

Лекция 6.5: Коллекции / HashMap/HashSet, hashCode().

1. Intro

....1.1 Первая попытка (нет hashCode)
....1.2 Вторая попытка (hashCode == 0)
....1.3 Третья попытка (hashCode ==)
....1.4 Сравнение второй и третьей попыток

2. HashSet/HashMap internals

....2.1 ???
....2.2 ???
....2.3 ???

3. Стандартная реализация hashCode

....3.1 ???
....3.2 ???
....3.3 ???

4. Математические требования на реализацию hashCode

....4.1 ???
....4.2 ???
....4.3 ???

LinkedHashSet/LinkedHashMap
System.IdentityHashCode()
WeakHashMap and IdentityHashMap
EnumMap,
ConcurrentHashMap
Hashtable

5. Больше информации о hash-структурах

...5.1 Стандартная реализация Object.hashCode()
...5.2 HashSet over HashMap
...5.3 chaining vs open address

Тесты по всей лекции
Видео
Лабораторные

....???
....???
....???
Литература

HashSet/HashMap

hashCode()

Тест

Для прохождения теста по теме Coll.HashCodeEquals (уровень сложности теста: Basic) нажмите "Start Quiz"

Тест

Для прохождения теста по теме Coll.Equals.Algebra (уровень сложности теста: Hard) нажмите "Start Quiz"

Первая попытка (нет hashCode)

Возьмем вот такую реализацию класса Человек(имя, возраст)

```
1 import java.util.Objects;
2
3 public class PersonA {
4     public final String name;
5     public final int age;
6
7     public PersonA(String name, int age) {
8         this.name = name;
9         this.age = age;
10    }
11
12    @Override
13    public boolean equals(Object obj) {
14        if (obj == null || this.getClass() != obj.getClass()) {
15            return false;
16        }
17        PersonA that = (PersonA) obj;
18
19        // 'age' comparing
20        if (age != that.age) {
21            return false;
22        }
23        // 'name' comparing
24        return Objects.equals(this.name, that.name);
25    }
26
27    @Override
28    public String toString() {
29        return "PersonA[name=" + name + '\'' + ", age=" + age + '\'';
30    }
31 }
```

Посмотрим, как она ведет себя с List-ом (ХОРОШО ведет)

```
1 import java.util.ArrayList;
2 import java.util.Collection;
3
4 public class App {
5     public static void main(String[] args) {
6         Collection<PersonA> coll = new ArrayList<>();
7         coll.add(new PersonA("Mike", 45));
8         System.out.println("coll = " + coll);
9         System.out.println("contains: " + coll.contains(new PersonA("Mike", 45)));
10        System.out.println("remove: " + coll.remove(new PersonA("Mike", 45)));
11        System.out.println("coll = " + coll);
12        System.out.println("contains: " + coll.contains(new PersonA("Mike", 45)));
13        System.out.println("remove: " + coll.remove(new PersonA("Mike", 45)));
14    }
15 }
16
17 >> coll = [PersonA[name='Mike', age=45]]
18 >> contains: true
19 >> remove: true
20 >> coll = []
21 >> contains: false
22 >> remove: false
```

Посмотрим, как она ведет себя с HashSet-ом (ПЛОХО ведет: не обнаруживает в коллекции и не удаляет)

```
1 import java.util.Collection;
2 import java.util.HashSet;
3
4 public class App {
5     public static void main(String[] args) {
6         Collection<PersonA> coll = new HashSet<>();
7         coll.add(new PersonA("Mike", 45));
8         System.out.println("coll = " + coll);
9         System.out.println("contains: " + coll.contains(new PersonA("Mike", 45)));
10        System.out.println("remove: " + coll.remove(new PersonA("Mike", 45)));
11        System.out.println("coll = " + coll);
12    }
13 }
14
15 >> coll = [PersonA[name='Mike', age=45]]
16 >> contains: false
17 >> remove: false
18 >> coll = [PersonA[name='Mike', age=45]]
```

