



SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Curso

**TÉCNICO EM DESENVOLVIMENTO
DE SISTEMAS**

**Métodos equals e hashCode em Java e o
uso de Lombok para otimizar código em
ambientes de desenvolvimento**

Ana Caroline Mena Bezerra de Paula

Sorocaba

Novembro – 2024



SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Ana Caroline Mena Bezerra de Paula

**Métodos equals e hashCode em Java e o
uso de Lombok para otimizar código em
ambientes de desenvolvimento**

Importância e funcionamento dos
métodos @equals e @hashCode

Prof. – Emerson Magalhães

Sorocaba
Novembro – 2024

SUMÁRIO

RESUMO	2
OBJETIVO	5
INTRODUÇÃO.....	6
1.1. Contextualização dos métodos @equals e @hashCode	6
1.2. Importância de @equals e @hashCode para coleções e frameworks como Spring 6	
1.3. Introdução ao Lombok e sua finalidade no desenvolvimento em Java.	6
2. FUNDAMENTOS TEÓRICOS	7
2.1. Explicação do contrato entre equals e hashCode.....	7

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.util.Objects;


@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ToString
@Entity
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private Double preco;

    // Método equals para comparar produtos com base em id e nome
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Produto produto = (Produto) obj;
        return Objects.equals(id, produto.id) && Objects.equals(nome, produto.nome);
    }

    // Método hashCode para garantir que objetos iguais tenham o mesmo código hash
    @Override
    public int hashCode() {
        return Objects.hash(id, nome);
    }
}
```

3. LOMBOK: SIMPLIFICAÇÃO DO CÓDIGO INTRODUÇÃO À BIBLIOTECA LOMBOK.....	9
3.1. Vantagens de usar Lombok em projetos Java.	9
- Redução de código boilerplate: Lombok automatiza a geração de métodos comuns como getters, setters, equals, hashCode e toString, tornando o código mais limpo e focado nas regras de negócio.....	9
- Maior produtividade: Menos código manual para escrever significa mais tempo para focar na lógica do negócio, aumentando a produtividade.	9
- Código mais legível: Com Lombok, o código fonte se torna mais conciso e fácil de ler, já que métodos triviais são omitidos da implementação.	9
3.2. Análise das anotações @EqualsAndHashCode e @Data.....	10
CONCLUSÃO.....	12
BIBLIOGRAFIA	13



Métodos equals e hashCode em Java e o uso de Lombok para otimizar código em ambientes de desenvolvimento

RESUMO

Em Java, os métodos "equals()" e "hashCode()" são muito importantes para garantir a comparação correta, principalmente em tipos de dados como "HashSet" e "HashMap". Eles ajudam a tornar as pesquisas e operações de pesquisa mais eficientes, ao mesmo tempo que mantêm a integridade dos dados. **Lombok** fornece simplicidade de código e gera automaticamente ``getters'`, ``setter'`, ``equals()'` e ``hashCode()'`, permitindo que os programadores se concentrem na lógica de negócios e melhorem o desempenho, reduzindo a duplicação de código.

OBJETIVO

O objetivo deste texto é explicar claramente como os métodos ``equal()`` e ``hashCode()`` afetam diretamente o comportamento das coleções em Java, especialmente quando se trata de persistência e cache, e como sua implementação adequada pode melhorar o desempenho e consistência do código. . Além disso, mostraremos como o **Lombok** pode melhorar o processo de desenvolvimento, reduzir a quantidade de duplicação de código e erros comuns. Usando essas práticas e ferramentas recomendadas, os desenvolvedores podem escrever códigos mais limpos, eficientes e de fácil manutenção, sem sacrificar funcionalidades importantes.

INTRODUÇÃO

1.1. Contextualização dos métodos `@equals` e `@hashCode`

São fundamentais para a comparação de objetos em Java.

O método `@equals` determina se dois objetos são iguais, enquanto

`@hashCode` retorna o valor de um hash que é usado em coleções baseadas em hash, como `@HashMap` e `HashSet`.

1.2. Importância de `@equals` e `@hashCode` para coleções e frameworks como Spring

Implementações corretas desses métodos garantem o comportamento esperado em coleções que utilizam hashing, evitando problemas como duplicação de objetos e falhas na recuperação de dados.

No Spring, esses métodos são cruciais para a gestão de entidades, especialmente em operações de persistência e caching.

1.3. Introdução ao Lombok e sua finalidade no desenvolvimento em Java.

Lombok é uma biblioteca que automatiza a geração de código boilerplate, como getters, setters, métodos `equals` e `hashCode`, facilitando o desenvolvimento e manutenção do código.

2. FUNDAMENTOS TEÓRICOS

2.1. Explicação do contrato entre equals e hashCode

Regras que governam a implementação de equals e hashCode

Se dois objetos são iguais de acordo com o método equals, eles devem ter o mesmo valor de hash retornando por hashCode.

Se o método hashCode é chamado várias vezes no mesmo objeto durante a execução de um programa, ele deve retornar consistentemente o mesmo valor, desde que nenhuma informação usada no equals tenha sido modificada.

Como o contrato entre equals e hashCode afeta o comportamento das coleções (ex.: HashMap, HashSet).

Utilizam o valor de hash para armazenar e recuperar objetos de forma eficiente.

Uma implementação incorreta pode levar a comportamentos inesperados, como incapacidade de encontrar objetos que deveriam estar presentes na coleção.

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

São essenciais para garantir a integridade e a eficiência das operações de comparação e armazenamento de objetos em coleções e frameworks.

Utilização Prática em Coleções Java e no Spring

