DANIEL ABELLA
www.daniel-abella.com

Daniel Abella

Profissional PMP, PMI-ACP e CSM, Gerente de Projetos no Virtus & UniFacisa e Professor do Curso de Sistemas de Informação da UniFacisa. Autor dos livros Gestão A3 (Ágil) e Scrum Arretado.

Conteúdo:

Capítulo 8

Miscelânia

Livro Java do Zero (Uma Viagem ao Mundo Java)

01 Estrutura Switch

- O switch possui a seguinte estrutura. Entre os parênteses do switch, informamos uma expressão a ser avaliada, podendo ser do tipo int, char, enum ou String (a partir do Java 7).
- Em cada um dos case, são os valores que você quer comparar com base na expressão, onde se a expressão for igual a um dos valores, o bloco de código abaixo a esse caso será executado até que encontre a instrução break, que encerra a execução, fazendo com que a execução siga nas linhas após o switch.

SWITCH COM JAVA (ANTERIORES AO 12)

SWITCH (EXPRESSAO) **{**

CASE VALOR1**:**

BLOCO DE CÓDIGO A SER EXECUTADO QUANDO A EXPRESSÃO É IGUAL A VALOR1

BREAK;

CASE VALOR2**:**

BLOCO DE CÓDIGO A SER EXECUTADO QUANDO A EXPRESSÃO É IGUAL A VALOR2

BREAK;

DEFAULT:

BLOCO DE CÓDIGO A SER EXECUTADO SE NENHUM DOS CASES FOR IGUAL À EXPRESSÃO

BREAK;

}

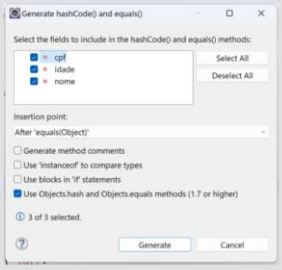
```java
public class SwitchExemplo1 {
    public static void main(String[] args) {

        int diaDaSemana = 3;
        String nomeDoDia = switch (diaDaSemana) {
            case 1 -> "Domingo";
            case 2 -> "Segunda";
            case 3 -> "Terça";
            case 4 -> "Quarta";
            case 5 -> "Quinta";
            case 6 -> "Sexta";
            case 7 -> "Sábado";
            default -> "Dia inválido";
        };

        System.out.println("Hoje é " + nomeDoDia);
    }
}
```

02 Equals

- A classe Object é a superclasse de todo mundo, diretamente ou indiretamente. Dentro desta classe, temos métodos importantes como equals, hashCode e toString.
- O operador == com variáveis primitivas avalia o conteúdo. O mesmo operador, se usado para comparar objetos, verifica se ambos estão na mesma posição da memória.
- Para comparar o conteúdo de objetos, devemos usar equal. Para que a comparação equals, avalie os atributos, precisamos sobrescrever (override) do método equals.

Gerando o Equals na IDE

Atualmente, existem diversas maneiras de gerar os métodos equals e hashCode. Usando a IDE Eclipse, basta fazer o atalho Alt - Shift - S (S de *Source Code*, Código Fonte), selecionar a opção Generate hashCode() and equals(). E, na tela apresentada, selecione as variáveis que serão usadas como critérios de comparação no equals. No nosso exemplo acima, selecionamos apenas a variável CPF.



Na IDE IntelliJ Idea, devemos seguir o atalho Alt - Insert e depois eleger a opção equals() and hashCode() e seguir os passos indicados. Note que, assim como no Eclipse, precisamos indicar que campos serão usados como critério de comparação. Novamente, apenas o CPF.

Ah, se você quiser gerar sem uso da IDE, você pode usar a biblioteca Lombok, discutida no capítulo referente a IDE Eclipse.

03 HashCode

- Assim como o equals, o hashCode é um método oriundo da classe Object, tendo como responsabilidade, gerar um código hash baseado nas propriedades relevantes usadas no método equals().
- A geração do HashCode foi apresentanda na seção acima.

