



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**

Manuela Leme Morais Almeida

**Sorocaba
Novembro – 2024**



SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL

SENAI “GASPAR RICARDO JUNIOR”

Manuela Leme Moraes Almeida

Métodos equals e hashCode em Java e o uso de Lombok

Pesquisa nsobre os métodos equals e hashCode em Java e
o uso de Lombok

Prof. – Emerson Magalhães

Sorocaba
Novembro – 2024

SUMÁRIO

INTRODUÇÃO	2
1. FUNDAMENTOS TEÓRICOS	5
1.1. CONTRATO ENTRE EQUALS E HASHCODE.....	5
2. UTILIZAÇÃO PRÁTICA EM COLEÇÕES JAVA E NO SPRING	6
2.1. COLEÇÕES BASEADAS EM HASHING (HASHMAP, HASHSET).....	6
2.2. UTILIZAÇÃO DE EQUALS() E HASHCODE() NO SPRING	6
2.2.1. ENTIDADES JPA (HIBERNATE)	6
2.2.2. ANOTAÇÕES @CACHEABLE COK SPRING CACHE.....	7
2.2.3. USO COM SET EM REPOSITÓRIO SPRING DATA	9
3. LOMBOK: SIMPLIFICAÇÃO DO CÓDIGO	10
3.1. INTRODUÇÃO À BIBLIOTECA LOMBOK.....	10
3.2. VANTAGENS.....	11
3.3. DESVANTAGENS	12
3.4. BOAS PRÁTICAS DO USO DE LOMBOK.....	13
CONCLUSÃO.....	14
BIBLIOGRAFIA	15
LISTA DE FIGURAS	16



Métodos equals e hashCode em Java e o uso de Lombok

INTRODUÇÃO

Os métodos `equals()` e `hashCode()` são fundamentais em Java, pois determinam como objetos são comparados e manipulados em coleções. Eles são herdados da classe `Object` e frequentemente sobrescritos para personalizar o comportamento de comparação.

Esses métodos são especialmente importantes em coleções baseadas em hashing, como `HashMap`, `HashSet` e `LinkedHashMap`. Em frameworks como o Spring, `equals()` e `hashCode()` são usados em operações de caching e na persistência de entidades, garantindo a consistência dos dados.

A biblioteca **Lombok** simplifica a implementação de `equals()` e `hashCode()` através de anotações como `@EqualsAndHashCode` e `@Data`, economizando tempo e melhorando a legibilidade do código.

1. FUNDAMENTOS TEÓRICOS

1.1. CONTRATO ENTRE EQUALS E HASHCODE

O contrato que governa equals() e hashCode() em Java é definido pelas seguintes regras:

- **Reflexividade:** Um objeto deve ser igual a si mesmo (ex.: `x.equals(x)` é sempre `true`).
- **Simetria:** Se `x.equals(y)` é `true`, então `y.equals(x)` também deve ser `true`.
- **Transitividade:** Se `x.equals(y)` e `y.equals(z)` são `true`, então `x.equals(z)` também deve ser `true`.
- **Consistência:** Múltiplas chamadas de `x.equals(y)` devem retornar consistentemente `true` ou `false`, desde que não haja modificações no estado dos objetos.
- **Não nulo:** `x.equals(null)` deve sempre retornar `false`.

Para `hashCode()`, as regras incluem:

- **Se `x.equals(y)` é `true`, então `x.hashCode()` deve ser igual a `y.hashCode()`.**
- **Se `x.equals(y)` é `false`, não há obrigatoriedade de `x.hashCode()` e `y.hashCode()` serem diferentes**, mas isso melhora a performance das coleções baseadas em hashing.

Essas regras garantem que coleções como `HashSet` e `HashMap` funcionem corretamente. Quando `hashCode()` não é consistente com `equals()`, isso pode causar falhas na busca e inserção de elementos nas coleções.

