spv32 14 марта 2019 в 16:52

# Java Challengers #4: Сравнение объектов с equals() и hashCode()

Автор оригинала: Rafael Chinelato Del Nero

Блог компании OTUS. Онлайн-образование, Java, Программирование

Перевод

## Java Challengers #4: Сравнение объектов с equals() и hashCode()

*В преддверии запуска нового потока по курсу "Разработчик Java" мы продолжаем перевод серии статей Java Challengers, предыдущие част которых можно прочитать по ссылкам ниже:*

- Перегрузка методов в JVM
- Сравнение строк
- Полиформизм и наследование

*Поехали!*

В этой статье вы узнаете, как связаны между собой методы `equals()` и `hashCode()` и как они используются при сравнении объектов.


Без использования `equals()` и `hashCode()` для сравнения состояния двух объектов нам нужно писать много сравнений "`if`", сравнивая каждое поле объекта. Такой подход делает код запутанным и трудным для чтения. Работая вместе, эти два метода помогают создавать более гибкий и согласованный код.

Исходный код для статьи находится здесь.

## Переопределение equals() и hashCode()

*Переопределение метода* (method overriding) — это приём при котором поведение родительского класса или интерфейса переписывается (переопределяется) в подклассе (см. Java Challengers #3: Полиморфизм и наследование, анг.). В Java у каждого объекта есть методы `equals()` `hashCode()` и для правильной работы они должны быть переопределены.

Чтобы понять, как работает переопределение `equals()` и `hashCode()`, изучим их реализацию в базовых классах Java. Ниже приведён метод `equals()` класса `Object`. Метод проверяет, совпадает ли текущий экземпляр с переданным объектом `obj`.

```
public boolean equals(Object obj) {
        return (this == obj);
}
```

Теперь посмотрим на метод `hashCode()` в классе `Object`.

```
@HotSpotIntrinsicCandidate
public native int hashCode();
```

Это native — метод, который написан на другом языке, таком как Си, и он возвращает некоторый числовой код, связанный с адресом памяти объекта. (Если вы не пишете код JDK, то не важно точно знать, как работает этот метод.)
***Примечание переводчика:*** *про значение, связанное с адресом сказано не совсем корректно (спасибо @vladimir_dolzhenko). В HotSpot JVM по умолчанию используются псевдослучайные числа. Описание реализации hashCode() для HotSpot, есть здесь и здесь.*

Если методы `equals()` и `hashCode()` не переопределены, вместо них будут вызваны методы класса `Object`, описанные выше. В этом случае методы не выполняют реальной цели `equals()` и `hashCode()`, которая состоит в том, чтобы проверить, имеют ли объекты одинаковые состояния

Как правило, при переопределении `equals()` также переопределяется `hashCode()`.

## Сравнение объектов с equals()

Метод `equals()` используется для сравнения объектов. Чтобы определить одинаковые объекты или нет, `equals()` сравнивает значения полей объектов:

```
public class EqualsAndHashCodeExample {
  public static void main(String... args){
    System.out.println(new Simpson("Homer", 35, 120)
      .equals(new Simpson("Homer",35,120)));

    System.out.println(new Simpson("Bart", 10, 120)
      .equals(new Simpson("El Barto", 10, 45)));

    System.out.println(new Simpson("Lisa", 54, 60)
      .equals(new Object()));
  }

  static class Simpson {
    private String name;
    private int age;
    private int weight;

    public Simpson(String name, int age, int weight) {
        this.name = name;
        this.age = age;
        this.weight = weight;
    }

    @Override
    public boolean equals(Object o) {
        // 1
        if (this == o) {
            return true;
        }

        // 2
        if (o == null || getClass() != o.getClass()) {
            return false;
        }

        // 3
        Simpson simpson = (Simpson) o;
        return age == simpson.age &&
               weight == simpson.weight &&
               name.equals(simpson.name);
    }
  }
}
```

Посмотрим на метод `equals()`. Первое сравнение сравнивает текущий экземпляр объекта `this` с переданным объектом `o`. Если это один и тот ж объект, то `equals()` вернёт `true`.

Во втором сравнении проверяется, является ли переданный объект `null` и какой у него тип. Если переданный объект другого типа, то объекты н равны.

Наконец, `equals()` сравнивает поля объектов. Если два объекта имеют одинаковые значения полей, то объекты совпадают.

## Анализ вариантов сравнения объектов

Теперь давайте посмотрим на варианты сравнений объектов в методе `main()`. Сначала мы сравниваем два объекта `Simpson`:

```
System.out.println(
    new Simpson("Homer", 35, 120).equals(
    new Simpson("Homer", 35, 120)));
```

У полей этих объектов одинаковые значения, поэтому результат будет `true`.