А вот для стандартного класса java.lang.String – все отлично

```
1 import java.util.Collection;
2 import java.util.HashSet;
3
4 public class App {
5     public static void main(String[] args) {
6         Collection<String> coll = new HashSet<>();
7         coll.add("Mike-45");
8         System.out.println("coll = " + coll);
9         System.out.println("contains: " + coll.contains("Mike-45"));
10        System.out.println("remove: " + coll.remove("Mike-45"));
11        System.out.println("coll = " + coll);
12        System.out.println("contains: " + coll.contains("Mike-45"));
13        System.out.println("remove: " + coll.remove("Mike-45"));
14    }
15 }
16
17 >> coll = [Mike-45]
18 >> contains: true
19 >> remove: true
20 >> coll = []
21 >> contains: false
22 >> remove: false
```

И для java.lang.String не допускает дубликатов

```
1 import java.util.Collection;
2 import java.util.HashSet;
3
4 public class App {
5     public static void main(String[] args) {
6         Collection<String> coll = new HashSet<>();
7         System.out.println(coll.add("Mike-45"));
8         System.out.println(coll.add("Mike-45"));
9         System.out.println("coll = " + coll);
10    }
11 }
12
13 >> true
14 >> true
15 >> coll = [Mike-45]
```

Для нашего же класса – допускает дубликаты

```
1 import java.util.Collection;
2 import java.util.HashSet;
3
4 public class App {
5     public static void main(String[] args) {
6         Collection<PersonA> coll = new HashSet<>();
7         System.out.println(coll.add(new PersonA("Mike", 45)));
8         System.out.println(coll.add(new PersonA("Mike", 45)));
9         System.out.println("coll = " + coll);
10    }
11 }
12
13 >> true
14 >> true
15 >> coll = [PersonA[name='Mike', age=45], PersonA[name='Mike', age=45]]
```

Итак, наш класс каким-то образом не приспособлен для работы с HashSet, однако отлично справляется с ArrayList. Разберемся в чем дело.

Вторая попытка (hashCode == 0)

На удивление, нас спасает тривиальнейшая реализация метода hashCode

```
1 public class PersonB {
2     ...
3     public int hashCode() {
4         return 0;
5     }
6     ...
7 }
8
9 import java.util.Objects;
10
11 public class PersonB {
12     public final String name;
13     public final int age;
14
15     public PersonB(String name, int age) {
16         this.name = name;
17         this.age = age;
18     }
19
20     @Override
21     public int hashCode() {
22         return 0;
23     }
24
25     @Override
26     public boolean equals(Object obj) {
27         if (obj == null || this.getClass() != obj.getClass()) {
28             return false;
29         }
30         PersonB that = (PersonB) obj;
31
32         // 'age' comparing
33         if (age != that.age) {
34             return false;
35         }
36         // 'name' comparing
37         return Objects.equals(this.name, that.name);
38     }
39
40     @Override
41     public String toString() {
42         return "PersonB[name=" + name + '\'' + ", age=" + age + '\'';
43     }
44 }
```

Наслаждаемся эффектом (нет дубликатов, работают contains и remove)

```
1 import java.util.Collection;
2 import java.util.HashSet;
3
4 public class App {
5     public static void main(String[] args) {
6         Collection<PersonB> coll = new HashSet<>();
7
8         System.out.println("add = " + coll.add(new PersonB("Mike", 45)));
9         System.out.println("add = " + coll.add(new PersonB("Mike", 45)));
10
11        System.out.println("coll = " + coll);
12        System.out.println("contains: " + coll.contains(new PersonB("Mike", 45)));
13        System.out.println("remove: " + coll.remove(new PersonB("Mike", 45)));
14        System.out.println("coll = " + coll);
15        System.out.println("contains: " + coll.contains(new PersonB("Mike", 45)));
16        System.out.println("remove: " + coll.remove(new PersonB("Mike", 45)));
17    }
18 }
19
20 >> add = true
21 >> add = false
22 >> coll = [PersonB[name='Mike', age=45]]
23 >> contains: true
24 >> remove: true
25 >> coll = []
26 >> contains: false
27 >> remove: false
```