2. UTILIZAÇÃO PRÁTICA EM COLEÇÕES JAVA E NO SPRING

2.1. COLEÇÕES BASEADAS EM HASHING (HASHMAP, HASHSET)

O HashSet utiliza o hashCode() para determinar o bucket onde o objeto será armazenado e, em seguida, usa equals() para verificar se o objeto já existe no bucket.

Por exemplo, ao adicionar um objeto em um HashSet, ele primeiro verifica o hashCode() para localizar o bucket e, se houver colisão (mesmo hashCode()), utiliza equals() para comparar os elementos.

2.2. UTILIZAÇÃO DE EQUALS() E HASHCODE() NO SPRING

2.2.1. ENTIDADES JPA (HIBERNATE)

No Spring Data JPA, a implementação correta dos métodos equals() e hashCode() é fundamental para garantir o funcionamento correto de entidades persistentes. Por padrão, o **Hibernate** utiliza um proxy para gerenciar entidades, e a implementação adequada desses métodos evita problemas relacionados a identificadores (id) e estados transitórios.

```
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.EqualsAndHashCode;

@Entity
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Product {

    @Id
    @EqualsAndHashCode.Include
    private Long id;

    private String name;
    private double price;

    // Construtores, getters e setters omitidos para brevidade
}
```

FIGURA 1- EXEMPLO ENTIDADE JPA

Explicação:

- Utilizamos `@EqualsAndHashCode.Include` para garantir que `equals()` e `hashCode()` sejam baseados apenas no campo `id`, que é único para cada entidade. Isso é importante porque o uso de outros campos, especialmente aqueles que podem mudar, pode causar problemas de consistência no estado da entidade.
- Quando `@EqualsAndHashCode` do **Lombok** é utilizado, ele gera automaticamente os métodos `equals()` e `hashCode()`, reduzindo a quantidade de código repetitivo.

2.2.2. ANOTAÇÕES @CACHEABLE COK SPRING CACHE

O Spring oferece suporte a caching por meio da anotação `@Cacheable`. Aqui, `equals()` e `hashCode()` desempenham um papel crítico para definir a chave do cache, que é usada para armazenar e recuperar resultados de métodos.

```

import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;
import java.util.Objects;

@Service
public class UserService {

    @Cacheable("users")
    public User getUserById(Long id) {
        // Simula busca de usuário no banco de dados
        return new User(id, "John Doe");
    }
}

public class User {
    private Long id;
    private String name;

    public User(Long id, String name) {
        this.id = id;
        this.name = name;
    }
}

```

FIGURA 2 - EXEMPLO ANOTAÇÕES

```

}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    User user = (User) o;
    return Objects.equals(id, user.id);
}

@Override
public int hashCode() {
    return Objects.hash(id);
}
}

```

FIGURA 3 - EXEMPLO ANOTAÇÕES

Explicação:

- Aqui, o método `getUserById()` é armazenado em cache com a chave sendo derivada do parâmetro `id`.
- A implementação personalizada de `equals()` e `hashCode()` garante que o cache funcione corretamente ao evitar a inserção de duplicatas.

2.2.3.USO COM SET EM REPOSITÓRIO SPRING DATA

Se você usar uma coleção como `Set` para armazenar entidades dentro de outra entidade (por exemplo, `@OneToMany`), é crucial que `equals()` e `hashCode()` estejam implementados corretamente para evitar problemas de duplicação.

```
import jakarta.persistence.*;
import lombok.EqualsAndHashCode;
import lombok.Data;
import java.util.Set;

@Entity
@Data
public class Author {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy = "author", cascade = CascadeType.ALL)
    @EqualsAndHashCode.Exclude
    private Set<Book> books;
}
```

FIGURA 4 – EXEMPLO DE USO COM SET EM SPRING DATA

```

@Entity
@Data
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @EqualsAndHashCode.Include
    private Long id;

    private String title;

    @ManyToOne
    private Author author;
}

```

FIGURA 5 – EXEMPLO DE USO COM SET EM SPRING DATA

Explicação:

- A entidade Book depende do id como o campo principal para equals() e hashCode().
- Em Author, @EqualsAndHashCode.Exclude é usado para evitar referências circulares e problemas de desempenho ao lidar com coleções Set.

