
# Melhor Expressividade

Nivelamento de Lógica de Programação e Programação Orientada a Objeto

# Melhor Expressividade

## Expressões Mais Ricas

Quando um projeto de software está mais próximo de um "código" que de uma "linguagem" (de programação) é hora de se preocupar. Isso quer dizer que um código, seja em qual linguagem ou tecnologia for, não deve ser algo incompreensível, mas antes deve prezar pela comunicação - entre as pessoas envolvidas na manutenção e evolução.

Dessa forma, as linguagens devem evoluir e devemos nos apropriar dos recursos que nos permitem comunicar com a flexibilidade e propriedade que precisamos. Vamos explorar dois novos recursos do Java que nos capacitam a tratar switch como expressão e o pattern matching para checagem e conversão mais simples.

# Switch expressions

# Switch expressions

*Esse tópico é melhor compreendido com o conhecimento de **expressões lambda.***

A sintaxe tradicional de um bloco `switch` é baseada na avaliação de uma variável e os blocos para cada caso, contendo o controle `break` em cada, e o caso `default`.

# Switch expressions

Vamos retomar o caso que trabalhamos no estudo de estruturas de condicional:

```
void main() {
    String tipoDiaSemana = switchDiaSemana();
    System.out.println(tipoDiaSemana);
}
static String switchDiaSemana() {
    DayOfWeek hoje = LocalDate.now().getDayOfWeek();
    switch (hoje) {
        case MONDAY, TUESDAY, WEDNESDAY, FRIDAY, THURSDAY:
            tipoDiaSemana = "Dia de Semana";
            break;
        case SATURDAY, SUNDAY:
            tipoDiaSemana = "Fim de Semana";
            break;
    }
    return tipoDiaSemana;
}
```

# Switch expressions

O método não é muito legível e a variável `tipoDiaSemana` parece um pouco perdida. Vamos mudar para a sintaxe usando `switch expression` e comparar a legibilidade,

```
void main() {
    String tipoDiaSemana = switchDiaSemana();
    System.out.println(tipoDiaSemana);
}
static String switchDiaSemana() {
    DayOfWeek hoje = LocalDate.now().getDayOfWeek();
    return switch (hoje) {
        case MONDAY, TUESDAY, WEDNESDAY, FRIDAY, THURSDAY -> "Dia de
Semana";
        case SATURDAY, SUNDAY -> "Fim de Semana";
    };
}
```

# Switch expressions

A sintaxe geral do switch expression é

```
int result = switch (expression) {
    case constant1 -> value1;
    case constant2 -> value2;
    default -> defaultValue;
};
```

# Switch expressions

Voltando ao nosso projeto de escola. Considere que queremos aplicar um adicional ao salário baseado na senioridade do InstrutorFixo.

```
class Financeiro {
   //código da classe sem mais alterações
   public BigDecimal aplicarAdicional(Senioridade senioridade) {
       return switch (senioridade) {
           case L1 -> BigDecimal.valueOf(1.1);
           case L2 -> BigDecimal.valueOf(1.12);
           case L3 -> BigDecimal.valueOf(1.175);
       };
   }
}
```

# Pattern matching

# Pattern matching

"Pattern matching" envolve testar o tipo específico de um objeto e, então, realizar o "casting" para o subtipo específico.

Relembrando o método realizarPagamento em Financeiro e aplicando o "pattern matching"

```
class Financeiro {
   public void realizarPagamento(Pagavel pagavel) {
       if (pagavel instanceof Instrutor instrutor) {
           System.out.println(instrutor.resumoContrato());
       }
       System.out.println(pagavel.gerarRecibo());
   }
}
```

# Pattern matching

Além disso, também é possível utilizar "*pattern matching*" com *switch* e em qualquer situação que segue o padrão "*testar com instance of e converter com casting*".

Considere o exemplo hipotético, apenas para fins didáticos

```
public static double getPerimeter(Shape shape) throws IllegalArgumentException {
   return switch (shape) {
       case Rectangle r -> 2 * r.length() + 2 * r.width();
       case Circle c    -> 2 * c.radius() * Math.PI;
       default          -> throw new IllegalArgumentException("Unrecognized shape");
   };
}
```

# Record Pattern

# Record Pattern

Outra feature, ainda mais recente, e com semelhanças ao "*pattern matching*" é o "record pattern".

Quando testar se um objeto é do subtipo de um *record* existente, você pode acessar os atributos do *record* como se fossem variáveis locais.

```
record Point(double x, double y) {}

static void printAngleFromXAxis(Object obj) {
    if (obj instanceof Point(double x, double y)) {
        System.out.println(Math.toDegrees(Math.atan2(y, x)));
    }
}
```

# Exercícios

**Exercício 1: Sistema de Gerenciamento de Biblioteca Parte Final**

**Finalização do projeto:** Uso de switch expressions para categorizar o estado dos empréstimos e aplicar "pattern matching" ao processar diferentes tipos de itens (Livro, Revista) no sistema.

Integração desses recursos ao sistema, mantendo a coesão e baixo acoplamento.

**Desafio:**

Implementar melhorias no sistema, usando switch expressions para lidar com diferentes estados de empréstimo e aplicar pattern matching para processar operações específicas de cada tipo de item.

Obrigada