04 Records

- Atualmente, quando criamos uma classe, geralmente criamos construtores, getters, setters, hashCode, equals e toString. Esta árdua atividade pode ser facilitada com uso de uma IDE que gere todo este boilerplate ou ainda usando algum framework como o Lombok.

TERMO *BOILERPLATE*

Usamos o termo *boilerplate* para que você agregue ao seu dicionário! *Boilerplate* significa seções de código que devem ser incluídas em muitos lugares com pouca ou nenhuma alteração, como *getters*, *setters*, *toString*, entre outros.

- A partir do Java 17, foi introduzido o conceito de Records, que permite a criação de classes de dados imutáveis, acabando a necessidade de geração de boilerplate. A sua estrutura é apresentada a seguir. Note que, após o modificador de acesso, usamos a palavra-chave record e, entre os parênteses, definimos, separados por vírgulas os campos imutáveis que irão compor a Record.

- Com base neste entendimento inicial, a seguir apresentamos um exemplo prático de Record. Note que, criamos a exata record apresentada acima, enquanto que no main da classe TesteRecord, criamos uma instância pessoa da maneira tradicional, com o operador new. Entretanto, para apresentar os campos, os chamamos pelo seu nome, a exemplo da linha indicada na seta 1. Devido ao fato de serem imutáveis, os valores atribuídos via construtor nas variáveis, não podem ser modificados, como poderíamos tentar na linha indicada pela seta 2, sob pena de culminar em um erro de compilação.

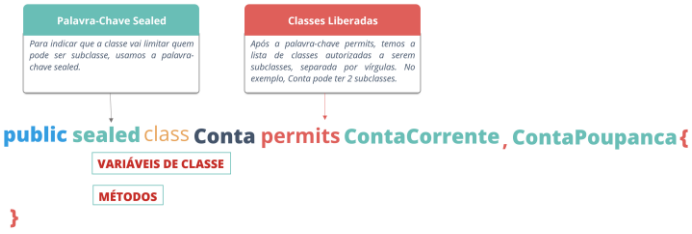
RECORD JAVA

```
public record Pessoa(String nome, int idade, String cpf) {
    // variáveis, métodos podem listar aqui
}
```

```
class UsandoRecords {
    public static void main(String[] args) {
        Pessoa pessoa1 = new Pessoa("Daniel Abella", 37,"123456");

    1   System.out.println(pessoa1.nome());  // Imprime: Daniel Abella
        System.out.println(pessoa1.idade()); // Imprime: 37
        System.out.println(pessoa1.idade()); // Imprime: 123456

        // Não é possível alterar os valores dos campos de um Record,
        // pois eles são imutáveis. Veja exemplo a seguir
    2   // pessoa.cpf("1234567");  // Este código dá um erro de compilação
    }
}
```

05 Sealed Classes

- Nos guias anteriores, apresentamos classes abstract como uma maneira de criar uma classe base para várias subclasses. Por outro lado, o usamos classes final quando queremos que, uma classe não possa ser estendida, isto é, não tenha subclasses. Com Sealed Class, temos um controle maior, podendo definir quais classes podem estender sua classe.
- Abaixo temos a estrutura de uma classe Sealed. Usamos antes da palavra-chave class, o termo sealed e, após o nome da classe, usamos a palavra-chave permits para indicar quais classes, separado por vírgulas, podem estender esta classe (Conta). Por fim, se uma classe não autorizada tentar estender a Classe Sealed, ocorrerá um erro de compilação.



- Oportunamente, esta funcionalidade também está disponível em interfaces, quase da mesma maneira, porém delimitando quem pode implementar a interface sealed. Da mesma maneira, se uma classe não autorizada tentar implementar uma interface Sealed, ocorrerá um erro de compilação.

- As subclasses (Cachorro e Gato) devem ter obrigatoriamente um dos 3 modificadores: final caso não queiram extensões, sealed caso queira ter o mesmo comportamento da superclasse (delimitar as suas subclasses) ou non-sealed (indicando que está aberto para extensão, isto é, subclasses).