3. LOMBOK: SIMPLIFICAÇÃO DO CÓDIGO

3.1. INTRODUÇÃO À BIBLIOTECA LOMBOK

Lombok é uma biblioteca Java que elimina a necessidade de escrever código boilerplate usando anotações que geram automaticamente métodos como equals(), hashCode(), getters, setters, e toString().

@EqualsAndHashCode:

- Gera automaticamente implementações de equals() e hashCode() com base nos campos da classe.
- Permite personalização para incluir ou excluir certos campos, usando parâmetros como exclude e of.

```
import lombok.EqualsAndHashCode;

@EqualsAndHashCode
public class MyClass {
    private int id;
    private String name;
}
```

FIGURA 6 – EXEMPLO SIMPLIFICAÇÃO DE CÓDIGO COM LOMBOK

@Data:

- Combina múltiplas funcionalidades (@Getter, @Setter, @ToString, @EqualsAndHashCode) para gerar automaticamente código comum em classes de entidades.

3.2. VANTAGENS

Redução de Código Repetitivo:

Lombok elimina a necessidade de escrever métodos comuns como getters, setters, equals(), hashCode(), e toString(), que normalmente resultariam em muitas linhas de código repetitivo (boilerplate). Com a anotação @Data, por exemplo, todos esses métodos são gerados automaticamente. Isso simplifica a manutenção, pois os desenvolvedores não precisam ajustar manualmente esses métodos se os atributos da classe mudarem.

Melhor Legibilidade e Manutenção:

- Com menos código repetitivo, o foco do código-fonte fica nas regras de negócio e na lógica principal, tornando-o mais conciso e fácil de ler. Isso é especialmente benéfico em grandes projetos, onde a clareza do código é crucial.
- Equipes de desenvolvimento se beneficiam com um código mais enxuto, reduzindo a complexidade e facilitando futuras alterações e refatorações.

Facilita o Desenvolvimento Rápido:

- Em contextos ágeis ou onde há necessidade de prototipação rápida, o uso do **Lombok** acelera o processo de desenvolvimento, permitindo que os desenvolvedores se concentrem na funcionalidade e não na infraestrutura do código.

3.3. DESVANTAGENS

Introduz uma Dependência Externa:

- **Lombok** é uma biblioteca de terceiros que precisa ser adicionada como dependência ao projeto. Em ambientes onde há políticas rigorosas de uso de bibliotecas externas (por exemplo, sistemas corporativos de produção com exigências de segurança ou controle de versão), a inclusão de dependências adicionais pode ser um problema.
- Além disso, em ambientes altamente regulados, a inclusão de bibliotecas externas requer análises de segurança e aprovação, o que pode atrasar o ciclo de desenvolvimento.

Potenciais Problemas de Depuração (Debugging):

- Como o **Lombok** gera código em tempo de compilação, os métodos `equals()`, `hashCode()`, `getters`, `setters`, etc., não são visíveis diretamente no código-fonte. Isso pode causar dificuldades durante o processo de depuração (debugging). Ferramentas de IDE podem não mostrar esses métodos no código gerado, tornando mais complicado rastrear problemas.
- Por exemplo, se um bug surgir em um método `equals()` gerado pelo Lombok, os desenvolvedores terão que confiar em ferramentas como **javap** (um descompilador) para inspecionar o bytecode e entender o comportamento exato, em vez de simplesmente revisar o código Java.
- Esse aspecto pode ser desafiador para desenvolvedores menos experientes ou para equipes que não estão familiarizadas com o uso de **Lombok**.

