****

**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.**

Victoria de Mattos Ferreira

Sorocaba

Novembro-2024



**SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL**

**SENAI “GASPAR RICARDO JUNIOR”**

Victoria de Mattos Ferreira

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

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.

Prof. – Emerson

Cidade

Mês – Ano

HISTÓRICO DE VERSÕES

| **Data** | **Versão** | **Descrição** | **Autor** |
| --- | --- | --- | --- |
| 08/11/2024 | 1.0 | Explorar a importância e o funcionamento dos métodos equals e hashCode em Java, e o Lombok | Victoria de Mattos |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**SUMÁRIO**

[RESUMO 2](#_1fob9te)

[INTRODUÇÃO 2](#_2et92p0)

[1. Fundamentos Teóricos 5](#_tyjcwt)

[1.1. Explicação do contrato e regras entre equals e hashCode: 6](#_3dy6vkm)

[1.2. Como o contrato entre equals e hashCode afeta o comportamento das coleções (ex.: HashMap, HashSet). 7](#_1t3h5sf)

[1.3 Importância da implementação correta de equals e hashCode em entidades de aplicações Java. 8](#_9vj9vb2vf0vd)

[2. Lombok: Simplificação do Código 10](#_4d34og8)

[2.1. Vantagens de usar Lombok em projetos Java 11](#_2s8eyo1)

[2.2. Como o Lombok pode gerar automaticamente os métodos equals e hashCode. 11](#_17dp8vu)

[2.3. Vantagens e Desvantagens de Usar Lombok para equals e hashCode 12](#_3rdcrjn)

[Resumo dos benefícios e desafios de usar equals, hashCode e Lombok. 13](#_26in1rg)

[Relevância dos conceitos aprendidos para o desenvolvimento de aplicações eficientes e escaláveis. 13](#_3hti09bnqh8c)

[Perguntas 14](#_dttfct9qtwx7)

[2.3.1. SUBTOPICO 2 – NÍVEL 2 - TEMA 2 16](#_lnxbz9)

[CONCLUSÃO 17](#_35nkun2)

[BIBLIOGRAFIA 18](#_1ksv4uv)

[LISTA DE FIGURAS 19](#_44sinio)

[LISTA DE TABELAS 20](#_2jxsxqh)

# 

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

# **RESUMO**

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**

Equals: Define a noção de igualdade entre objetos. Ao sobrescrever esse método, você determina quais atributos são relevantes para a comparação e garante que objetos com valores iguais sejam considerados iguais.

HashCode: Calcula um valor hash para o objeto, utilizado em estruturas de dados como HashMap e HashSet. Se dois objetos são considerados iguais por equals, seus valores hash devem ser iguais. Isso garante que objetos iguais sejam armazenados na mesma posição dessas estruturas.

A correta implementação de equals() e hashCode() é crucial para garantir que os objetos funcionem adequadamente em coleções baseadas em hash e no Spring Framework. Ao garantir que os objetos tenham uma comparação lógica precisa e uma distribuição eficiente de hash, você ajuda a evitar problemas de desempenho e inconsistências nas aplicações.

O Lombok é uma poderosa ferramenta para desenvolvedores Java, permitindo a criação de código mais limpo e de fácil manutenção, ao mesmo tempo em que reduz a quantidade de código repetitivo. Ele é particularmente útil em projetos com muitas classes simples, como **POJOs** (Plain Old Java Objects), onde a geração automática de métodos pode economizar tempo e esforço. No entanto, é importante ter cuidado com sua adoção, considerando os prós e contras e garantindo que a equipe esteja familiarizada com seu uso.

# **Fundamentos Teóricos**

.

## **Explicação do contrato e regras entre equals e hashCode:**

1. Se dois objetos são iguais segundo o equals(), eles devem ter o mesmo valor de hashCode().

Ou seja, se a comparação obj1.equals(obj2) retorna true, então a comparação obj1.hashCode() == obj2.hashCode() **deve ser verdadeira**

2. Se dois objetos têm valores de hashCode() diferentes, então eles são considerados diferentes pelo equals().

Essa parte do contrato é implicada, mas não explicitamente declarada. Se dois objetos têm valores diferentes de hashCode(), então eles **não podem** ser considerados iguais pelo método equals(). Ou seja, se dois objetos têm códigos de hash diferentes, você pode ter certeza de que eles são distintos.

