

Herança II - Classe Object

Programação Orientada a Objetos — QXD0007



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará

2º semestre/2021



Leituras para esta aula

- **Capítulo 9** (Herança, reescrita e polimorfismo) da apostila da Caelum – Curso FJ-11, Disponível no link: <https://www.caelum.com.br/apostila/apostila-java-orientacao-objetos.pdf>
- **Capítulo 9** (Herança) do livro Java Como Programar, Décima Edição, Disponível no link: <http://libgen.lc/ads.php?md5=728636A04ACA056038BB5F079403AC96>
- **Capítulo 8** (Reutilização de classes) do livro Introdução à Programação Orientada a Objetos usando Java, Rafael Santos, Disponível no link: https://www.academia.edu/6227746/Introdu%C3%A7%C3%A3o_%C3%A0_Programa%C3%A7%C3%A3o_Orientada_a_Objeto_Usando_Java

Superclasse Object



A superclasse Object

- **Toda** classe em Java **extends** a classe `java.lang.Object`
 - Não precisa herdar explicitamente

A superclasse Object

- **Toda** classe em Java **extends** a classe `java.lang.Object`
 - Não precisa herdar explicitamente
- Como toda classe em Java **é-um Object**, podemos usar uma variável do tipo `Object` para referenciar objetos de qualquer tipo. Porém, para fazer qualquer coisa específica com o objeto, precisamos saber qual o tipo original do objeto e aplicar um cast:

```
Object obj = new Pessoa("Olga");  
Pessoa p = (Pessoa) obj;
```

A superclasse Object

- **Toda** classe em Java **extends** a classe `java.lang.Object`
 - Não precisa herdar explicitamente
- Como toda classe em Java **é-um Object**, podemos usar uma variável do tipo `Object` para referenciar objetos de qualquer tipo. Porém, para fazer qualquer coisa específica com o objeto, precisamos saber qual o tipo original do objeto e aplicar um cast:

```
Object obj = new Pessoa("Olga");  
Pessoa p = (Pessoa) obj;
```

- Em Java, somente os tipos nativos não são objetos.
- Todos os tipos de array, sejam eles arrays de tipos nativos ou de objetos, são tipos de classes que estendem a classe `Object`.

```
Object obj = new int[10];  
obj = new Pessoa[10];
```

Métodos da classe Object

String toString()

- Este método retorna uma string representando o valor do objeto.
- A implementação default desse método retorna o **nome do pacote** e o **nome da classe** do objeto, seguido de @ e de uma **representação hexadecimal** do valor retornado pelo método **hashCode** do objeto em questão.
- No entanto, como já vimos, podemos sobrescrever este método na nossa subclasse para que ele atenda as necessidades da subclasse.

Métodos da classe Object

Class getClass()

- Todo objeto em Java conhece o seu tipo em tempo de execução.
- O método `getClass()` retorna um objeto da classe `Class` (pacote `java.lang`), que contém informação sobre o tipo do objeto, tal como o nome da classe.
- Dois dos métodos da classe `Class` são:
 - String `getName()`: retorna o nome desta classe.
 - Class `getSuperClass()`: retorna a superclasse desta classe como um objeto da classe `Class`.

Exemplo – Classe Empregado

```
1 import java.time.LocalDate;
2
3 public class Empregado {
4     private String nome;
5     private double salario;
6     private LocalDate dataAdmissao;
7
8     public Empregado(String nome, double salario, int ano, int
        mes, int dia) {
9         this.nome = nome;
10        this.salario = salario;
11        dataAdmissao = LocalDate.of(ano, mes, dia);
12    }
13
14    public String getNome() {
15        return nome;
16    }
17
18    public double getSalario() {
19        return salario;
20    }
```

Exemplo – Classe Empregado

```
1      public LocalDate getDataAdmissao() {
2          return dataAdmissao;
3      }
4
5      public void aumentaSalario(double porcentagem) {
6          double aumento = salario * porcentagem / 100;
7          salario += aumento;
8      }
9
10     @Override
11     public String toString() {
12         String resultado = getClass().getName();
13         resultado += "[nome:" + nome + ", salario:" + salario
14             + ", data de admissão:" + dataAdmissao + "];"
15         return resultado;
16     }
17
18 }
```

Métodos da classe Object

`boolean equals(Object obj)`

- O método `equals` verifica se dois objetos são iguais e retorna `true` se e somente se este for o caso.
- A igualdade que se verifica neste método diz respeito ao **conteúdo** dos objetos sendo comparados (se eles possuem ou não o mesmo **estado**).
- `equals` recebe qualquer `Object` como argumento.
- A implementação *default* do método `equals` na classe `Object` usa o operador de igualdade `==` para determinar se as duas referências apontam para o mesmo objeto na memória.
 - Logo, a **funcionalidade default** deste método geralmente **não é adequada** e, se quisermos chamar este método em objetos da nossa classe, **teremos que sobrescrevê-lo**.

equals() – Exemplo

Vamos considerar dois empregados iguais se eles possuem o mesmo nome, o mesmo salário e a mesma data de admissão.

```
1  @Override
2  public boolean equals(Object obj) {
3      // um teste para ver se os objetos são idênticos
4      if(this == obj) return true;
5
6      // retorna se false se o argumento é nulo
7      if(obj == null) return false;
8
9      // se as classes não casam, não podem ser iguais
10     if(getClass() != obj.getClass()) return false;
11
12     // agora, sabemos que obj é um Empregado não-nulo
13     Empregado p = (Empregado) obj;
14
15     // checa se os estados dos objetos são idênticos
16     return nome.equals(p.nome)
17         && salario == p.salario
18         && dataAdmissao.equals(p.dataAdmissao);
19 }
```