Possíveis Conflitos com Atualizações de Versão:

- Como uma dependência externa, o **Lombok** precisa estar sempre compatível com a versão do **Java** utilizada no projeto. Atualizações no **Java** (por exemplo, de Java 17 para uma versão mais nova) podem causar incompatibilidades, exigindo atualizações na própria biblioteca **Lombok** ou mudanças no código do projeto.
- Em alguns casos, isso pode causar falhas inesperadas, especialmente se o projeto não for mantido regularmente.

Impacto na Portabilidade do Código:

- O uso do **Lombok** pode reduzir a portabilidade do código, especialmente se for necessário transferir o código para um ambiente onde o **Lombok** não está disponível ou permitido.
- Por exemplo, ao compartilhar uma biblioteca interna com outra equipe ou ao migrar um projeto para um novo ambiente, você pode precisar remover ou substituir anotações **Lombok** para garantir compatibilidade.

3.4. BOAS PRÁTICAS DO USO DE LOMBOK

Uso Controlado das Anotações:

- Utilize anotações como `@Data` com cuidado, especialmente em classes que representam entidades persistentes (JPA). Em vez de usar `@Data`, prefira `@Getter`, `@Setter` e `@EqualsAndHashCode` com a configuração `onlyExplicitlyIncluded` para evitar problemas com proxies do Hibernate.

Documentação e Treinamento:

- Garanta que a equipe esteja bem treinada no uso de **Lombok** e compreenda as implicações de suas anotações, para que possam depurar e manter o código com eficiência.

Ferramentas de Integração:

- Algumas IDEs como **IntelliJ IDEA** e **Eclipse** têm suporte nativo para **Lombok** por meio de plugins, o que pode minimizar os problemas de depuração.

CONCLUSÃO

Embora **Lombok** ofereça grandes vantagens, como a redução de código repetitivo e aumento da produtividade, especialmente em projetos ágeis, ele apresenta desafios que devem ser considerados. A dependência externa introduzida por **Lombok** pode ser problemático em ambientes que exigem maior controle sobre bibliotecas, como em empresas que priorizam estabilidade e segurança. Além disso, o código gerado por **Lombok** não é visível diretamente, o que pode dificultar a depuração e tornar o troubleshooting mais complexo.

Em projetos onde a transparência e a manutenção são essenciais, a implementação manual de `equals()` e `hashCode()` pode ser mais vantajosa, pois oferece maior controle sobre o comportamento desses métodos. Contudo, em ambientes ágeis, onde a velocidade de desenvolvimento é crucial, **Lombok** pode ser uma ferramenta poderosa, desde que utilizado com boas práticas para minimizar as dificuldades de manutenção e depuração. O uso de **Lombok** deve ser equilibrado conforme as necessidades do projeto, levando em conta a produtividade e a sustentabilidade do código a longo prazo.

BIBLIOGRAFIA

GUIDES. In: **Spring io.** Disponível em:< <https://spring.io/guides>>. Acesso em: 10 nov, 2024.

PROJECT LOMBOK. In: **Project Lombok.** Disponível em:
< <https://projectlombok.org/>>. Acesso em: 10 nov, 2024.

CLASS HASHMAP. In: **Docs Oracle.** Disponível em: <
<https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>>. Acesso em:
10 nov, 2024.

LISTA DE FIGURAS

Figura 1 – EXEMPLO ENTIDADE JPA, 2024. Acesso em: 10 nov. 2024

Figura 2 – EXEMPLO ANOTAÇÕES, 2024. Acesso em: 10 nov. 2024

Figura 3 – EXEMPLO ANOTAÇÕES, 2024. Acesso em: 10 nov. 2024

Figura 4 – EXEMPLO DE USO COM SET EM SPRING DATA, 2024. Acesso em: 10 nov. 2024

Figura 5 – EXEMPLO DE USO COM SET EM SPRING DATA, 2024. Acesso em: 10 nov. 2024

Figura 6 – EXEMPLO SIMPLIFICAÇÃO DE CÓDIGO COM LOMBOK, 2024.

Acesso em: 10 nov. 2024