3. Se equals() retorna false, isso não implica que hashCode() deva ser diferente.

O contrato entre equals() e hashCode() não exige que dois objetos diferentes em termos de equals() (ou seja, quando equals() retorna false) tenham valores de hashCode() diferentes. Objetos diferentes podem ter o mesmo valor de hashCode(), o que é conhecido como uma **colisão de hash**. No entanto, se dois objetos são considerados iguais (equals() retorna true), seus hashCode() devem ser iguais.

4. Imutabilidade e Consistência de hashCode()

O valor retornado por hashCode() deve ser consistente durante a vida de um objeto. Ou seja, enquanto o objeto não for modificado, o valor de hashCode() não deve mudar. Se um objeto é alterado (por exemplo, se um campo relevante para a comparação de igualdade for alterado), isso pode fazer com que o objeto se comporte de maneira inesperada em coleções baseadas em hash.

.

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

O contrato entre equals() e hashCode() tem um impacto direto e fundamental no comportamento das coleções baseadas em hash em Java, como HashMap, HashSet, Hashtable e LinkedHashMap, entre outras. Essas coleções usam o valor do hashCode() para organizar os objetos internamente e o método equals() para comparar objetos. Se o contrato não for seguido corretamente, isso pode causar problemas sérios, como falhas na busca de objetos, inserções incorretas e inconsistências nas coleções.

**Resumo de Como o Contrato Afeta o Comportamento:**

Comportamento em HashSet: Se equals() não for consistente com hashCode(), o HashSet pode falhar ao identificar objetos como duplicatas, inserindo objetos que deveriam ser considerados iguais.

Comportamento em HashMap: O HashMap pode ter dificuldades para encontrar um objeto corretamente durante a busca ou substituição de valores, se o contrato entre equals() e hashCode() não for seguido.

Colisões de hashCode(): Mesmo com a mesma chave de hash (colisão), o método equals() precisa ser corretamente implementado para garantir que objetos diferentes com o mesmo hashCode() sejam comparados corretamente.

Desempenho e Eficiência: Um bom hashCode() reduz as colisões e melhora o desempenho das coleções baseadas em hash, enquanto um hashCode() mal implementado pode levar a uma alta taxa de colisões, afetando a eficiência da coleção.

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

A implementação correta de **equals()** e **hashCode()** em entidades de aplicações Java é **crucial** para o comportamento esperado da aplicação em diversas situações, especialmente quando as entidades são manipuladas em coleções baseadas em hash ou em frameworks de persistência. Garantir que esses métodos sejam consistentes e sigam o contrato de Java previne problemas como duplicatas em coleções, falhas de comparação, erros de cache e inconsistências nos dados persistidos. Além disso, garante que o sistema seja robusto, eficiente e fácil de manter.

Exemplo 1:

Set<Produto> produtos = new HashSet<>();

Produto p1 = new Produto("ProdutoA", 100);

Produto p2 = new Produto("ProdutoA", 100);

produtos.add(p1);

produtos.add(p2); // Se hashCode() não for consistente, pode inserir dois itens ao invés de um.

Exemplo 2:

Map<Produto, String> mapa = new HashMap<>();

mapa.put(p1, "Item1");

System.out.println(mapa.get(p2)); // Se hashCode() for errado, pode retornar null, mesmo sendo iguais.

Exemplo 3:

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import java.util.Objects;

@Entity

public class Produto {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String nome;

private double preco;

// Construtores

public Produto() {}

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;

}

// Implementação do equals() - baseado no id, pois ele é único

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Produto produto = (Produto) o;

return Objects.equals(id, produto.id);

}

// Implementação do hashCode() - baseado no id, para garantir consistência com equals

@Override

public int hashCode() {

return Objects.hash(id);

}

@Override

public String toString() {

return "Produto{id=" + id + ", nome='" + nome + "', preco=" + preco + "}";

}

}

# **Lombok: Simplificação do Código**

## **Vantagens de usar Lombok em projetos Java**

O Lombok oferece várias vantagens em projetos Java, como redução de código boilerplate, aumento de legibilidade, facilidade de implementação de métodos comuns (como equals() e hashCode()), e criação de classes imutáveis. É uma ferramenta poderosa que pode melhorar significativamente a produtividade e a qualidade do código em projetos Java, especialmente em sistemas grandes e complexos.

