- A instrução *break*, quando executada em um *while*, *for*, *do-while* ou *switch*, ocasiona a saída imediata desta instrução.
  - A execução continua com a primeira instrução depois da instrução de controle.
- Já a instrução *continue*, quando executada em um *while*, *for* ou *do-while*, pula as instruções restantes no corpo do laço e prossegue com a próxima iteração.

# Prática

- Execute o código a seguir:

```
for (int i = 1; i <= 10; i++) {  
    if (i == 3) continue;  
    if (i == 8) break;  
    System.out.println(i);  
}
```

- Como os comandos *break* e *continue* afetam a contagem dos números?

## Laço *for each*

- O Java possui uma sintaxe especial que pode ser utilizada em alguns casos para acessar elementos de um vetor
- Veremos esse laço adiante no curso

# Funções e recursão

# Funções

- Sendo Java uma linguagem orientada a objetos, é mais comum utilizarmos a nomenclatura *método* em vez de *função*
  - Veremos mais adiante no curso qual é a diferença
- Para definir uma “função” que poderá ser chamada diretamente pela função *main*, devemos definí-la da seguinte maneira:

```
public static [tipoDoRetorno] [nomeDaFuncao]([lista de parâmetros]) {  
    ...  
    return [valor de retorno];  
}
```



# Prática

- Escreva um programa que leia 3 números *double* e informe sua média
  - O cálculo da média deve ser feito através de uma função

# Métodos recursivos

## 1. Caso base

Define quando a recursão deve parar.

**Se não houver, haverá uma recursão infinita!**

No exemplo, a recursão para quando  $n = 0$

## 2. Caso recursivo

Define quando o caso pode ser resolvido usando casos menores.

**As chamadas recursivas devem conduzir ao caso base, senão haverá recursão infinita!**

No exemplo, há chamada recursiva quando  $n > 0$

### Fatorial

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n - 1)! & \text{se } n > 0 \end{cases}$$

# Prática

- Implemente uma função recursiva para calcular o fatorial de um número  $n$
- Teste a função criada

## Fatorial

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n - 1)! & \text{se } n > 0 \end{cases}$$

# Referências

DEITEL, Paul J. **Java: como programar**. 8.ed. São Paulo: Pearson Prentice Hall, 2010. 1144 p. ISBN: 9788576055631.

SANTOS, R. **Introdução à programação orientada a objetos usando JAVA**. 2. ed. Rio de Janeiro: Campus, 2013. 336p.