Introdução

Nivelamento de Lógica de Programação e Programação Orientada a Objeto

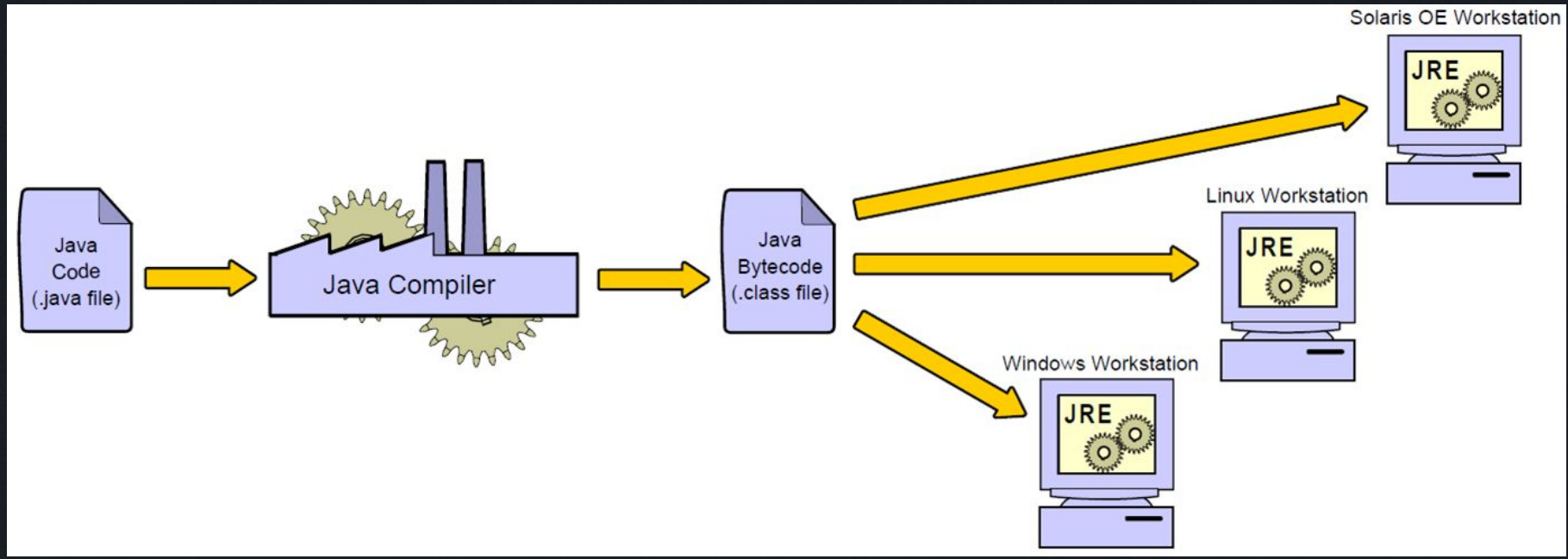
Introdução ao Java

Sucesso da Internet

Introdução ao Java

Sucesso da Internet

"Write once, run everywhere"



Introdução ao Java

O que é **JDK** e **JRE**?

- *JDK(Java Development Kit)* é um pacote que inclui tudo o que é necessário para escrever aplicações.

- *JRE*(*Java Runtime Environment*) é a camada de software, que é executado sobre um sistema operacional, e nos fornece os recursos necessários para ter um ambiente que seja possível executar programas em Java.

- *JVM(Java Virtual Machine)* é o coração do Java, responsável por executar os programas em formato de *bytecode* e por fornecer a capacidade multi-plataforma.

Introdução ao Java

Por que Java?

- Java é uma das linguagens de Orientação a Objetos mais usadas;
- Java é simples, versátil, robusta e segura;
- Portável (independente de sistema operacional);
- Gratuita e *open-source;*
- Popular, rodeada por uma comunidade muito ativa;
- Muitas ferramentas disponíveis;
- Muita documentação disponível.

Introdução ao Java

O que é Java?