Затем снова сравниваем два объекта `Simpson`:

```
System.out.println(
    new Simpson("Bart", 10, 45).equals(
    new Simpson("El Barto", 10, 45)));
```

Объекты здесь похожи, но значения имён разные: *Bart* и *El Barto*. Поэтому результат будет `false`.

Наконец, давайте сравним объект `Simpson` и экземпляр класса `Object`:

```
System.out.println(
    new Simpson("Lisa", 54, 60).equals(
    new Object()));
```

В этом случае результат будет `false`, так как типы объектов отличаются.

## equals() в сравнении с ==

На первый взгляд кажется, что оператор `==` и метод `equals()` делают одно и то же, но, на самом деле, они работают по-разному. Оператор `==` сравнивает, указывают ли две ссылки на один и тот же объект. Например:

```
Simpson homer  = new Simpson("Homer", 35, 120);
Simpson homer2 = new Simpson("Homer", 35, 120);

System.out.println(homer == homer2);
```

Мы создали два разных экземпляра `Simpson` с помощью оператора `new`. Поэтому переменные `homer` и `homer2` будут указывать на разные объект в куче. Таким образом, в результате получим `false`.

Во следующем примере используем переопределенный метод `equals()`:

```
System.out.println(homer.equals(homer2));
```

В этом случае будут сравниваться поля. Поскольку значения полей у обоих объектов `Simpson` одинаковые, результат будет `true`.

## Идентификация объектов с hashCode()

Для оптимизации производительности при сравнении объектов используется метод `hashCode()`. Метод `hashCode()` возвращает уникальный идентификатор для каждого объекта, что упрощает сравнение состояний объектов.

Если хэш-код объекта не совпадает с хэш-кодом другого объекта, то можно не выполнять метод `equals()`: вы просто знаете, что два объекта не совпадают. С другой стороны, если хэш-код одинаковый то, необходимо выполнить метод `equals()`, чтобы определить, совпадают ли значения полей.

Рассмотрим практический пример с `hashCode()`.

```
public class HashcodeConcept {

  public static void main(String... args) {
    Simpson homer = new Simpson(1, "Homer");
    Simpson bart  = new Simpson(2, "Homer");

    boolean isHashcodeEquals = homer.hashCode() == bart.hashCode();

    if (isHashcodeEquals) {
      System.out.println("Следует сравнить методом equals.");
    } else {
      System.out.println("Не следует сравнивать методом equals, т.к. " +
        "идентификатор отличается, что означает, что объекты точно не равны.");
    }
  }

  static class Simpson {
    int id;
    String name;

    public Simpson(int id, String name) {
       this.id = id;
       this.name = name;
    }

    @Override
    public boolean equals(Object o) {
      if (this == o) return true;
      if (o == null || getClass() != o.getClass()) return false;
      Simpson simpson = (Simpson) o;
      return id == simpson.id &&
             name.equals(simpson.name);
    }

    @Override
    public int hashCode() {
      return id;
    }
  }
}
```

Метод `hashCode()`, который всегда возвращает одно и то же значение, допустим, но не эффективен. В этом случае сравнение всегда будет возвращать `true`, поэтому метод `equals()` будет выполняться всегда. В этом случае нет никакого улучшения производительности.

## Использование equals() и hashCode() с коллекциями

Классы, реализующие интерфейс `Set` (множество) должны не допускать добавления повторяющихся элементов. Ниже приведены некоторые классы, реализующие интерфейс `Set`:

- HashSet
- TreeSet
- LinkedHashSet
- CopyOnWriteArraySet

В `Set` могут быть добавлены только уникальные элементы. Таким образом, если вы хотите добавить элемент, например, в `HashSet`, вы должны использовать сначала методы `equals()` и `hashCode()`, чтобы убедиться, что этот элемент уникальный. Если методы `equals()` и `hashCode()` не были переопределены, вы рискуете вставить повторяющиеся значения.

Посмотрим на часть реализации метода `add()` в `HashSet`:

```
if (e.hash == hash && ((k = e.key) == key || (key != null && key.equals(k))))
  break;
  p = e;
```

Перед добавлением нового элемента `HashSet` проверяет, существует ли элемент в данной коллекции. Если объект совпадает, то новый элемент вставляться не будет.

Методы `equals()` и `hashCode()` используются не только в `Set`. Также эти методы требуются для HashMap, Hashtable, и LinkedHashMap. Как правило, если вы видите коллекцию с префиксом *"Hash"*, вы можете быть уверены, что для её корректной работы требуется переопределение методов `hashCode()` и `equals()`.

