**CompareTo**

O método compareTo em Java é utilizado para comparar dois objetos para determinar a ordem entre eles. Ele faz parte da interface **Comparable**, que é usada para definir a ordem natural dos objetos de uma classe. O método retorna um valor inteiro que indica se o objeto atual é menor, igual ou maior em relação ao objeto comparado.

**Assinatura do método compareTo**

public int compareTo(T o);

**Retorno do método**

* **Valor negativo (< 0):** O objeto atual é menor que o objeto comparado.
* **Valor zero (0):** O objeto atual é igual ao objeto comparado.
* **Valor positivo (> 0):** O objeto atual é maior que o objeto comparado.

**Quando usar o compareTo**

1. **Ordenação em coleções:**
   * É usado pelo método Collections.sort() para ordenar listas.
   * Estruturas como TreeSet e TreeMap usam compareTo para manter os elementos ordenados.
2. **Comparação personalizada:**
   * Para implementar a lógica de ordenação específica para uma classe personalizada.

**Exemplo simples com números**

public class Main {

public static void main(String[] args) {

Integer num1 = 10;

Integer num2 = 20;

System.out.println(num1.compareTo(num2)); // -1 (10 < 20)

System.out.println(num2.compareTo(num1)); // 1 (20 > 10)

System.out.println(num1.compareTo(10)); // 0 (10 == 10)

}

}

**Exemplo com classe personalizada**

**1. Classe implementando Comparable**

class Pessoa implements Comparable<Pessoa> {

private String nome;

private int idade;

public Pessoa(String nome, int idade) {

this.nome = nome;

this.idade = idade;

}

public String getNome() {

return nome;

}

@Override

public int compareTo(Pessoa outra) {

// Ordenar por idade

return Integer.compare(this.idade, outra.idade);

}

@Override

public String toString() {

return nome + " (" + idade + ")";

}

}

2. Ordenação usando compareTo

import java.util.ArrayList;

import java.util.Collections;

public class Main {

public static void main(String[] args) {

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

pessoas.add(new Pessoa("Ana", 30));

pessoas.add(new Pessoa("João", 25));

pessoas.add(new Pessoa("Maria", 35));

// Ordenação usando compareTo

Collections.sort(pessoas);

System.out.println(pessoas);

// Saída: [João (25), Ana (30), Maria (35)]

}

}

**Quando usar outra abordagem: Comparator**

Se você precisar de múltiplos critérios de ordenação ou não puder alterar a classe para implementar Comparable, use a interface **Comparator**.

**Exemplo com Comparator**

import java.util.Comparator;

class ComparadorPorNome implements Comparator<Pessoa> {

@Override

public int compare(Pessoa p1, Pessoa p2) {

return p1.getNome().compareTo(p2.getNome());

}

}

public class Main {

public static void main(String[] args) {

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

pessoas.add(new Pessoa("Ana", 30));

pessoas.add(new Pessoa("João", 25));

pessoas.add(new Pessoa("Maria", 35));

// Ordenar por nome usando Comparator

pessoas.sort(new ComparadorPorNome());

System.out.println(pessoas);

// Saída: [Ana (30), João (25), Maria (35)]

}

}

**Resumo**

* Use **compareTo** quando:
  + Você deseja definir uma **ordem natural** para uma classe.
  + Sua classe pode implementar a interface Comparable.
* Retorna um valor que indica a relação de ordem entre os objetos.
* Em cenários mais complexos, use **Comparator** para maior flexibilidade.

**Diferenças entre equals, hashCode e compareTo**

Esses três métodos desempenham papéis distintos na manipulação e comparação de objetos em Java. Aqui está uma explicação detalhada:

**1. equals**

* **Função:** Verifica se dois objetos são **iguais logicamente**.
* **Assinatura:**

public boolean equals(Object obj)

* **Uso:** Determina se dois objetos representam o mesmo valor ou estado, com base em uma implementação personalizada.
* **Comportamento padrão:** O método da classe Object compara referências, ou seja, verifica se os objetos ocupam o mesmo endereço de memória.

