Estruturas Condicionais em Java - Das Primeiras Versões até Java 23

Neste artigo, exploraremos a evolução das estruturas condicionais em Java, desde a introdução das primeiras instruções if, else, switch, até as mais recentes inovações, como *Switch Expressions* e *Pattern Matching* nas versões mais recentes.

Analisaremos cada versão e como as melhorias em estruturas condicionais aumentaram a expressividade e a eficiência do código Java.

Finalizaremos com um conjunto de boas práticas para o uso adequado dessas condições.

Java 1 e 1.1: Fundamentos das Estruturas Condicionais

A primeira versão de Java trouxe as condições essenciais if, else if, else, e switch, que suportavam apenas tipos numéricos (int) e caracteres (char). Estas estruturas foram criadas para oferecer controle básico de fluxo, fornecendo os blocos de construção para decisões simples e ramificações.

```
public class EstruturasCondicionaisJava1 {
    public static void main(String[] args) {
        int idade = 18;

        if (idade >= 18) {
            System.out.println("Acesso permitido.");
        } else {
            System.out.println("Acesso negado.");
        }
    }
}
```

```
public class EstruturasCondicionaisJava1 {
    public static void main(String[] args) {
        int dia = 2;

        switch (dia) {
            case 1:
```

```
            System.out.println("Domingo");
            break;
        case 2:
            System.out.println("Segunda-feira");
            break;
        default:
            System.out.println("Outro dia");
    }
  }
}
```

Java 5: Integração de enum no switch

Com Java 5, switch foi aprimorado para suportar o tipo enum, tornando o código mais legível e confiável. Enums ajudam a definir constantes específicas e eliminar o uso de valores mágicos.

```
enum DiaSemana {
  DOMINGO, SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO
}

public class EstruturasCondicionaisJava5 {
  public static void main(String[] args) {
    DiaSemana dia = DiaSemana.QUINTA;

    switch (dia) {
      case SEGUNDA:
        System.out.println("Início da semana de trabalho.");
        break;
      case SEXTA:
        System.out.println("Último dia da semana de trabalho.");
        break;
      case SABADO, DOMINGO:
        System.out.println("Fim de semana.");
        break;
      default:
        System.out.println("Dia útil.");
    }
  }
}
```

Java 7: Adição de switch com Strings

Java 7 introduziu a possibilidade de usar switch com variáveis String, aumentando a flexibilidade ao lidar com textos. Essa melhoria contribuiu para simplificar o código em situações onde o String é o critério de decisão.

```
public class EstruturasCondicionaisJava7 {
  public static void main(String[] args) {
    String dia = "Segunda";
```

```
        switch (dia) {
            case "Domingo":
                System.out.println("Descanso merecido.");
                break;
            case "Segunda":
                System.out.println("Começo da semana.");
                break;
            default:
                System.out.println("Dia comum.");
        }
    }
}
```

Java 14: Introdução das Switch Expressions

A partir do Java 14, as *Switch Expressions* eliminaram a necessidade de break, permitindo que o switch retornasse valores diretamente. Esta mudança simplificou o código, tornando-o mais direto e reduzindo a probabilidade de erros acidentais.

```
public class EstruturasCondicionaisJava14 {
    public static void main(String[] args) {
        String dia = "Domingo";
        String mensagem = switch (dia) {
            case "Domingo" -> "Descanso no fim de semana.";
            case "Segunda", "Terça" -> "Início de uma nova semana.";
            default -> "Dia comum.";
        };
        System.out.println(mensagem);
    }
}
```

Java 17: Pattern Matching com instanceof

O *Pattern Matching* no instanceof simplificou a verificação de tipos, permitindo verificar e realizar o casting de forma mais direta e segura.

Exemplo de Pattern Matching com instanceof

```
public class EstruturasCondicionaisJava17 {
    public static void main(String[] args) {
        Object obj = "Texto de exemplo";

        if (obj instanceof String str) {
            System.out.println("Objeto é uma string: " + str);
        }
    }
}
```

Java 23: Match Expressions com Pattern Matching no switch

A introdução de *Pattern Matching* no switch no Java 23 trouxe ainda mais flexibilidade, permitindo combinar condições de tipos e valores em uma única expressão.

Exemplo de Match Expressions

```
public class EstruturasCondicionaisJava23 {
    public static void main(String[] args) {
        Object obj = -15;

        String resultado = switch (obj) {
            case Integer i && i > 0 -> "Número positivo";
            case Integer i && i < 0 -> "Número negativo";
            case String s -> "É uma string";
            default -> "Tipo não identificado";
        };
        System.out.println(resultado);
    }
}
```

Boas Práticas para Uso de Condicionais em Java

1. **Escolha a Estrutura Correta**:
   - Utilize if-else para condições simples e onde há poucas verificações.
   - Prefira switch para múltiplas condições em um único critério, como constantes de enum ou String.
2. **Evite Condicionais Aninhadas**:
   - Condicionais aninhadas podem prejudicar a legibilidade do código. Opte por simplificar as expressões e extrair trechos complexos para métodos auxiliares.
3. **Opte por Switch Expressions Quando Possível**:
   - A partir do Java 14, as *Switch Expressions* são mais legíveis e podem evitar erros comuns, como esquecer break.
4. **Use Pattern Matching Quando Apropriado**:
   - Pattern Matching, introduzido no Java 17, é útil para evitar verificações redundantes, especialmente quando se trabalha com tipos heterogêneos.
5. **Comentários Claros e Precisos**:
   - Comente o propósito de cada condição para facilitar a manutenção e entendimento do código, especialmente em condicionais mais complexas.

Resumo das Estruturas Condicionais em Java

1. **if-else Básico**

```
if (condicao) {
    // Código verdadeiro
} else {
    // Código falso
}
```

2. **switch com Inteiros, enum e String**

```
switch (variavel) {
    case CONSTANTE1:
        // Código
        break;
    default:
        // Código padrão
}
```

3. **Switch Expressions (Java 14)**

```
String resultado = switch (variavel) {
    case VALOR1 -> "Resultado para VALOR1";
    default -> "Valor padrão";
};
```

4. **Pattern Matching com instanceof (Java 17)**

```
if (obj instanceof String s) {
    System.out.println("String: " + s);
}
```

5. **Pattern Matching com Match Expressions (Java 23)**

```
String resultado = switch (obj) {
    case Integer i && i > 0 -> "Número positivo";
    case String s -> "É uma string";
    default -> "Outro tipo";
};
```

Conclusão

As estruturas condicionais em Java evoluíram significativamente, oferecendo maior flexibilidade e desempenho, essenciais para o desenvolvimento de software moderno. O uso correto dessas estruturas, aliado às boas práticas, resulta em códigos mais limpos, performáticos e fáceis de manter. Desenvolvedores que dominam estas ferramentas estão melhor preparados para construir soluções robustas e escaláveis.

EducaCiência FastCode para a comunidade