```
import
org.springframework.beans.factory.annot
ation.Autowired;
import
org.springframework.stereotype.Service;
import java.util.List;

@Service
public class CategoriaService {

    @Autowired
    private CategoriaRepository
categoriaRepository;

    @Autowired
    private ProdutoRepository
produtoRepository;

    public Categoria
criarCategoria(Categoria categoria) {
        return
categoriaRepository.save(categoria);
    }

    public List<Categoria>
listarCategorias() {
        return
categoriaRepository.findAll();
    }
}
```

```
import
org.springframework.beans.factory.annot
ation.Autowired;
import
org.springframework.web.bind.annotation
.*;
import java.util.List;

@RestController
@RequestMapping("/categorias")
public class CategoriaController {

    @Autowired
    private CategoriaService
categoriaService;

    @PostMapping
    public Categoria
criarCategoria(@RequestBody Categoria
categoria) {
        return
categoriaService.criarCategoria(categor
ia);
    }

    @GetMapping
    public List<Categoria>
listarCategorias() {
        return
categoriaService.listarCategorias();
    }
}
```

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

```
import java.util.Objects;

public class Produto {
    private Long id;
    private String nome;

    public Produto(Long id, String
nome) {
        this.id = id;
        this.nome = nome;
    }

    // Método equals para comparar
objetos Produto
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        // Se for o mesmo objeto
        if (obj == null || getClass()
!= obj.getClass()) return false; // Se
forem de tipos diferentes
        Produto produto = (Produto)
obj;
        return Objects.equals(id,
produto.id) && Objects.equals(nome,
produto.nome); // Comparar id e nome
    }

    // Método hashCode para gerar o
hash do objeto Produto
    @Override
    public int hashCode() {
        return Objects.hash(id, nome);
    }
    // Gerar hash baseado no id e nome
}

// Getters e Setters (não serão
mostrados para manter o exemplo
simples)
    public Long getId() {
        return id;
    }

    public String getNome() {
        return nome;
    }
}
```


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

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.util.Objects;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ToString
@Entity
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private Double preco;

    // Método equals para comparar
    // produtos com base em id e nome
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || !getClass().isAssignableFrom(obj.getClass())) return false;
        Produto produto = (Produto) obj;
        return Objects.equals(id, produto.id) && Objects.equals(nome, produto.nome);
    }

    // Método hashCode para garantir
    // que objetos iguais tenham o mesmo
    // código hash
    @Override
    public int hashCode() {
        return Objects.hash(id, nome);
    }
}
```

3. LOMBOK: SIMPLIFICAÇÃO DO CÓDIGO INTRODUÇÃO À BIBLIOTECA LOMBOK

3.1. Vantagens de usar Lombok em projetos Java.

- **Redução de código boilerplate:** Lombok automatiza a geração de métodos comuns como getters, setters, equals, hashCode e toString, tornando o código mais limpo e focado nas regras de negócio.
- **Maior produtividade:** Menos código manual para escrever significa mais tempo para focar na lógica do negócio, aumentando a produtividade.
- **Código mais legível:** Com Lombok, o código fonte se torna mais conciso e fácil de ler, já que métodos triviais são omitidos da implementação.