No entanto, é importante usar Lombok com cautela, levando em consideração a equipe envolvida, o tipo de projeto e a compatibilidade com ferramentas de depuração e análise de código.

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

O que o Lombok Faz?

O Lombok é uma biblioteca que ajuda a reduzir o código repetitivo em Java. Ele pode gerar automaticamente os métodos equals() e hashCode(), que são essenciais para comparar objetos e funcionar corretamente em coleções baseadas em hash (como HashSet e HashMap).

Sem:

public class Produto {

private String nome;

private double preco;

// Construtor, getters e setters

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Produto produto = (Produto) o;

return Double.compare(produto.preco, preco) == 0 && nome.equals(produto.nome);

}

@Override

public int hashCode() {

return Objects.hash(nome, preco);

}

}

Com:

import lombok.EqualsAndHashCode;

@EqualsAndHashCode

public class Produto {

private String nome;

private double preco;

}

O Lombok vai gerar os métodos equals() e hashCode() automaticamente, usando os campos nome e preco.

## **Vantagens e Desvantagens de Usar Lombok para equals e hashCode**

Vantagens:

Redução de Código: Lombok gera automaticamente os métodos, evitando código repetitivo.

Consistência: Garante que equals() e hashCode() sigam as boas práticas e o contrato entre eles.

Menos Erros: Reduz a chance de erros como esquecer campos ou criar implementações inconsistentes.

Facilidade de Manutenção: Atualiza automaticamente os métodos quando você altera os campos da classe.

Produtividade: Economiza tempo e permite focar em outras partes da aplicação.

Desvantagens:

Dependência Externa: Introduz a dependência de uma biblioteca adicional no projeto.

Dificuldade de Depuração: O código gerado não é visível, o que pode complicar a depuração.

Menos Controle: Você perde flexibilidade para personalizar completamente os métodos.

Ocultação de Comportamento: Desenvolvedores podem não entender que os métodos são gerados automaticamente.

Incompatibilidade com Ferramentas de Análise: Algumas ferramentas podem não entender o código gerado pelo Lombok.

Desempenho: Pode impactar o desempenho em casos muito específicos (grandes classes ou coleções).

## Resumo dos benefícios e desafios de usar equals, hashCode e Lombok.

Redução de Código: Lombok gera automaticamente os métodos, economizando tempo e evitando código repetitivo.

Consistência: Garante que os métodos sejam implementados de maneira correta e consistente.

Facilidade de Manutenção: Atualiza automaticamente os métodos quando os campos da classe mudam, facilitando a manutenção do código.

Desafios ao Usar Lombok para equals() e hashCode()

Dependência Externa: Adiciona uma dependência no projeto, o que pode não ser desejável em alguns contextos.

Menos Controle: Pode não ser adequado quando é necessário personalizar profundamente a lógica de equals() e hashCode().

Dificuldade de Depuração: O código gerado pelo Lombok não é visível, o que pode dificultar a depuração e a análise do comportamento do código.

Incompatibilidade com Ferramentas: Algumas ferramentas de análise de código podem não entender o código gerado pelo Lombok.

## Relevância dos conceitos aprendidos para o desenvolvimento de aplicações eficientes e escaláveis.

A correta implementação de equals() e hashCode() é crucial para garantir a eficiência e escalabilidade de coleções em Java, como HashMap e HashSet, otimizando operações de busca e inserção.

Lombok facilita esse processo, gerando automaticamente esses métodos, o que reduz código repetitivo, aumenta a produtividade, e garante consistência. No entanto, ele introduz dependências externas e pode dificultar a depuração e o controle personalizado da lógica.

## Perguntas

Por que é importante implementar corretamente equals e hashCode em uma classe Java?

A implementação correta de equals() e hashCode() é fundamental para garantir que objetos possam ser comparados de maneira consistente e eficiente, especialmente quando são usados em coleções como **HashMap**, **HashSet**, e outras coleções baseadas em hash. As principais razões para implementá-los corretamente são:

O HashMap<K, V> utiliza generics, onde **K** representa o tipo da chave e **V** o tipo do valor. Isso permite que você defina de forma segura e clara os tipos de dados que serão armazenados no mapa, garantindo que as chaves e os valores sigam o tipo especificado. Isso ajuda a evitar erros em tempo de execução e melhora a legibilidade e manutenção do código.