- Uma tecnologia;
- Uma linguagem de programação;
- Uma plataforma de desenvolvimento;
- Um software distribuído pela *Oracle;*
- Um ambiente de execução de programas;
- Uma ilha da Indonésia (é o mar ao norte da ilha).

ada

Hands-on: Preparando o ambiente

Preparando ambiente

1. Baixe e instale o JDK;
2. Baixe e instale a IDE (recomendo IntelliJ Community)

O tradicional "Hello, World!

A forma mais tradicional da primeira linha de código, em Java, é:

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

        System.out.println("Hello, World!");
    }
}
```

O tradicional "Hello, World!

Podemos utilizar as seguintes ferramentas para o nosso primeiro *"Hello, World!"*:

- **Bloco de Notas**
- **J Shell** (já vem instalado no pacote da JDK)
- **IntelliJ**

Variáveis

Variáveis

Java é **fortemente tipada**, então é necessário definir o tipo de cada variável no momento de sua declaração. Além disso, toda instrução Java termina com ponto-e-vírgula.

```
tipo nomeVariavel = valor;
```

Como você vê em **nomeVariavel**, seguimos uma convenção "*camel case*" para nomes de variáveis e métodos e "*pascal case*" para nomes de classes.

Variáveis

Tipos

Os tipos primitivos são oito: quatro para inteiros, dois para ponto flutuante, um para booleanos e um para caracteres.

Palavra-chave	Tamanho	Descrição	Exemplo
byte	1 byte	Números inteiros de pequeno porte	100
short	2 bytes	Números inteiros de porte médio	3200
int	4 bytes	Números inteiros	42
long	8 bytes	Números inteiros grandes	10000000000L
float	4 bytes	Números com ponto decimal (precisão simples)	3.14f
double	8 bytes	Números com ponto decimal (precisão dupla)	3.14159265359
char	2 bytes	Um único caractere Unicode	'A'
boolean	1 bit	Valor lógico	true

A sintaxe de casting em Java é simples:

```
int numero = (int) 3.0;
```

Cuidado para respeitar os limites de cada tipo ou resultará em erro.

Variáveis

Valores precisos

Lembre-se que `double` e `float` são números de ponto flutuante e, portanto, não são precisos.

Quando for necessário garantir precisão, seja em cálculo científico, financeiro ou outro, opte por usar `java.math.BigDecimal`.

Java carrega apenas o pacote *java.lang* por padrão. Os demais precisam ser importados. Para usar *BigDecimal* importe antes da sua declaração de classe:

```
import java.math.BigDecimal;
```

Para a representação de textos usando `String`. Veja que todos os tipos primitivos iniciam com letra minúscula, mas `String` inicia com maiúscula. Isso porque `String` segue a convenção "pascal case", ou seja, é uma classe assim como `System`.

Entrada e saída do sistema

Entrada e saída do sistema

A entrada e saída de dados padrão do sistema permite a aplicação ser interativa com o usuário através do terminal.

No primeiro código, o *"Hello, World!"*, já fizemos uma saída usando `System.out.println`. Esse comando imprime uma linha inteira no terminal. Ou seja, ao final do conteúdo (parâmetro) adiciona uma quebra de linha.

```
public static void main(String[] args) {
    System.out.print(1);
    System.out.print(2);
    System.out.print(3);
}
```

A saída do código acima serão os números 1, 2 e 3 em linhas separadas. Ainda existem outros dois métodos para saída a partir de *System.out*.

Entrada e saída do sistema

Para escrever uma saída sem quebra de linha, use `print` e para formatar a saída, use `printf`.

```
public static void main(String[] args) {
    System.out.print(1);
    System.out.print(2);
    System.out.print(3);
}
```

Aqui a saída será "123", onde os números impressos estão em sequência um após o outro. Se quiser separá-los por vírgula pode formatar a saída

```
public static void main(String[] args) {
    System.out.printf("%d, %d, %d", 1, 2, 3);
}
```

Entrada e saída do sistema

Para fazer leitura de dados do sistema, precisamos ler de `System.in`. Ambos `System.in` e `System.out` representam os *"streams"* de entrada e saída padrão do sistema e são usados em manipulações de terminal. Para facilitar a manipulação de leitura, usamos a classe `java.util.Scanner` e é necessário importá-la.

