SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Curso

TÉCNICO EM DESENVOLVIMENTO

DE SISTEMAS

Métodos equals, hashCode e uso de Lombok

Fernanda de Oliveira Nunes

Sorocaba
Novembro – 2024

SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Fernanda de Oliveira Nunes

Métodos equals, hashCode e uso de Lombok

Importância e o funcionamento dos métodos em Java
e como Lombok simplifica a sua implementação

Prof. – Emerson Magalhães

Sorocaba
Novembro – 2024

HISTÓRICO DE VERSÕES

Data	Versão	Descrição	Autor
09/11/2024	1.0	Pesquisa feita e documentada para envio do orientador Emerson Magalhães	Fernanda de O. Nunes

SUMÁRIO

OBJETIVO

Explorar a importância e o funcionamento dos métodos equals e hashCode em Java, analisando como eles influenciam o comportamento de coleções que utilizam hashing e como são usados para gerenciar entidades em frameworks como o Spring. Além disso, investigar como a biblioteca Lombok pode simplificar a implementação desses métodos e otimizar o desenvolvimento.

INTRODUÇÃO

Os métodos equals() e hashCode() são essenciais em Java para garantir comparações corretas de objetos e o funcionamento adequado em coleções baseadas em hash, como HashMap e HashSet. Implementá-los corretamente é importante para evitar problemas de busca e inserção e é essencial em frameworks como o Spring, que dependem deles para gerenciar entidades, caching e operações de persistência de forma consistente.

Lombok é uma biblioteca que ajuda a reduzir código repetitivo em Java. Com anotações ele gera automaticamente métodos, facilitando o desenvolvimento e minimizando erros manuais. Apesar de suas vantagens em produtividade, é importante considerar suas limitações antes de usá-lo em um projeto.

1. Fundamentos Teóricos

1.1. Contrato entre equals e hashCode

O hashCode é uma ferramenta usada para montar a tabela de hash de modo correto. Tabela Hash é uma onde as informações são armazenadas conforme um **"número hash"** calculado com base nas propriedades da informação. Isso permite que seja muito rápido recuperar uma informação na tabela. A ideia do equals é garantir que dois objetos são "significativamente iguais", ou seja, que os objetos tem um valor igual dadas as suas propriedades.

1.2. Regras que governam a implementação de equals e hashCode

O HashCode e equals andam juntos. Ao implementar um deve fazer o mesmo com o outro.

O contrato de equals() segue as seguintes diretrizes:

- Para qualquer valor de referência x, x.equals() deve retornar true;
- Para qualquer valor de referência x e y, x.equals(y) deve retornar true se, e somente se, y.equals(x) retornar true.
- Para qualquer valor de referência de x, y e z, se x.equals(y) retornar true e y.equals(z) também retornar true, então, x.equals(z) deve retornar true.
- Para qualquer valor de referência de x e y, múltiplas chamadas de x.equals(y) retornarão consistentemente true ou consistentemente false, contanto que nenhuma informação usada nas comparações do objeto de equals tenha sido alterada.
- Para qualquer valor de referência x que não seja null, x.equals(null) deve retornar false.

Já o contrato de hashCode segue as diretrizes:

- Um determinado objeto tem um determinado hashcode.
- Se dois objetos são iguais, seus hashcodes são os mesmos. O contrário não é verdade.
- Se os códigos hash forem diferentes, os objetos não são iguais com certeza.

- Objetos diferentes podem ter o mesmo código hash. No entanto, é um evento muito improvável. Nesse ponto, temos uma colisão, uma situação em que podemos perder dados.

1.3. Como o contrato entre equals e hashCode afeta o comportamento das coleções

No caso de um HashMap, a chave é armazenada com base no valor de hashCode() para que o mapa possa localizar rapidamente a chave em um bucket específico. Quando um objeto é procurado em um HashMap ou HashSet, a coleção usa o hashCode() para encontrar o bucket apropriado e depois usa equals() para localizar o objeto exato.