3.2. Análise das anotações @EqualsAndHashCode e @Data

Como o Lombok pode gerar automaticamente os métodos equals e hashCode.

- @EqualsAndHashCode gera automaticamente os métodos equals e hashCode com base nos campos da classe.
- @Data combina várias anotações do Lombok, incluindo @Getter, @Setter, @EqualsAndHashCode, @ToString e @RequiredArgsConstructor, simplificando ainda mais a implementação de classes de dados

Exemplo prático de implementação de uma entidade com Lombok, comparando com uma implementação manual.

```
public class Produto {
    private Long id;
    private String nome;
    private Double preco;

    // Construtor
    public Produto(Long id, String nome, Double preco) {
        this.id = id;
        this.nome = nome;
        this.preco = preco;
    }

    // Getters e Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Double getPreco() {
        return preco;
    }

    public void setPreco(Double preco) {
        this.preco = preco;
    }

    // equals(), hashCode() e toString()
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Produto produto = (Produto) o;
        return id != null && id.equals(produto.id);
    }

    @Override
    public int hashCode() {
        return 31 * (id != null ? id.hashCode() : 0);
    }

    @Override
    public String toString() {
        return "Produto{id=" + id + ", nome=" + nome + ", preco=" + preco + '}';
    }
}
```

Vantagens e Desvantagens de Usar Lombok para equals e hashCode

Vantagens:

- **Redução de código boilerplate:** Lombok elimina a necessidade de escrever manualmente métodos repetitivos.
- **Melhor legibilidade e manutenção do código:** O código se torna mais limpo e fácil de manter, focando nas regras de negócio.

Desvantagens:

- **Dependência de uma biblioteca externa:** Adicionar Lombok como dependência significa depender de uma biblioteca de terceiros.
- **Potenciais problemas com depuração de código (debugging):** O código gerado pelo Lombok não aparece diretamente no código-fonte, o que pode dificultar a depuração

CONCLUSÃO

Ao trabalhar com coleções baseadas em @hash do Java, como @HashMap e @HashSet, é essencial substituir o '@equals' e. No entanto, executar este procedimento manualmente pode ser trabalhoso e sujeito a imprecisões, particularmente com conjuntos extensos que engloba numerosos atributos. Nesses cenários, a biblioteca Lombok realmente brilha ao criar e adicionar automaticamente métodos como 'toString', 'getter' e 'setter', o que reduz o código usual e facilita a leitura e o controle de números.

O Lombok ajuda os desenvolvedores a trabalhar mais rápido e a se concentrar em projetos importantes, mas também pode dificultar a localização de algumas ferramentas e levar tempo para que os novatos aprendam. Eles também nos motivam a melhorar e usar menos cópias, o que torna as coisas de maior qualidade. No entanto, é essencial que os fabricantes tenham uma compreensão profunda destes factores para evitar erros e garantir um sucesso duradouro.

BIBLIOGRAFIA

TREINAWEB

<https://www.treinaweb.com.br/blog/projeto-lombok-acelerando-o-desenvolvimento-java>

DIO

<https://www.dio.me/articles/como-lombok-pode-transformar-seu-codigo-java>

ALGAWORKS

<https://blog.algaworks.com/entendendo-o-equals-e-hashcode/>

DEVMEDIA

<https://codegym.cc/pt/groups/posts/pt.264.metodos-equals-e-hashcode-praticas-recomendadas>

MEDIUM

<https://medium.com/collabcode/projeto-lombok-escrevendo-menos-c%C3%B3digo-em-java-8fc87b379209>

