Nova Sintaxe de Controle

Nivelamento de Lógica de Programação e Programação Orientada a Objeto

Melhor Controle

Neste módulo já estudamos que a orientação a objetos e o Java permitem o reaproveitamento de código através dos tipos e subtipos definidos por herança.

Também abordamos que a palavra-chave `final`, quando usada em classes, não permitem subtipos.

Em cenários que precisam controlar os possíveis subtipos, Java possui a funcionalidade de "classes seladas" (tradução livre de "sealed classes").

Melhor Controle

Outro novo recurso das versões mais recentes é o "*bloco de texto*" (tradução livre para "*text blocks*"), que permite definir uma String de várias linhas.

Essa funcionalidade é útil em situações de textos longos com muitas concatenações.

Esses novos recursos são fortes motivos para usar as versões mais recentes do Java nos seus projetos.

Classes Seladas

Classes seladas

As classes seladas permitem controlar quais subtipos são permitidos.

Também é possível definir sealed interfaces.

```
abstract sealed class Instrutor extends Pessoa implements Pagavel
permits InstrutorFixo, InstrutorTemporario {
   //código da classe sem mais alterações
}
final class InstrutorTemporario extends Instrutor {
   //código da classe sem mais alterações
}
final class InstrutorFixo extends Instrutor {
   //código da classe sem mais alterações
}
```

Classes seladas

As definições dos subtipos é marcada pela palavra-chave `permits` seguindo a lista dos subtipos.

Observe que essa parte deve vir após a herança (`extends`) e implementações de interface (`implements`).

Todo subtipo de classe selada deve ser um subtipo direto e seguir uma das opções:

- `final`: não pode ser herdada;
- `sealed`: deve definir os subtipos permitidos;
- `non-sealed`: aberta para qualquer subtipo.

Uma vez que uma `classe final` pode ser subtipo de uma `classe selada`, então é possível definir um `record` como subtipo permitido.

Classes seladas

Exemplo Prático

Imagine que você tem uma classe Forma e quer que apenas Circulo, Quadrado e Triangulo possam ser suas subclasses.

```
// Declara a classe selada "Forma"
public sealed class Forma permits Circulo, Quadrado, Triangulo {
    // ...
}

// "Circulo" pode herdar de "Forma"
public final class Circulo extends Forma {
    // ...
}

// "Quadrado" pode herdar de "Forma"
public final class Quadrado extends Forma {
    // ...
}

// "Triangulo" pode herdar de "Forma"
public non-sealed class Triangulo extends Forma {
    // ...
}

// Esta classe não pode herdar de "Forma", pois não está na lista
// public class Losango extends Forma { // Erro de compilação
//     // ...
// }
```

Classes seladas

Exemplo Prático

O non-sealed é, portanto, uma maneira de dizer: "Esta subclasse específica *(que faz parte da minha hierarquia controlada)* pode ser estendida por qualquer um a partir de agora." Ele quebra o "selo".

```
// Como "Triangulo" é "non-sealed", não há restrição.
// Esta classe pode herdar de "Triangulo" sem problemas,
// mesmo não estando na lista de permissões da classe
"Forma".
public class TrianguloEquilatero extends Triangulo {
    // Código específico de um triângulo equilátero...
}
```

Neste caso, a hierarquia é: Forma → Triangulo → TrianguloEquilatero.

O sealed em Forma garante que você não possa criar uma classe *TrianguloIsoceles* diretamente de Forma, mas o non-sealed em *Triangulo* permite que *TrianguloIsoceles*, *TrianguloRetangulo* e outras classes herdem dele sem restrições.

Classes seladas

Por que usar?

- Controle de herança: Você tem um controle granular sobre a hierarquia de classes, evitando heranças não intencionais ou inadequadas.

- Melhora a clareza do código: Fica explícito quais classes fazem parte de um grupo relacionado, facilitando a leitura e manutenção.

- Ajuda em padrões de design: É útil para implementar padrões como o de "visitor" ou quando você tem um conjunto fixo de subclasses para um tipo de dado.

- Uso com switch: Em Java, você pode usar um switch com *instanceof* para lidar com tipos de subclasses. Classes seladas garantem que o compilador verifique se todos os casos possíveis foram cobertos, o que ajuda a evitar bugs.

Bloco de
Texto

Blocos de Textos

Blocos de texto são uma nova funcionalidade que permite Strings de múltiplas linhas.

```
String boasVindas = """
                    Olá! Te damos as boas vindas ao Java.
                    Este módulo é para quem busca se atualizar na linguagem
                     ou já programa em outra tecnologia.
                    """;
String jsonPessoa = """
   {
      "nome": "Pessoa"
   }
   """;
```

Esse recurso é comumente usado para mensagens, representações de JSON e para instruções SQL - mas sem nenhuma restrição para qualquer outro cenário.

String templates

Esta funcionalidade está em "preview" e deve ser usada com cautela, uma vez que pode ser removida nas próximas versões.

São uma nova sintaxe (ainda em "preview") para Strings que permite a interpolação entre texto estático e variáveis.

```
final class InstrutorFixo extends Instrutor {
   //código da classe sem mais alterações
   @Override
   String resumoContrato() {
       return STR."Nome: \{this.getNome()}; Início:
\{this.dataInicio.format(DateTimeFormatter.ISO_DATE)}; Senioridade:
\{this.senioridade.getDescricao()}";
   }
}
```

String templates

Perceba como esse recurso nos permite implementar o método `resumoContrato` interpolando o texto estático com o dinamismo do código Java. E também é possível combinar o bloco de texto com o "String template".

```
final class InstrutorFixo extends Instrutor {
   //código da classe sem mais alterações
   String converterParaJson() {
       return STR."""
       {
           "nome": "\{this.getNome()}",
           "dataInicio": "\{this.dataInicio.format(DateTimeFormatter.ISO_DATE)}",
           "senioridade": "\{this.senioridade.getDescricao()}"
       }
       """;
   }
}
```

Exercícios

Exercício 1: Sistema de Gerenciamento de Biblioteca Parte 4

Evolução do projeto: Implementação de subtipos controlados usando `sealed classes` para diferentes perfis de usuário (`Bibliotecario`, `UsuarioComum`), restringindo as classes permitidas.

Uso de `text blocks` para criar mensagens aos usuários e relatórios de forma clara e legível.

Desafio:

Expandir o sistema de perfis de usuário para incluir diferentes tipos de relatórios gerados dinamicamente, aplicando tanto `sealed classes` quanto `text blocks`.

Obrigada