02

Teste a igualdade entre Strings e outros objetos usando == e equals() - Parte 3

O método equals

Para comparar duas referências, podemos sempre usar o operador `==`. Dada a classe `Cliente`:

```
class Cliente {
    private String nome;
    Cliente(String nome) {
        this.nome = nome;
    }
}
```

COPIAR CÓDIGO

```
Cliente c1 = new Cliente("guilherme");
Cliente c2 = new Cliente("mario");
System.out.println(c1==c2); // false
System.out.println(c1==c1); // true

Cliente c3 = new Cliente("guilherme");
System.out.println(c1==c3);
// false, pois não é a mesma
// referência: são objetos diferentes na memória
```

COPIAR CÓDIGO

Para comparar os objetos de uma outra maneira, que não através da referência, podemos utilizar o método `equals`, cujo comportamento padrão é fazer a simples comparação com o `==`:

```
Cliente c1 = new Cliente("guilherme");
Cliente c2 = new Cliente("mario");
System.out.println(c1.equals(c2)); // false
System.out.println(c1.equals(c1)); // true

Cliente c3 = new Cliente("guilherme");
System.out.println(c1.equals(c3));
// false, pois não é a mesma
// referência: são objetos diferentes na memória
```

COPIAR CÓDIGO

Isso é, existe um método em `Object` que você pode reescrever para definir um **critério de comparação de igualdade**. Classes como `String`, `Integer` e muitas outras possuem esse método reescrito, assim `new Integer(10) == new Integer(10)` dá `false`, mas `new Integer(10).equals(new Integer(10))` dá `true`.

É interessante reescrever esse método quando você julgar necessário um critério de igualdade diferente que o `==` retorna. Imagine o caso de nosso `Cliente`:

```
class Cliente {
    private String nome;
    Cliente(String nome) {
        this.nome = nome;
    }
```

```
    public boolean equals(Object o) {
        if (! (o instanceof Cliente)) {
            return false;
        }
        Cliente outro = (Cliente) o;
        return this.nome.equals(outro.nome);
    }
}
```

COPIAR CÓDIGO

O método `equals` não consegue tirar proveito do `generics`, então precisamos receber `Object` e ainda verificar se o tipo do objeto passado como argumento é realmente uma `Cliente` (o contrato do método diz que você deve retornar `false`, e não deixar lançar exception em um caso desses). Agora sim, podemos usar o método `equals` como esperamos:

```
Cliente c1 = new Cliente("guilherme");
Cliente c2 = new Cliente("mario");
System.out.println(c1.equals(c2)); // false
System.out.println(c1.equals(c1)); // true

Cliente c3 = new Cliente("guilherme");
System.out.println(c1.equals(c3)); // true
```

COPIAR CÓDIGO

Cuidado ao sobrescrever o método `equals`: ele deve ser público, e deve receber `Object`. Caso você receba uma referência a um objeto do tipo `Cliente`, seu método não está sobrescrevendo aquele método padrão da classe `Object`, mas sim criando um novo método (overload). Por polimorfismo o compilador fará funcionar neste caso pois o compilador fará a conexão ao método mais específico, entre `Object` e `Cliente`, ele escolherá o método que recebe `Client`

```
class Cliente {
    private String nome;
    Cliente(String nome) {
        this.nome = nome;
    }

    public boolean equals(Cliente outro) {
        return this.nome.equals(outro.nome);
    }
}
```

COPIAR CÓDIGO

```
Cliente c1 = new Cliente("guilherme");
Cliente c2 = new Cliente("mario");
System.out.println(c1.equals(c2)); // false
System.out.println(c1.equals(c1)); // true

Cliente c3 = new Cliente("guilherme");
System.out.println(c1.equals(c3)); // true
System.out.println(c1.equals((Object) c3));
// false, o compilador não sabe que Object é cliente,
// invoca o equals tradicional, e azar do desenvolvedor
```

COPIAR CÓDIGO

Mas caso você use alguma biblioteca (como a API de coleções e de `ArrayList` do Java), o resultado não será o esperado.