**HashCode e Equals**

Os métodos hashCode e equals são fundamentais em Java, especialmente ao trabalhar com coleções como HashMap, HashSet e Hashtable. Eles são usados para determinar a igualdade de objetos e a forma como eles são armazenados em estruturas de dados baseadas em hash.

**1. equals**

O método equals verifica se dois objetos são "logicamente iguais". A implementação padrão na classe Object compara referências, ou seja, dois objetos só são considerados iguais se ocuparem o mesmo espaço na memória.

**Exemplo de equals:**

class Pessoa {

private String nome;

private int idade;

public Pessoa(String nome, int idade) {

this.nome = nome;

this.idade = idade;

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

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

Pessoa pessoa = (Pessoa) obj;

return idade == pessoa.idade && nome.equals(pessoa.nome);

}

@Override

public int hashCode() {

return Objects.hash(nome, idade);

}

}

public class Main {

public static void main(String[] args) {

Pessoa p1 = new Pessoa("João", 30);

Pessoa p2 = new Pessoa("João", 30);

System.out.println(p1.equals(p2)); // true

}

}

Aqui, a implementação personalizada de equals verifica se dois objetos Pessoa têm o mesmo nome e idade.

**2. hashCode**

O método hashCode retorna um valor inteiro que representa o objeto. Esse valor é usado em coleções baseadas em hash (como HashSet e HashMap) para localizar objetos rapidamente. Se dois objetos são iguais pelo método equals, eles **devem** ter o mesmo hashCode.

**Exemplo de hashCode:** No exemplo anterior, usamos Objects.hash para gerar um código hash com base nos atributos nome e idade.

Se sobrescrevermos apenas o hashCode sem sobrescrever equals, ou vice-versa, o comportamento das coleções pode ser imprevisível.

**Problema ao não sobrescrever equals e hashCode:**

public class Main {

public static void main(String[] args) {

HashSet<Pessoa> pessoas = new HashSet<>();

Pessoa p1 = new Pessoa("Maria", 25);

Pessoa p2 = new Pessoa("Maria", 25);

pessoas.add(p1);

pessoas.add(p2);

System.out.println(pessoas.size()); // Pode imprimir 2, se `hashCode` ou `equals` não forem sobrescritos corretamente.

}

}

**3. Por que sobrescrever ambos?**

Quando trabalhamos com coleções como HashMap ou HashSet, o processo de verificação de duplicatas ou busca envolve duas etapas:

1. **hashCode:** Primeiro, verifica o código hash do objeto. Isso é rápido, mas pode gerar colisões (dois objetos diferentes com o mesmo código hash).
2. **equals:** Se os códigos hash forem iguais, o método equals é usado para verificar se os objetos são realmente iguais.

**Exemplo completo com ambos sobrescritos:**

class Produto {

private String nome;

private double preco;

public Produto(String nome, double preco) {

this.nome = nome;

this.preco = preco;

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

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

Produto produto = (Produto) obj;

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

}

@Override

public int hashCode() {

return Objects.hash(nome, preco);

}

}

public class Main {

public static void main(String[] args) {

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

Produto prod1 = new Produto("Cadeira", 199.99);

Produto prod2 = new Produto("Cadeira", 199.99);

produtos.add(prod1);

produtos.add(prod2);

System.out.println(produtos.size()); // 1, pois equals e hashCode foram sobrescritos corretamente.

}

}

**4. Resumo**

 Use equals para verificar a igualdade lógica entre objetos.

 Use hashCode para otimizar buscas e inserções em coleções baseadas em hash.

 Sempre sobrescreva equals e hashCode juntos para evitar inconsistências.

 Classes como String, Integer e Double já possuem implementações apropriadas, mas classes personalizadas precisam dessas implementações para garantir comportamento correto.

As classes do Java, como String, Integer e Double, já possuem implementações de equals e hashCode que garantem o comportamento esperado ao comparar valores. Por outro lado, classes personalizadas, como Pessoa ou Produto, precisam de implementações específicas para que possam ser comparadas corretamente.

**Exemplo com String**

A classe String compara os valores dos caracteres ao verificar igualdade, não a referência na memória.

public class Main {

public static void main(String[] args) {

String str1 = new String("Java");

String str2 = new String("Java");

// Comparação com equals

System.out.println(str1.equals(str2)); // true

// Comparação de referência

System.out.println(str1 == str2); // false

// hashCode

System.out.println(str1.hashCode() == str2.hashCode()); // true

}

}

* **equals em String:** Verifica se os conteúdos das strings são iguais.
* **hashCode em String:** Calcula o hash com base no conteúdo dos caracteres.

**Exemplo com Integer e Double**

As classes Integer e Double usam o valor numérico para definir igualdade.

public class Main {

public static void main(String[] args) {

Integer num1 = 100;

Integer num2 = 100;

// Comparação com equals

System.out.println(num1.equals(num2)); // true

// Comparação de referência

System.out.println(num1 == num2); // true (valores de -128 a 127 são cacheados)

Double d1 = 100.5;

Double d2 = 100.5;

// Comparação com equals

System.out.println(d1.equals(d2)); // true

// hashCode

System.out.println(d1.hashCode() == d2.hashCode()); // true

}

}

**Por que classes personalizadas precisam sobrescrever equals e hashCode?**

As classes personalizadas herdam a implementação padrão de Object, que compara referências. Para garantir a igualdade lógica (comparar atributos), precisamos sobrescrever os métodos.

**Exemplo sem sobrescrita**

class Pessoa {

String nome;

int idade;

public Pessoa(String nome, int idade) {

this.nome = nome;

this.idade = idade;

}

}

public class Main {

public static void main(String[] args) {

Pessoa p1 = new Pessoa("Ana", 25);

Pessoa p2 = new Pessoa("Ana", 25);

System.out.println(p1.equals(p2)); // false, compara referências

System.out.println(p1.hashCode() == p2.hashCode()); // false, códigos hash diferentes

}

}

**Exemplo com sobrescrita de equals e hashCode**

class Pessoa {

String nome;

int idade;

public Pessoa(String nome, int idade) {

this.nome = nome;

this.idade = idade;

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

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

Pessoa pessoa = (Pessoa) obj;

return idade == pessoa.idade && nome.equals(pessoa.nome);

}

@Override

public int hashCode() {

return Objects.hash(nome, idade);

}

}

public class Main {

public static void main(String[] args) {

Pessoa p1 = new Pessoa("Ana", 25);

Pessoa p2 = new Pessoa("Ana", 25);

System.out.println(p1.equals(p2)); // true, igualdade lógica

System.out.println(p1.hashCode() == p2.hashCode()); // true, mesmo hash

}

}

**Resumo**

|  |  |  |
| --- | --- | --- |
| Classe | Implementação de equals | Implementação de hashCode |
| String | Compara os caracteres | Baseado nos caracteres da string |
| Integer | Compara os valores numéricos | Baseado no valor numérico |
| Double | Compara os valores numéricos | Baseado no valor numérico |
| Personalizada | Necessita sobrescrita para comparar atributos | Necessita sobrescrita para refletir equals |

As classes como String, Integer e Double já garantem igualdade lógica por padrão, enquanto classes personalizadas exigem implementações explícitas para que coleções e comparações funcionem corretamente.