## Рекомендации по использованию equals() и hashCode()

Выполняйте метод `equals()` только для объектов с одинаковым хэш-кодом. *Не выполняйте* `equals()`, если хэш-код отличается.

Таблица 1. Сравнение хэш-кодов

| Если сравнение hashCode() ... | То ... |
|---|---|
| возвращает `true` | выполнить `equals()` |
| возвращает `false` | не выполнять `equals()` |

Этот принцип в основном используется в коллекциях `Set` или `Hash` по соображениям производительности.

## Правила сравнения объектов

Когда сравнение `hashCode()` возвращает `false`, метод `equals()` *также должен возвращать* `false`. Если хэш-код отличается, то объекты определенно не равны.

Таблица 2.Сравнение объектов с hashCode()

| Когда сравнение `hashCode()` возвращает ... | Метод `equals()` должен вернуть ... |
|---|---|
| `true` | `true` или `false` |
| `false` | `false` |

Когда метод `equals()` возвращает `true`, это означает, что объекты равны *во всех значениях и атрибутах*. В этом случае сравнение хэш-кода также должно быть истинным.

Таблица 3. Сравнение объектов с equals()

| Когда метод `equals()` возвращает ... | Метод `hashCode()` должен вернуть ... |
|---|---|
| `true` | `true` |
| `false` | `true` или `false` |

## Решите задачку на equals() и hashCode()

Пришло время проверить ваши знания методов `equals()` и `hashCode()`. Задача состоит в том, чтобы выяснить результат нескольких `equals()` и итоговый размер коллекции `Set`.

Для начала, внимательно изучите следующий код :

```
public class EqualsHashCodeChallenge {

  public static void main(String... args) {
    System.out.println(new Simpson("Bart").equals(new Simpson("Bart")));

    Simpson overriddenHomer = new Simpson("Homer") {
      public int hashCode() {
        return (43 + 777) + 1;
      }
    };
    System.out.println(new Simpson("Homer").equals(overriddenHomer));

    Set set = new HashSet(Set.of(new Simpson("Homer"), new Simpson("Marge")));
    set.add(new Simpson("Homer"));
    set.add(overriddenHomer);
    System.out.println(set.size());
  }

  static class Simpson {
    String name;

    Simpson(String name) {
      this.name = name;
    }

    @Override
    public boolean equals(Object obj) {
      Simpson otherSimpson = (Simpson) obj;
      return this.name.equals(otherSimpson.name) &&
             this.hashCode() == otherSimpson.hashCode();
    }

    @Override
    public int hashCode() {
      return (43 + 777);
    }
  }

}
```

Сначала проанализируйте код, подумайте, какой будет результат. И только потом запустите код. Цель в том, чтобы улучшить ваши навыки анализа кода и усвоить основные концепции Java, чтобы вы могли сделать свой код лучше.

Какой будет результат?.

```
A)
true
true
4

B)
true
false
3

C)
true
false
2
```

```
D)
false
true
3
```

## Что произошло? Понимание equals() и hashCode()

В первом сравнении результат `equals()` равен `true`, поскольку состояния объектов одинаковые, и метод `hashCode()` возвращает одно и то же значнеие для обоих объектов.

Во втором сравнении для переменной `overriddenHomer` был переопределён метод `hashCode()`. Для обоих объектов `Simpson` имя равно *"Homer"*, но для `overriddenHomer` метод `hashCode()` возвращает другое значение. В этом случае результат метода `equals()` будет `false`, так как в нём содержится сравнение с хэш-кодом.

Вы, должно быть, поняли, что в коллекции будет три объекта `Simpson`. Давайте разберём это.

Первый объект в наборе будет вставлен как обычно:

```
new Simpson("Homer"); // Добавляется
```

Следующий объект также будет вставлен в обычном порядке, поскольку содержит значение, отличное от предыдущего объекта:

```
new Simpson("Marge"); // Добавляется
```

Наконец, следующий объект `Simpson` имеет то же значение имени, что и первый объект. В этом случае объект вставляться не будет:

```
set.add(new Simpson("Homer")); // Не добавляется
```

Как мы знаем, объект `overridenHomer` использует другое значение хэш-кода в отличие от обычного экземпляра `Simpson("Homer")`. По этой причине этот элемент будет вставлен в коллекцию:

```
set.add(overriddenHomer); // Добавляется
```

## Ответ

Правильный ответ — B. Вывод будет:

```
true
false
3
```

## Частые ошибки с equals() и hashCode()

- Отсутствие переопределения `hashCode()` вместе с переопределением `equals()` или наоборот.
- Отсутствие переопределения `equals()` и `hashCode()` при использовании хэш-коллекций, таких как `HashSet`.
- Возврат постоянного значения в методе `hashCode()` вместо возврата уникального кода для каждого объекта.
- Равнозначное использование `==` и `equals()`. Оператор `==` сравнивает ссылки на объекты, тогда как метод `equals()` сравнивает значения объектов.