O **HashSet** é uma implementação da interface **Set**, que faz parte da coleção de dados em Java. Ele armazena elementos de forma que **não permite elementos duplicados** e **não é indexado**. Vamos explicar isso com mais detalhes:

Consistência e Integridade: O contrato entre equals() e hashCode() especifica que se dois objetos são considerados iguais (equals() retorna true), então eles devem ter o mesmo valor de hashCode(). Isso é crucial para garantir que objetos iguais sejam tratados corretamente em coleções baseadas em hash.

Funcionamento de Coleções: Em coleções como HashMap e HashSet, o hashCode() determina como os objetos são armazenados e distribuídos internamente. Se equals() e hashCode() não forem implementados corretamente, o comportamento das coleções pode ser inesperado (por exemplo, objetos iguais podem ser armazenados ou buscados incorretamente).

Como o uso de equals e hashCode impacta o comportamento de coleções como HashSet e HashMap?

As coleções como HashSet e HashMap dependem dos métodos equals() e hashCode() para armazenar e recuperar objetos de maneira eficiente:

HashMap:

O hashCode() é usado para determinar a posição (bucket) onde o objeto será armazenado na tabela interna do mapa.

Quando você tenta acessar um valor usando uma chave, o hashCode() é usado para localizar rapidamente o bucket. Dentro do bucket, o método equals() é chamado para comparar a chave e garantir que o valor correto seja retornado.

Se equals() e hashCode() não forem consistentes (por exemplo, se dois objetos iguais tiverem valores de hashCode() diferentes), o HashMap pode não conseguir encontrar o valor correto associado à chave, resultando em inconsistências e falhas de recuperação.

HashSet:

Funciona de forma semelhante ao HashMap, mas armazena objetos ao invés de pares chave-valor. O hashCode() é usado para determinar a localização do objeto, e equals() é usado para verificar se o objeto já está presente no conjunto.

Se os métodos não forem implementados corretamente, objetos iguais podem ser tratados como diferentes, resultando em duplicatas indesejadas ou na impossibilidade de encontrar elementos.

Como o Lombok simplifica a criação dos métodos equals e hashCode? Existem desvantagens no uso do Lombok?

Como Lombok simplifica:

Geração Automática: O Lombok usa as anotações @EqualsAndHashCode e @Data (que inclui equals() e hashCode()) para gerar automaticamente esses métodos de maneira consistente e sem erros.

Redução de Código Boilerplate: Evita a necessidade de escrever manualmente a lógica de comparação e cálculo de hash para cada classe, o que reduz o código repetitivo e melhora a legibilidade.

Consistência: O Lombok segue boas práticas na geração dos métodos, garantindo que os campos corretos sejam usados e que o contrato entre equals() e hashCode() seja mantido corretamente.

Desvantagens do Lombok:

Dependência Externa: O uso do Lombok adiciona uma dependência extra no projeto, o que pode ser indesejável em projetos que exigem menos dependências ou onde o Lombok não é suportado.

Opacidade na Depuração: Como o código é gerado automaticamente, ele não aparece no código-fonte, o que pode dificultar a depuração e o entendimento do comportamento de equals() e hashCode().

Menos Controle: Lombok pode não fornecer flexibilidade para casos em que é necessário um comportamento personalizado, como lógica condicional complexa nos métodos.

Em quais situações específicas o uso de Lombok seria preferível em relação à implementação manual de equals e hashCode?

Situações em que Lombok é preferível:

Classes Simples: Para classes simples, onde os campos envolvidos em equals() e hashCode() são diretos e não há necessidade de lógica complexa, o Lombok é uma escolha excelente para reduzir a verbosidade e evitar erros comuns.

Manutenção em Grande Escala: Em projetos grandes, onde as classes frequentemente mudam (adicionando ou removendo campos), o Lombok ajuda a manter equals() e hashCode() atualizados automaticamente sem exigir modificação manual constante.

Projetos com Múltiplos Desenvolvedores: Em equipes grandes, o uso de Lombok ajuda a garantir que todos sigam o mesmo padrão para esses métodos, garantindo consistência e reduzindo a chance de erros humanos.

