01

Equals e hashcode

Transcrição

Agora, queremos usufruir da grande vantagem dos `Set`s: **a velocidade**. Queremos perguntar para coleção, por exemplo, se determinado aluno está matriculado:

```
System.out.println("O aluno " + a1.getNome() + " está matriculado?");
System.out.println(javaColecoes.estaMatriculado(a1));
```

Novamente, usaremos o TDD para criar o método automaticamente, basta usar o comando `CTRL + 1` e o Eclipse criará para você. Porém precisamos alterar o retorno para `boolean`. Já dentro do método, utilizaremos-nos de um outro método que está presente em todas as classes que implementam a interface `Collection`, o `contains`. Com isso, vamos delegar a funcionalidade do método `estaMatriculado` para um já existente:

```
public boolean estaMatriculado(Aluno aluno) {
    return this.alunos.contains(aluno);
}
```

Agora, o `contains` utilizará a estrutura bem implementada da **tabela de espalhamento**, e irá retornar rapidamente `true` ou `false` para nós. Testando... Funciona! O aluno inserido algumas linhas antes, `a1`, está matriculado no curso. Se comentarmos a linha de inserção, o método nos retornará `false`.

O método equals

Porém, existe um grande problema, bastante comum ao trabalhar com conjuntos, o problema do `equals`. Imagine que estamos nos utilizando de um *web service* e ele possui um formulário perguntando quem estamos procurando. Se vamos digitar no formulário, o seu retorno será uma String com o nome do aluno procurado:

```
String alunoProcurado = "Rodrigo Turini";
```

Não podemos procurá-lo com o nosso método anterior, pois o método `estaMatriculado` recebe um objeto do tipo Aluno como parâmetro. Então vamos criar um objeto exatamente igual ao aluno `a1` criado anteriormente, e passá-lo na função, para saber se ele está ou não dentro do curso:

```
Aluno turini = new Aluno("Rodrigo Turini", 34672);
System.out.println("E esse Turini, está matriculado?");
System.out.println(javaColecoes.estaMatriculado(turini));
```

Testando e... Deu `false`! Esse `turini` não é o mesmo aluno que adicionamos no curso como `a1`. Mas isso já sabíamos da orientação a objetos, se dermos um `new`, mesmo que o objeto contenha tudo igual, ele não fará referência ao primeiro, e portanto, são diferentes. Você pode testar executando o exemplo abaixo:

```
public class TestaCursoComAluno {

    public static void main(String[] args) {

        Aluno aluno = new Aluno("Douglas Quintanilha", 11824763);
        Aluno alunoQueVeioDoFormulario = new Aluno("Douglas Quintanilha", 11824763);

        System.out.println("O aluno e igual ao aluno que veio do formulario?");
        System.out.println(aluno == alunoQueVeioDoFormulario);
    }
}

public class Aluno {

    private String nome;
    private int numeroMatricula;

    public Aluno(String nome, int numeroMatricula) {
        this.nome = nome;
        this.numeroMatricula = numeroMatricula;
    }
}
```

Olhando a documentação da interface `Collection` e indo no método `contains`, veremos que ele utiliza o método `equals`. Sabemos que a definição do `equals` usada pelo Java nem sempre é a que queremos. Usando nosso caso de alunos, sabemos que `a1.equals(turini) == true`, porém isso não é verdadeiro para o Java.

Por isso, precisamos reescrever o método `equals` na nossa classe `Aluno`. Para nós, dois alunos são iguais se ambos tiverem o mesmo nome, então vamos ao trabalho:

```
@Override
public boolean equals(Object obj) {
    Aluno outroAluno = (Aluno) obj;
    return this.nome.equals(outroAluno.nome);
}
```

Lembrando que é preciso ter cuidado nos casos em que o nome seja `null`. Podemos nos defender disso colocando uma condição no construtor em que só seja possível criar o objeto se o nome não for `null`:

```
public Aluno(String nome, int numeroMatricula) {
    if (nome == null) {
        throw new NullPointerException("Nome não pode ser nulo");
    }
    this.nome = nome;
    this.numeroMatricula = numeroMatricula;
}
```

E assim garantimos que no método `equals`, não teremos problemas com `NullPointerException`. Vamos testar agora e ver se "a1 é equals ao Turini":

```
System.out.println("O a1 é equals ao Turini?");
System.out.println(a1.equals(turini));
```

E sim, é `true` ! Porém, nossa comparação de cima ainda não funciona:

```
Aluno turini = new Aluno("Rodrigo Turini", 34672);
System.out.println("E esse Turini, está matriculado?");
System.out.println(javaColecoes.estaMatriculado(turini));
```

Temos `false` como resultado. No entanto, se mudamos o `equals` , por que ele continua dizendo que `turini` não está matriculado? Ao comparar `turini` com `a1` , o resultado é `true` (como visto no nosso teste), porém o `estaMatriculado` nos retorna `false` .

O método hashCode

Vamos à explicação: a estrutura `Set` usa uma **tabela de espalhamento** para realizar mais rapidamente suas buscas. Imagine que cada vez que você adiciona algo dentro do seu `Set` para espalhar os objetos, um número mágico é gerado e todos os objetos que o tenham são agrupados. E ao buscar, em vez de comparar o objeto com todos os outros objetos contidos dentro do `Set` (isso daria muitas comparações), ele gera novamente o mesmo número mágico, e compara apenas com aqueles que também tiveram como resultado esse número. Ou seja, ele compara apenas dentro do grupo de semelhança. No caso da matricula não reconhecida, o aluno `a1` estava num grupo diferente de `turini` , e por isso o método `contains` não conseguia encontrá-lo.

Como é gerado esse número mágico? Utilizando o método `hashCode` , por isso precisamos sobrescrevê-lo, mudando-o para quando criarmos um objeto `Aluno` com o mesmo nome, que esses objetos gerem o mesmo `hashCode` e portanto, fiquem no mesmo grupo. Podemos por exemplo pegar o primeiro caractere do nome. Dessa maneira, estaremos dividindo os grupos em grupos de alunos que começam com **A, B, C, D, ...**, e `Rodrigo Turini` tanto em `a1` quanto em `turini` estarão no grupo **R**:

```
@Override
public int hashCode(){
    return this.nome.charAt(0);
}
```

Testando, vemos que funciona! Mas temos outro probleminha... O espalhamento é feito para que se tenha o menor número possível de objetos dentro de um grupo, e separando alfabeticamente como estamos fazendo, não é a maneira mais eficiente. Entrando na classe `String` do Java, vemos que ela tem o método `hashCode` implementado, e ele já faz uma conta bem difícil, para que haja o melhor espalhamento e assim, a busca seja bastante eficiente. Então, podemos fazer com que o nosso `hashCode` devolva o `hashCode` da `String` `nome` :

```
@Override
public int hashCode(){
    return this.nome.hashCode();
}
```

Se rodarmos o código novamente, temos `true` em todos os testes. Considere a seguinte **regra**: caso você sobrescreva o método `equals` , obrigatoriamente deverá sobrescrever o método `hashCode` .

O que aprendemos neste capítulo:

- Implementação das nossas próprias regras de comparação entre objetos de uma mesma classe.
- Sobrescrita do método `equals`.
- A necessidade de sobrescrever o método `hashCode` quando o `equals` for sobrescrito.