## Что нужно помнить о equals() и hashCode()

- Рекомендуется всегда переопределять методы `equals()` and `hashCode()` в ваших POJO (рус., анг.)
- Используйте эффективный алгоритм для создания уникального хэш-кода.
- При переопределении метода `equals()` всегда переопределяйте метод `hashCode()`.
- Метод `equals()` должен сравнивать полное состояние объектов (значения из полей).
- Метод `hashCode()` может быть идентификатором (ID) POJO.
- Если результат сравнения хэш-кода двух объектов `false`, то метод `equals()` также должен иметь значение `false`.
- Если `equals()` и `hashCode()` не переопределяются при использовании хэш-коллекций, то коллекция будет иметь повторяющиеся элементы.

## Изучите больше о Java

- Посмотрите больше примеров работы с equals() и hashCode() (анг.).
- См. Java Dev Gym и NoBugsProject от Рафаэля Дел Неро (Rafael Del Nero).

*Традиционно жду ваши комментарии и приглашаю на открытый урок, который уже 18 марта проведет наш преподаватель Сергей Петрелевич*

**Теги:** java

25 мая 2017 в 13:33

**Стать востребованным на рынке труда: какие темы нужно знать Java разработчику?**

+1 13,9k 76 19

## Комментарии 7

**sshikov** 14 марта 2019 в 17:30 +1

В половине мест hashcode. А Java case sensitive язык, между прочим…

**spv32** 14 марта 2019 в 17:38 0

Большое спасибо! Вот это фейл. Поправил. PS В оригинале везде hashcode(), а я как-то не обратил внимания.

**sshikov** 14 марта 2019 в 17:40 0

У вас пост на хабре не компилируется… да, я такого тоже не видел :)

**aleksandy** 14 марта 2019 в 20:35 +1

> Метод equals() должен сравнивать полное состояние объектов (значения из полей)

Довольно-таки спорное утверждение. А если у объекта имеется коллекция других объектов, в свою очередь тоже содержащая коллекцию объектов? Я бы сказал, что сравнивать следует минимальное количество полей, необходимое для однозначной идентификации объекта.

Самой простой аналогией является запись в таблице БД: необязательно сравнивать все поля записи, т.к. достаточно сравнить значения первичных ключей.

Также неплохо было бы упомянуть, что хорошей практикой является использование в equals() и hashCode() immutable-полей.

**spv32** 15 марта 2019 в 19:49 +2

Спасибо за дополнение. Действительно, эта тема в статье раскрыта недостаточно.
У объекта можно выделить два типа полей: относящиеся к идентичности объекта и относящиеся к его состоянию.

Идентичность — то, что уникально идентифицирует объект и не меняется в процессе его жизни (например, ID в базе, дата рождения и тд).
Состояние — то, что может изменяться (например, возраст человека, цвет машины и тд).

В equals должны принимать участие *только поля, относящиеся к идентичности объекта*, но не к состоянию. Иначе, например, при изменении возраста у человека — это уже будет несколько разных объектов.

Для хранимых сущностей (которые entity, но не value object), как правило, в equals участвует только id.
Например, AbstractPersistable из spring-data-jpa:

```
return null == this.getId() ? false : this.getId().equals(that.getId());
```

Хотя для hibernate есть немного мудренее рекомендации (раз, два) — использовать бизнес (натуральные) ключи.
И еще несколько ссылок про JPA, hashCode() и equals(): раз, два, три, четыре

**vladimir_dolzhenko** 16 марта 2019 в 23:40 0

Понимаю, что это перевод и в оригинале написана чепуха

```
This is a native method, which means it will be executed in another language like C, and will return some code regarding the object's memory address.
```

которую вы и перевели, но по-умолчанию hotspot (как и многие другие) таки возвращают просто псевдослучайное число — можно было бы и указать в пометках переводчика.

**spv32** 17 марта 2019 в 17:25 +1

Спасибо за очередное полезное замечание. Добавил в статью.

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки | Неделя | Месяц

У телефона Samsung со сгибаемым экраном за $2000 на дисплее образуется складка

+23 39,7k 15 142

«Пора валить из фронтенда»: Андрей Ситник о стагнации сообщества, опенсорсе и не только

+61 44,5k 132 150

Как я не стал специалистом по машинному обучению

+68 21,5k 98 82

Любительская голография — начало пути

+118 16,3k 119 89

В гости к отцу

+48 143k 66 61