Produtividade: Em projetos onde agilidade e produtividade são prioritárias, e onde a personalização dos métodos não é uma exigência, Lombok é uma boa solução para automatizar a geração desses métodos.

Situações onde a implementação manual é preferível:

Lógica Complexa: Quando é necessário personalizar a lógica de comparação (por exemplo, incluindo ou excluindo campos com base em condições específicas) ou otimizar a geração do hashCode() para casos de alto desempenho, a implementação manual é preferível.

Controle Fino: Se você precisa de controle total sobre a implementação de equals() e hashCode(), como garantir um comportamento específico com base em alguns campos (como tratar campos nulos ou mutáveis), a implementação manual é mais adequada.

# **CONCLUSÃO**

O vídeo fornece uma maneira poderosa de ajudá-lo a provar seu argumento. Ao clicar em Vídeo Online, você pode colar o código de inserção do vídeo que deseja adicionar.

Você também pode digitar uma palavra-chave para pesquisar online o vídeo mais adequado ao seu documento. Para dar ao documento uma aparência profissional, o Word fornece designs de cabeçalho, rodapé, folha de rosto e caixa de texto que se complementam entre si.

# **BIBLIOGRAFIA**

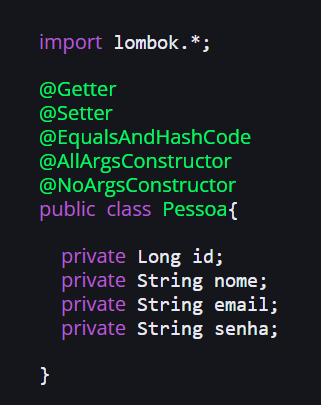
<https://medium.com/@angeliski/equals-e-hashcode-o-que-%C3%A9-porque-usar-e-como-implementar-1ee70fa67396>

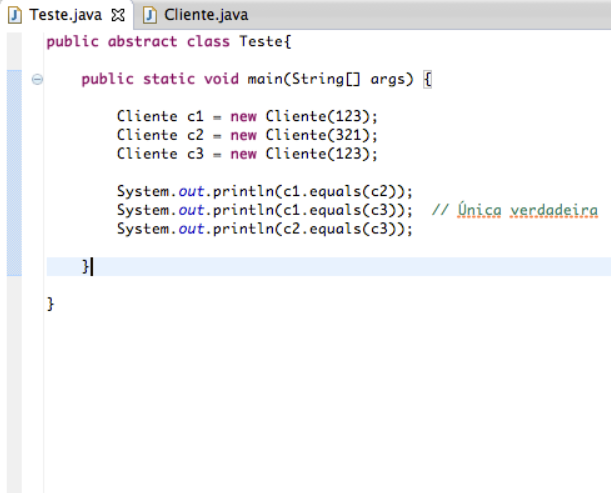
<https://www.devmedia.com.br/sobrescrevendo-o-metodo-equals-em-java/26484>

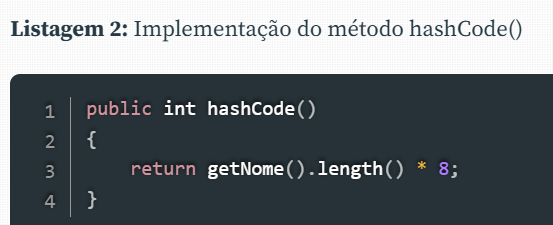
<https://www.devmedia.com.br/sobrescrevendo-o-metodo-hashcode-em-java/26488>

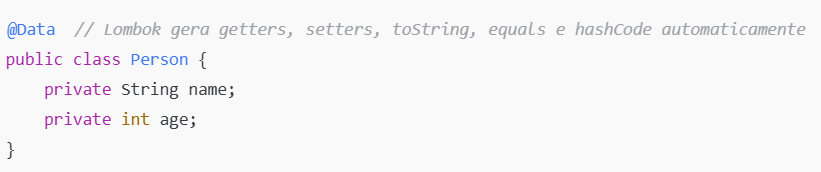
<https://www.devmedia.com.br/uma-visao-sobre-o-projeto-lombok/28321>

# **LISTA DE FIGURAS**

**Lombok**

**Equals**

**hashcode**



lombok