REVISÃO DE PROGRAMAÇÃO ORIENTADA A OBJETOS

PROFESSOR: MOISES SOBRINHO

Guia Completo de Introdução a Java

Sumário

1. Introdução ao Java

Java é uma linguagem de programação **orientada a objetos**, **compilada** e **interpretada**, criada pela Sun Microsystems (atualmente Oracle) em 1995.

Características do Java

- **Portável**: "Write Once, Run Anywhere" (WORA) – roda em qualquer sistema com JVM.
- **Orientada a Objetos (POO)**: Usa classes, objetos, herança, encapsulamento e polimorfismo.
- **Robusta**: Possui coleta de lixo (Garbage Collector) e tratamento de exceções.
- **Segura**: Execução em ambiente controlado (JVM).

Aplicações do Java

- Aplicações Desktop (JavaFX, Swing)
- Aplicações Web (Spring, JSP)
- Aplicações Mobile (Android)
- Sistemas Embarcados e IoT

2. Configurando o Ambiente de Desenvolvimento

JDK (Java Development Kit)

É o kit de desenvolvimento necessário para compilar e executar programas Java.

- Baixe em: Oracle JDK ou OpenJDK

JRE (Java Runtime Environment)

Ambiente de execução para programas Java (já incluso no JDK).

Variáveis de Ambiente

Após instalar o JDK, configure a variável JAVA_HOME e adicione o bin ao PATH.

Exemplo (Windows):

```
JAVA_HOME = C:\Program Files\Java\jdk-17
PATH = %JAVA_HOME%\bin
```

3. Sintaxe Básica do Java

Estrutura de um Programa Java

```java
public class MeuPrimeiroPrograma {
    public static void main(String[] args) {
        System.out.println("Olá, Mundo!");
    }
}
```

- public class MeuPrimeiroPrograma: Declaração da classe.
- public static void main(String[] args): Método principal (ponto de entrada).
- System.out.println(): Imprime no console.

Tipos de Dados Primitivos

Tipo	Descrição	Exemplo
int	Números inteiros	int idade = 20;
double	Números decimais	double preco = 10.5;
char	Caractere único	char letra = 'A';
boolean	Verdadeiro/Falso	boolean ativo = true;
String	Texto (não primitivo)	String nome = "João";

Operadores

- **Aritméticos**: +, -, *, /, %
- **Comparação**: ==, !=, >, <, >= , <=
- **Lógicos**: && (E), || (OU), ! (NÃO)

4. Estruturas de Controle e Loops

Condicionais (if, else, switch)

```java
if (idade >= 18) {
    System.out.println("Maior de idade");
} else {
    System.out.println("Menor de idade");
}

switch (diaDaSemana) {
    case 1: System.out.println("Domingo"); break;
    case 2: System.out.println("Segunda"); break;
    default: System.out.println("Dia inválido");
}
```

Loops (for, while, do-while)

```java
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}

while (condicao) {
    // Código
}

do {
    // Código
} while (condicao);
```

5. Arrays e Coleções

Arrays

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

String[] nomes = new String[3];

nomes[0] = "Ana";
```

Listas (ArrayList)

java

```
import java.util.ArrayList;

ArrayList<String> frutas = new ArrayList<>();

frutas.add("Maçã");

frutas.add("Banana");

System.out.println(frutas.get(0)); // Maçã
```

6. Introdução à Programação Orientada a Objetos (POO)

Pilares da POO

1. **Encapsulamento**: Proteger dados usando private e métodos get/set.
2. **Herança**: Reutilizar código com extends.
3. **Polimorfismo**: Um objeto se comportar de várias formas.
4. **Abstração**: Esconder detalhes complexos (usando abstract e interfaces)

7. Classes e Objetos

Definindo uma Classe

```java
public class Carro {
    // Atributos
    private String modelo;
    private int ano;

    // Método Construtor
    public Carro(String modelo, int ano) {
        this.modelo = modelo;
        this.ano = ano;
    }

    // Método
    public void acelerar() {
        System.out.println("Acelerando...");
    }
}
```

Criando um Objeto

```java
Carro meuCarro = new Carro("Fusca", 1970);
meuCarro.acelerar();
```

8. Herança, Polimorfismo e Interfaces

Herança (extends)

```java
public class Animal {
    public void comer() {
        System.out.println("Animal comendo...");
    }
}

public class Cachorro extends Animal {
    @Override
    public void comer() {
        System.out.println("Cachorro comendo...");
    }
}
```

Interface (implements)

```java
public interface Veiculo {
    void acelerar();
}

public class Carro implements Veiculo {
    @Override
    public void acelerar() {
        System.out.println("Carro acelerando...");
    }
}
```

9. Tratamento de Exceções (try-catch)

```java
try {
    int resultado = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Erro: Divisão por zero!");
} finally {
    System.out.println("Sempre executa.");
}
```

10. IDE Recomendadas para Java

- **IntelliJ IDEA** (Poderosa e recomendada)
- **Eclipse** (Clássica e gratuita)
- **NetBeans** (Boa para iniciantes)
- **VS Code** (Leve, com extensões Java)

11. Próximos Passos

- Aprofundar em **JavaFX** (Interface Gráfica)
- Estudar **Spring Boot** (Desenvolvimento Web)
- Praticar algoritmos e estruturas de dados

Conclusão

Java é uma linguagem versátil e poderosa, essencial para desenvolvimento de software. Este guia cobre desde o básico até conceitos intermediários de POO, preparando você para avançar em projetos reais.

PALAVRAS RESERVADAS

if: Inicia uma estrutura condicional.

else: Complementa a estrutura if, executado quando a condição if é falsa.

switch: Seleciona um bloco de código a ser executado dentre várias alternativas.

case: Marca um bloco de código dentro de uma estrutura switch.

default: Define um bloco de código a ser executado se nenhum caso do switch for correspondente.

while: Inicia um loop que continua enquanto uma condição é verdadeira.

do: Inicia um loop que executa pelo menos uma vez e continua enquanto a condição for verdadeira.

for: Inicia um loop com inicialização, condição e incremento/decremento.

break: Encerra um loop ou estrutura switch.

continue: Pula para a próxima iteração de um loop.

return: Retorna um valor de um método e encerra sua execução.

try: Inicia um bloco de código que será testado para erros.

catch: Define um bloco de código que será executado se um erro ocorrer no bloco try.

finally: Define um bloco de código que é sempre executado após o bloco try e catch.

throw: Lança uma exceção**.**

throws: Indica que um método pode lançar exceções específicas.

static: Indica que um campo ou método pertence à classe, e não a instâncias individuais.

final: Define constantes e métodos que não podem ser sobrescritos, ou classes que não podem ser estendidas.

abstract: Define métodos que não têm implementação e devem ser implementados por subclasses, ou classes que não podem ser instanciadas.

synchronized: Indica que um método ou bloco de código é seguro para execução em threads

múltiplas.

volatile: Indica que uma variável pode ser modificada de forma assíncrona.

transient: Indica que uma variável não deve ser serializada.

public: Torna a classe, método ou variável acessível de qualquer outro código.

protected: Torna a classe, método ou variável acessível dentro do mesmo pacote e subclasses.

private: Torna a classe, método ou variável acessível apenas dentro da própria classe.