```
import java.util.Scanner;

public class PrimeiroPrograma {
   public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Digite um número inteiro");
        int numero = input.nextInt();
        System.out.println("numero= "+ numero);
        input.close();
   }
}
```

Formatação de Texto

Formatação de Texto

O `printf` usa a mesma sintaxe de formatação disponível em *String.format*.

As marcações sempre acontecem precedidas de `%`, logo `%d` é um marcador para um parâmetro inteiro decimal, que pode ser um `int`. Algumas outras opções

- `b`: para valores booleanos (boolean)
- `s`: para textos (String)
- `f`: para números decimais (double)
- `%`: o caractere %, literalmente.
- `n`: quebra de linha, específica de sistema.

A quebra de linha com `%n` é especialmente útil porque nem todo sistema usa o mesmo símbolo para quebra de linha, alguns \n enquanto outros \rn.

Formatação de Texto

Também é possível definir outros detalhes de formatação, como a quantidade de casas decimais a serem escritas. Se quiser escrever PI como 3,14 use:

```
public static void main(String[] args) {
    System.out.printf("%.2f", Math.PI);
}
```

E se quiser 3.14, como é usado em inglês, basta usar as configurações de java.util.Locale:

```
import java.util.Locale;

public class PrimeiroPrograma {
    public static void main(String[] args) {
        String s = String.format(Locale.of("en", "us"), "%.2f", Math.PI);
        System.out.println(s);
    }
}
```

Operadores

Operadores

Assim como temos os operadores matemáticos da aritmética, álgebra, trigonometría, cálculo etc. O Java define símbolos especiais para realizar operações com nossas variáveis também. Esses operadores podem tomar um, dois ou três termos e agir sobre eles.

- **Aritméticos**: soma (+), subtração (-), multiplicação (*) e divisão (/);
- **Lógicos**: negação (!), E (&&), OU (||) // duplo pipe
- **Relacionais**: maior que (>), menor que (<), igual (==), diferente (≠)
- **Ternário**: ? :

Operadores

Existem algumas variações desses operadores que encurtam as expressões que escrevemos. Por exemplo, ao invés de:

```
int contador = 0;

contador = contador + 1;
```

Podemos utilizar o operador unário ++ na forma **pós-fixada**, nesse cenário, o valor atual do contador é informado, e em seguira, seu valor é adicionado 1.

```
contador++
```

Operadores

Um operador aritmético especial, e que não foi mencionado, é o *mod* %. Este operador informa o "resto da divisão", por exemplo:

```
int resultado = 3 % 2; // resultado = 1
```

O operador *mod* é, frequentemente, usado para verificar se um número é par ou ímpar.

Precedência

Operadores

A precedência refere-se à ordem em que operações são avaliadas em uma expressão. Em lógica de programação e em muitas linguagens de programação, existem regras claras sobre a ordem em que operadores são aplicados, garantindo que as operações sejam realizadas de maneira consistente.

Precedência

Operadores

1. **Parênteses:** Operações dentro de parênteses são sempre avaliadas primeiro. Se houver parênteses aninhados, os mais internos serão avaliados primeiro.

2. **Multiplicação, Divisão e Módulo:** Estas operações têm uma precedência mais alta do que adições e subtrações. Assim, elas são avaliadas antes das operações de adição e subtração.

3. **Adição e Subtração:** Se não houver parênteses, as operações de adição e subtração são avaliadas depois das operações de multiplicação, divisão e módulo.

4. **Operadores Relacionais:** São usados para comparar valores e, geralmente, têm uma precedência mais baixa do que operadores aritméticos, mas mais alta do que operadores lógicos.

5. **Operadores Lógicos:** Como **AND, OR, NOT**. Eles têm a precedência mais baixa na maioria das linguagens de programação, mas é sempre uma boa prática usar parênteses para tornar as intenções claras, especialmente em expressões complexas.

Precedência

Resumo

Regras Gerais de Precedência:

1. **Parênteses:** Têm a precedência mais alta; as operações dentro dos parênteses são sempre avaliadas primeiro.

2. **Operações Aritméticas (Multiplicação, Divisão, Módulo):** Têm precedência sobre adições e subtrações.

3. **Adições e Subtrações:** São avaliadas depois das operações aritméticas.

4. **Operadores Relacionais e Lógicos:** Têm uma precedência específica para comparações e operações booleanas.

Exercícios

Exercício 1: Controle de Acesso Múltiplo

Para que um usuário tenha acesso a um sistema, ele deve ser ativo, ter nível de permissão acima de 7 e não ter pendências financeiras.

Qual das alternativas representa corretamente essa condição?

a) *usuarioAtivo* && nivelPermissao > 7 && !pendenciasFinanceiras
b) *usuarioAtivo* || nivelPermissao > 7 && !pendenciasFinanceiras
c) *usuarioAtivo* && nivelPermissao > 7 || !pendenciasFinanceiras
d) *usuarioAtivo* && nivelPermissao >= 7 && !pendenciasFinanceiras

Exercício 2: Cálculo de Desconto Complexo

Um cliente recebe um desconto de 20% se comprar a partir 20 itens e seu valor total exceder a partir de 1000.

Qual das alternativas representa corretamente essa condição?

a) *numItens* > 20 && *valorTotal* > 1000
b) *numItens* >= 20 && *valorTotal* >= 1000
c) *numItens* > 20 || *valorTotal* > 1000
d) *numItens* >= 20 || *valorTotal* >= 1000

Exercício 3: Critérios de Promoção

Para um produto ser elegível para promoção, ele deve estar disponível em estoque e ter sido lançado nos últimos 6 meses.

Qual das alternativas representa corretamente essa condição?

a) `emEstoque && lancamentoRecente`
b) `!emEstoque || !lancamentoRecente`
c) `emEstoque || lancamentoRecente`
d) `emEstoque && !lancamentoRecente`

Exercício 4: Combinação de Requisitos

Um cliente recebe frete grátis se ele for um membro premium ou se sua compra exceder 500 e o endereço estiver dentro do país.

Qual das alternativas representa corretamente essa condição?

a) `isMembroPremium || valorCompra > 500 || enderecoNacional`
b) `isMembroPremium && valorCompra > 500 || enderecoNacional`
c) `isMembroPremium || valorCompra > 500 && enderecoNacional`
d) `isMembroPremium && valorCompra > 500 && enderecoNacional`

Exercício 5: Avaliação de Acesso Especial

Para um usuário ter acesso especial a recursos premium, ele deve ter uma assinatura ativa e pelo menos 50 pontos de reputação.

Qual das alternativas representa corretamente essa condição?

a) `assinaturaAtiva && pontoReputacao > 50`
b) `assinaturaAtiva || pontoReputacao >= 50`
c) `assinaturaAtiva || pontoReputacao > 50`
d) `assinaturaAtiva && pontoReputacao >= 50`

Exercício 6: Calculadora

Escreva um programa que aceite dois números inteiros no terminal e informe o resultado da soma, subtração, divisão e multiplicação.

Desafio

Ler a operação como terceiro parâmetro, por exemplo: **3 5 +**

Exercício 7: Determinar a Categoria de um Atleta com Base na Idade

Suponha que você queira determinar a categoria de um atleta com base em sua idade:

- Se a idade for inferior a 18, a categoria é "Juvenil".
- Se a idade estiver entre 18 e 40 (inclusive), a categoria é "Adulto".
- Se a idade for superior a 40, a categoria é "Master".

Usar apenas o operador ternário.

Exercício 8: Verificar se um Ano é Bissexto

Um ano é considerado bissexto se:

- É divisível por 4, mas não por 100, ou
- É divisível por 400.

Escreva um programa que determine se um ano é bissexto ou não, **usando apenas o operador ternário.**

Obrigada