No caso de um HashSet, o valor de hashCode() determina o bucket em que o objeto será armazenado para garantir que não haja duplicatas. Se hashCode() e equals() não forem consistentes, a coleção pode não encontrar o objeto corretamente, mesmo que ele esteja presente.

1.4. Importância da implementação correta de equals e hashCode em entidades de aplicações Java

A implementação correta de `equals()` e `hashCode()` em entidades Java é essencial para garantir bom funcionamento em coleções baseadas em hash. Esses métodos asseguram que objetos iguais sejam tratados de forma consistente e armazenados corretamente, evitando problemas de busca, duplicatas e perda de desempenho devido a colisões de hash.

Se `equals()` indicar que dois objetos são iguais, `hashCode()` deve retornar o mesmo valor para ambos. Uma má implementação pode causar comportamentos inesperados, como falhas na busca e inserção de dados incorretos.

Implementar corretamente esses métodos também facilita a manutenção do código, evita erros e melhora a testabilidade da aplicação. Ferramentas de desenvolvimento e bibliotecas utilitárias podem ajudar a garantir uma implementação adequada.

2. Utilização Prática em Coleções Java e no Spring

2.1. Demonstração de exemplos práticos de equals e hashCode aplicados em coleções como HashSet e HashMap

```java
import java.util.HashSet;
import java.util.Set;

public class Main {
    public static void main(String[] args) {
        Pessoa p1 = new Pessoa("João", 30, "joao@example.com");
        Pessoa p2 = new Pessoa("João", 30, "joao@example.com");

        Set<Pessoa> pessoas = new HashSet<>();
        pessoas.add(p1);
        pessoas.add(p2);

        System.out.println("Número de pessoas no HashSet: " + pessoas.size());
    }
}

class Pessoa {
    private String nome;
    private int idade;
    private String email;

}
```

FIGURA 1

```java
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        Pessoa p1 = new Pessoa("Maria", 25, "maria@example.com");
        Pessoa p2 = new Pessoa("Maria", 25, "maria@example.com");

        Map<Pessoa, String> mapaPessoas = new HashMap<>();
        mapaPessoas.put(p1, "Professora");
        mapaPessoas.put(p2, "Engenheira");

        System.out.println("Número de entradas no HashMap: " + mapaPessoas.size());
    }
}

class Pessoa {
    private String nome;
    private int idade;
    private String email;


}
```

FIGURA 2

2.2. Exemplo prático de uma entidade Spring onde equals e hashCode são relevantes para operações de persistência e caching.

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.HashSet;
import java.util.Set;

@Service
public class ProdutoService {
    @Autowired
    private ProdutoRepository produtoRepository;

    public void exemploCache() {
        Set<Produto> cacheProdutos = new HashSet<>();
        Produto p1 = produtoRepository.findById(1L).orElseThrow();
        Produto p2 = produtoRepository.findById(1L).orElseThrow();

        cacheProdutos.add(p1);
        cacheProdutos.add(p2);

        System.out.println("Número de produtos no cache: " + cacheProdutos.size());
    }
}
```

FIGURA 3

3. Lombok: Simplificação do Código

3.1. Vantagens de usar Lombok em projetos Java

Usar o Lombok pode economizar muito tempo e esforço ao escrever. Reduz a quantidade de código que é preciso escrever e manter, e se concentrar na lógica e na funcionalidade de sua classe. Ajuda a evitar armadilhas e bugs comuns que podem surgir ao escrever métodos equals e hashCode manualmente, como esquecer de atualizá-los ao adicionar ou alterar campos, violar o contrato de igualdade de objeto ou causar vazamentos de memória ou problemas de desempenho. O Lombok também pode ajudar a impor consistência e práticas recomendadas para a igualdade de objetos em suas classes e projetos.

3.2. Análise das anotações @EqualsAndHashCode e @Data

O Projeto Lombok faz uso de annotations. As anotações são muito famosas entre os desenvolvedores Java e são amplamente usadas para sobrescrever métodos (@Override), para desabilitar avisos da IDE

(@SupressWarnings) ou quando se faz uso de APIs específicas como JPA, Spring, por exemplo.

A annotation @EqualsAndHashCode gera automaticamente os métodos equals() e hashCode() com base nos campos da classe. Isso significa que, se dois objetos têm os mesmos valores em seus campos, eles serão considerados iguais e terão o mesmo hashcode.

FIGURA 4

A annotation @Data é usada para gerar automaticamente os métodos getters, setters, equals(), hashCode() e toString() de uma classe Java. Ela substitui a necessidade de escrever manualmente esses métodos, o que reduz muito a quantidade de código para ser escrita.

FIGURA 5

3.3. Implementação de uma entidade com Lombok, comparando com uma implementação manual

-- Implementação com Lombok