`equals()` – um problema

- A nossa implementação do método `equals` ainda tem um problema: se pelo menos um dos atributos `name` ou `dataAdmissao` for nulo, uma exceção será lançada, pois o método `equals` não pode ser invocado por uma referência nula.
- Neste caso, podemos usar o método estático `Objects.equals(a, b)`.
 - retorna `true` se ambos os argumentos forem nulos;
 - retorna `false` se somente um for nulo;
 - chama `a.equals(b)` caso contrário.
- A classe `Objects` pertence ao pacote `java.util`

equals() – Exemplo

```
1  @Override
2  public boolean equals(Object obj) {
3      // um teste para ver se os objetos são idênticos
4      if(this == obj) return true;
5
6      // retorna se false se o argumento é nulo
7      if(obj == null) return false;
8
9      // se as classes não casam, não podem ser iguais
10     if(getClass() != obj.getClass()) return false;
11
12     // agora, sabemos que obj é um Empregado não-nulo
13     Empregado p = (Empregado) obj;
14
15     // checa se os estados dos objetos são idênticos
16     return Objects.equals(nome, p.nome)
17         && salario == p.salario
18         && Objects.equals(dataAdmissao, p.dataAdmissao);
19 }
```

`equals()` – Superclasses

- Quando você estiver definindo o método `equals` para uma subclasse, primeiro chame o método `equals` da superclasse.
- Se o teste na superclasse não passar, então os objetos não podem ser iguais.
- Se os atributos da superclasse forem iguais, então você estará pronto para comparar os atributos da subclasse.

equals() – Superclasses

- Quando você estiver definindo o método `equals` para uma subclasse, primeiro chame o método `equals` da superclasse.
- Se o teste na superclasse não passar, então os objetos não podem ser iguais.
- Se os atributos da superclasse forem iguais, então você estará pronto para comparar os atributos da subclasse.

```
1  @Override
2  public boolean equals(Object obj) {
3      if(!super.equals(obj)) return false;
4
5      // super.equals checou que obj pertence
6      // à classe Gerente
7      Gerente gerente = (Gerente) obj;
8      return bonus == gerente.bonus;
9  }
```


Métodos da classe Object

`int hashCode()`

- Um **hash code** é um inteiro derivado de um objeto.
 - Se x e y são dois objetos distintos, deve existir uma alta probabilidade de que seus hash codes sejam distintos.
- A implementação *default* do método `hashCode`, na classe Object, deriva o hash code do objeto a partir da sua posição na memória.
 - Logo, pela implementação *default*, dois objetos distintos na memória possuem diferentes hash codes.

hashCode()

- Algumas classes sobrescrevem o método `hashCode` de modo que ele tenha uma lógica diferente.
 - Por exemplo, a classe `String` calcula o hash code **a partir do conteúdo da string**. Logo, dois objetos `String` possuem o mesmo `hashCode` se e somente suas sequências de caracteres são idênticas.
 - Analisar o arquivo `TesteString.java`

hashCode()

- Algumas classes sobrescrevem o método `hashCode` de modo que ele tenha uma lógica diferente.
 - Por exemplo, a classe `String` calcula o hash code a partir do conteúdo da string. Logo, dois objetos `String` possuem o mesmo `hashCode` se e somente suas sequências de caracteres são idênticas.
 - Analisar o arquivo `TesteString.java`
- Porém, outras classes não sobrescrevem o método `hashCode`.
 - É o caso, por exemplo, da classe `StringBuilder`
 - Analisar o arquivo `TesteStringBuilder.java`

O casamento `hashCode()` e `equals()`

- **Obs.:** se você redefinir o método `equals` na sua classe, então você deve também redefinir o método `hashCode`, obedecendo a seguinte regra:
 - Objetos que são iguais de acordo com o método `equals` devem ter o mesmo `hashCode`.

O casamento hashCode() e equals()

- No nosso exemplo, dois objetos x e y não-nulos da classe **Empregado** são iguais se e somente se eles possuem o mesmo estado.
Ou seja, se as 3 condições a seguir são satisfeitas:
 - $x.nome$ é igual a $y.nome$
 - $x.salario$ é igual a $y.salario$
 - $x.dataAdmissao$ é igual a $y.dataAdmissao$
- Portanto, os três atributos da classe **Empregado** devem ser considerados no cálculo do hashCode de um objeto do tipo Empregado.

O casamento hashCode() e equals()

- A classe `Objects` (pacote `java.util`) possui várias versões sobrecarregadas do método estático `hashCode()`, que recebe um argumento e retorna o hash code dele. Esse método retorna 0 se o argumento passado para ele for `null`; caso contrário, ela chama o método `hashCode` para o argumento e retorna o seu valor.

O casamento hashCode() e equals()

- A classe `Objects` (pacote `java.util`) possui várias versões sobrecarregadas do método estático `hashCode()`, que recebe um argumento e retorna o hash code dele. Esse método retorna 0 se o argumento passado para ele for `null`; caso contrário, ela chama o método `hashCode` para o argumento e retorna o seu valor.
- Analisar a classe `Empregado4.java` e `Gerente4.java`

FIM