**Exemplo:**

String str1 = new String("Java");

String str2 = new String("Java");

// Comparação lógica

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

// Comparação de referência

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

**Quando sobrescrever?**

Sobrescreva quando precisar comparar atributos de objetos em vez de referências.

**2. hashCode**

* **Função:** Retorna um valor inteiro (código hash) que representa o objeto para fins de **armazenamento eficiente** em coleções baseadas em hash, como HashMap, HashSet e HashTable.
* **Assinatura:**

public int hashCode()

* **Uso:** Usado em estruturas de dados para identificar rapidamente um bucket onde o objeto pode estar armazenado.
* **Comportamento padrão:** O método da classe Object gera um código baseado na referência do objeto na memória.

**Regra de contrato:**

Se dois objetos são iguais (equals retorna true), eles devem ter o mesmo código hash (hashCode).

**Exemplo:**

String str1 = "Java";

String str2 = "Java";

// hashCode dos valores iguais será o mesmo

System.out.println(str1.hashCode()); // Exemplo: 2301506

System.out.println(str2.hashCode()); // Exemplo: 2301506

**Quando sobrescrever?**

Sobrescreva sempre que sobrescrever o equals, para manter o contrato entre os dois métodos.

**3. compareTo**

* **Função:** Compara dois objetos para determinar sua **ordem** (menor, igual ou maior).
* **Assinatura:**

public int compareTo(T o)

* **Uso:** Implementado na interface Comparable para definir a **ordem natural** de objetos.
* **Retorno:**
  + Valor **negativo**: O objeto atual é menor.
  + Valor **zero**: O objeto atual é igual.
  + Valor **positivo**: O objeto atual é maior.

**Exemplo:**

Integer num1 = 10;

Integer num2 = 20;

System.out.println(num1.compareTo(num2)); // -1 (10 < 20)

System.out.println(num2.compareTo(num1)); // 1 (20 > 10)

System.out.println(num1.compareTo(10)); // 0 (10 == 10)

**Quando usar?**

Implemente compareTo quando precisar de uma ordem natural para a classe ou quiser usar coleções ordenadas, como TreeSet ou TreeMap.

**Comparação**

|  |  |  |  |
| --- | --- | --- | --- |
| Característica | equals | hashCode | compareTo |
| Objetivo | Verificar igualdade lógica | Gerar código hash | Comparar ordem (menor, maior, igual) |
| Retorno | true ou false | Valor inteiro (int) | Valor inteiro (int) |
| Uso comum | Comparar objetos em geral | Estruturas de dados baseadas em hash | Ordenação de objetos |
| Assinatura | boolean equals(Object obj) | int hashCode() | int compareTo(T o) |
| Contrato | Igualdade lógica | Igualdade lógica implica hash igual | Ordem natural |

**Exemplo combinando os três**

**Classe Pessoa com equals, hashCode e compareTo:**

import java.util.Objects;

class Pessoa implements Comparable<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);

}

@Override

public int compareTo(Pessoa outra) {

return Integer.compare(this.idade, outra.idade); // Ordenar por idade

}

@Override

public String toString() {

return nome + " (" + idade + ")";

}

}

public class Main {

public static void main(String[] args) {

Pessoa p1 = new Pessoa("Ana", 30);

Pessoa p2 = new Pessoa("Ana", 30);

Pessoa p3 = new Pessoa("João", 25);

// equals e hashCode

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

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

// compareTo

System.out.println(p1.compareTo(p3)); // 1 (30 > 25)

System.out.println(p3.compareTo(p1)); // -1 (25 < 30)

System.out.println(p1.compareTo(p2)); // 0 (30 == 30)

}

}

**Resumo prático**

* **equals:** Verifica se dois objetos são iguais.
* **hashCode:** Usado para otimizar armazenamento em coleções baseadas em hash.
* **compareTo:** Define a ordem natural dos objetos para coleções ordenadas.