```java
import lombok.AccessLevel;
import lombok.Setter;
import lombok.Data;
import lombok.ToString;

@Data
public class POJOExemplo {
private final String nome;
@Setter(AccessLevel.PACKAGE)
private int idade;
private double peso;
private String[] tags;
```

FIGURA 6

-- Implementação manual

```java
import java.util.Arrays;
public class POJOExemplo {
private final String nome;
private int idade;
private double peso;
private String[] tags;

public POJOExemplo(String nome) {
this.nome = nome;
}

public String getNome() {
return nome;
}

void setIdade(int idade) {
this.idade = idade;
}

public int getIdade() {
return idade;
}

public void setPeso(double peso) {
this.peso = peso;
}

public double getPeso() {
return peso;
}

public String[] getTags() {
return tags;
}

public void setTags(String[] tags) {
this.tags = tags;
```

FIGURA 7

3.4. Vantagens e Desvantagens de usar Lombok para equals e hashCode

- Vantagens

O Lombok elimina a necessidade de escrever manualmente métodos repetitivos, tornando o código mais conciso e fácil de interpretar. A implementação manual de equals() e hashCode() pode ser propensa a erros, como esquecer campos importantes. Lombok gera esses métodos de forma consistente e sem erros. Se novos campos forem adicionados à classe, Lombok pode incluir automaticamente esses campos nos métodos equals() e hashCode() sem precisar alterar manualmente o código.

- **Desvantagens**

Lombok precisa ser incluído como dependência no projeto, o que pode não ser desejável em ambientes que priorizam a minimização de dependências. Quem não conhece o Lombok pode ter dificuldade em entender como os métodos equals() e hashCode() estão sendo gerados, o que pode prejudicar a compreensão do código. A personalização é mais limitada em comparação com uma implementação manual. Em alguns casos, pode não ser totalmente compatível com certas ferramentas de análise de código, IDEs ou processos de build, exigindo configurações adicionais para funcionar corretamente. Como o código é gerado automaticamente, pode ser mais difícil depurar problemas.

CONCLUSÃO

O uso correto dos métodos equals() e hashCode() é essencial para garantir o bom desempenho das coleções em Java, especialmente em frameworks como o Spring. O Lombok facilita a implementação desses métodos, reduzindo código e aumentando a produtividade, mas traz desafios como dependência de uma biblioteca externa e limitações na personalização. Compreender e aplicar esses conceitos é fundamental para o desenvolvimento de aplicações Java eficientes e fáceis de manter, ajudando a criar sistemas mais confiáveis e sustentáveis a longo prazo.

BIBLIOGRAFIA

Linkedin. https://pt.linkedin.com/advice/1/what-benefits-drawbacks-using-lombok-generate?lang=pt

Artefatox. https://artefatox.com/annotations-do-lombok/#:~:text=%40EqualsAndHashCode,-A%20annotation%20%40EqualsAndHashCode&text=Isso%20significa%20que%2C%20se%20dois,e%20ter%C3%A3o%20o%20mesmo%20hashcode.

Dev Media. https://www.devmedia.com.br/uma-visao-sobre-o-projeto-lombok/28321

Code Gym. https://codegym.cc/pt/groups/posts/pt.210.java-hashcode-

Alga Works. https://blog.algaworks.com/entendendo-o-equals-e-hashcode/

Angeliski. https://angeliski.com.br/equals-e-hashcode/

LISTA DE FIGURAS