Третья попытка (hashCode == "calculate")

Однако, лучше бы, реализовать hashCode вот так

```
1 public class PersonC {
2     ...
3     @Override
4     public int hashCode() {
5         return 31 * age + ((name == null) ? 0 : name.hashCode());
6     }
7     ...
8 }
```

или вот так (что тоже самое)

```
1 import java.util.Objects;
2
3 public class PersonC {
4
5     @Override
6     public int hashCode() {
7         return 31 * age + Objects.hash(name);
8     }
9     ...
10 }
```

Вот полная реализация третьей версии

```
1 import java.util.Objects;
2
3 public class PersonC {
4     public final String name;
5     public final int age;
6
7     public PersonC(String name, int age) {
8         this.name = name;
9         this.age = age;
10    }
11
12    @Override
13    public int hashCode() {
14        return 31 * age + Objects.hash(name);
15    }
16
17    @Override
18    public boolean equals(Object obj) {
19        if (obj == null || this.getClass() != obj.getClass()) {
20            return false;
21        }
22        PersonC that = (PersonC) obj;
23
24        // 'age' comparing
25        if (age != that.age) {
26            return false;
27        }
28        // 'name' comparing
29        return Objects.equals(this.name, that.name);
30    }
31
32    @Override
33    public String toString() {
34        return "PersonB[name=" + name + '\'' + ", age=" + age + '\'';
35    }
36 }
```

Сравнение второй и третьей попыток

Так почему же третья версия ЛУЧШЕ второй?
Обе – корректны, но третья ПОЗВОЛЯЕТ HashSet РАБОТАТЬ НАМНОГО БЫСТРЕЕ.

Скорость реализации

```
1 import java.util.Collection;
2 import java.util.HashSet;
3
4 public class App {
5     public static void main(String[] args) {
6         Collection<PersonB> coll = new HashSet<>();
7         long t0 = System.nanoTime();
8         for (int k = 0; k < 30_000; k++) {
9             // add
10            coll.add(new PersonB("", k));
11        }
12        long t1 = System.nanoTime();
13        System.out.println("dT(add)      = " + (t1 - t0));
14
15        long t2 = System.nanoTime();
16        // contains
17        coll.contains(new PersonC("", -1));
18        long t3 = System.nanoTime();
19        System.out.println("dT(contains) = " + (t3 - t2));
20    }
21 }
22
23 >> dT(add)      = 1 718 008 072
24 >> dT(contains) = 2 153 005
```

Скорость третьей реализации

```
1 import java.util.Collection;
2 import java.util.HashSet;
3
4 public class App {
5     public static void main(String[] args) {
6         Collection<PersonC> coll = new HashSet<>();
7         long t0 = System.nanoTime();
8         for (int k = 0; k < 30_000; k++) {
9             // add
10            coll.add(new PersonC("", k));
11        }
12        long t1 = System.nanoTime();
13        System.out.println("dT(add)      = " + (t1 - t0));
14
15        long t2 = System.nanoTime();
16        // contains
17        coll.contains(new PersonC("", -1));
18        long t3 = System.nanoTime();
19        System.out.println("dT(contains) = " + (t3 - t2));
20    }
21 }
22
23 >> dT(add)      = 20 465 537
24 >> dT(contains) = 973
```

Как видим на добавлении (add) выигрыш в 80 раз, на проверке вхождения (contains) – более чем в 200 раз.

HashSet/HashMap internals

Стандартная реализация hashCode

Математические требования на реализацию hashCode

Больше информации о hash-структурах

Тесты по всей лекции

Тест, состоящий из случайных вопросов тестов этой лекции

Some issues occurred with quiz randomizer :(

Видео

Набор декабрь 2013
Набор октябрь 2013
Набор июль 2013
Набор апрель 2013
Набор февраль 2013
Набор январь 2013
Набор октябрь 2012

Лабораторные

???

Литература

Хорстманн, Корнели. "Java 2. Том II. Тонкости программирования". Издание 7
....Глава 2. Наборы данных, стр 140-143

© 2014 Golovach Courses |