

Евгений Гродно  
36 уровень

30.04.2018   161026   71

## JUnit part I

Статья из группы Random  
1689423 участника

Вы в группе

### JUnit :: или как полюбить валидатор JavaRush



Кратко о том, зачем этот зверь нам нужен? **JUnit** — это фреймворк автоматического тестирования вашего хорошего **или не совсем хорошего** кода. Можно сказать: — зачем мне эти качели, я и так смогу легко, и просто протестировать свой хороший Java код. Можно много писать вступительной лирики, но поэт из меня никакой, перейдём лучше к делу...

### Создаем объект

И так, чтобы что-то тестировать, нам для начала нужен объект тестирования. Перед нами стоит задача.

1. Нам нужен объект, который будет хранить информацию о Пользователе.
  - a. Id — нужно считать по порядку добавления нового пользователя.
  - b. Имя пользователя.
  - c. Его возраст.
  - d. Пол (male/female)
2. Нужно предусмотреть хранение списка пользователей.
3. Класс должен уметь.

- с. Возвращать количество пользователей в общем списке, и посчитать количество по признаку пола пользователя.
- d. Посчитать общую сумму по возрасту пользователей, так же учесть по признаку пола.
- е. Посчитать средний возраст, как общий так и по признаку пола.

И так, приступим к созданию объекта...

Создадим Java класс `User` он будет содержать поля:

```
1 private int id;
2 private String name;
3 private int age;
4 private Sex sex;
```

Для хранения данных о пользователе этого достаточно, посмотрим что там еще нужно по задаче.

Нам нужно как-то хранить всех пользователей, сделаем в нашем классе статическое поле `allUsers`, думаю нормально если это будет `Map<Integer, User>`

```
1 private static Map<Integer, User> allUsers;
```

Еще нам как-то нужно присваивать порядковый номер пользователям, создадим статическое поле счетчик, который при создании нового пользователя, будет присваивать порядковый `Id` пользователю.

```
1 private static int countId = 0;
```

Так, с полями вроде разобрались, напишем конструктор для нашего объекта, и гетеры для полей `id`, `name`, `age`, `sex`. С гетерами там ничего сложного нет, попросим помощи у **IDEA**, она никогда не откажет, а конструктор сделаем немного с хитростью.

Конструктор будет уметь. Инициализировать поля, проверять есть ли такой объект в `allUsers`, если такого объекта нет, то увеличиваем наш счетчик `countId++`, и добавляем его в список всех пользователей. А так же инициализировать поле `allUsers` ели оно еще не было инициализировано.

Для удобства поиска одинаковых объектов, переопределим методы `equals()` и `hashCode()`, опять попросим помощи у любимой **IDEA** и будем сравнивать по полям `name`, `age`, `sex`. Плюс создадим приватный метод `hasUser()`, который будет проверять есть ли такой объект в списке.

```
1 @Override
2 public boolean equals(Object o) {
3     if (this == o) return true;
4     if (o == null || getClass() != o.getClass()) return false;
5     User user = (User) o;
6     return age == user.age &&
7         Objects.equals(name, user.name) &&
8         sex == user.sex;
9 }
10
11 @Override
12 public int hashCode() {
13
14     return Objects.hash(name, age, sex);
```

Конструктор в итоге у меня получился такой.

```
1  public User(String name, int age, Sex sex) {
2      if (allUsers == null){
3          allUsers = new HashMap<>();
4      }
5
6      this.name = name;
7      this.age = age;
8      this.sex = sex;
9
10     if (!hasUser()){
11         countId++;
12         this.id = countId;
13         allUsers.put(id, this);
14     }
15 }
```

и вспомогательный приватный метод

```
1  private boolean hasUser(){
2      for (User user : allUsers.values()){
3          if (user.equals(this) && user.hashCode() == this.hashCode()){
4              return true;
5          }
6      }
7      return false;
8  }
```

а также переопределим `toString()`

```
1  @Override
2  public String toString() {
3      return "User{" +
4          "id=" + id +
5          ", name='" + name + '\'' +
6          ", age=" + age +
7          ", sex=" + sex +
8          '}';
9  }
```

Теперь пришло время, реализовать логику требуемых методов. Так как логика в основном будет работать со статическими полями, методы тоже сделаем статическими, для объектов они не нужны.

- а. Формировать список всех пользователей.
- б. Формировать список пользователей по полу(MALE/FEMALE).

С пунктами **a** и **b** хорошо справится метод `getAllUsers()` который будет возвращать лист всех `User`, и перегруженный метод `getAllUsers(Sex sex)` он будет возвращать список, в зависимости от переданного параметра `Sex`.

```
1  public static List<User> getAllUsers(){
```

```
4
5    public static List<User> getAllUsers(Sex sex){
6        List<User> listAllUsers = new ArrayList<>();
7        for (User user : allUsers.values()){
8            if (user.sex == sex){
9                listAllUsers.add(user);
10        }
11    }
12    return listAllUsers;
13 }
```

с. Возвращать количество пользователей в общем списке, и посчитать количество по признаку пола пользователя.

```
1    public static int getHowManyUsers(){
2        return allUsers.size();
3    }
4
5    public static int getHowManyUsers(Sex sex){
6        return getAllUsers(sex).size();
7    }
```

д. Посчитать общую сумму по возрасту пользователей, так же учесть по признаку пола. Для этой задачи сделаем методы.

```
1    public static int getAllAgeUsers(){
2        int countAge = 0;
3        for (User user : allUsers.values()){
4            countAge += user.age;
5        }
6        return countAge;
7    }
8
9    public static int getAllAgeUsers(Sex sex){
10        int countAge = 0;
11        for (User user : getAllUsers(sex)){
12            countAge += user.age;
13        }
14        return countAge;
15    }
```

е. Посчитать средний возраст, как общий так и по признаку пола.

```
1    public static int getAverageAgeOfAllUsers(){
2        return getAllAgeUsers() / getHowManyUsers();
3    }
4
5    public static int getAverageAgeOfAllUsers(Sex sex){
6        return getAllAgeUsers(sex) / getHowManyUsers(sex);
7    }
```

Отлично, требуемый объект и его поведение мы описали. Теперь можно переходить к **JUnit**, но для начала покажу как примерно будет выглядеть простой тест если мы его будет делать в **main**.

```
1 public static void main(String[] args) {
2     new User("Евгений", 35, Sex.MALE);
3     new User("Марина", 34, Sex.FEMALE);
4     new User("Алина", 7, Sex.FEMALE);
5
6
7     System.out.println("Все пользователи:");
8     User.getAllUsers().forEach(System.out::println);
9     System.out.println("Все пользователи: MALE");
10    User.getAllUsers(Sex.MALE).forEach(System.out::println);
11    System.out.println("Все пользователи: FEMALE");
12    User.getAllUsers(Sex.FEMALE).forEach(System.out::println);
13    System.out.println("=====");
14    System.out.println("    всех пользователей: " + User.getHowManyUsers());
15    System.out.println("  всех пользователей MALE: " + User.getHowManyUsers(Sex.MALE));
16    System.out.println("всех пользователей FEMALE: " + User.getHowManyUsers(Sex.FEMALE));
17    System.out.println("=====");
18    System.out.println("    общий возраст всех пользователей: " + User.getAllAgeUsers());
19    System.out.println("  общий возраст всех пользователей MALE: " + User.getAllAgeUsers(Sex.MALE));
20    System.out.println("общий возраст всех пользователей FEMALE: " + User.getAllAgeUsers(Sex.FEMALE));
21    System.out.println("=====");
22    System.out.println("    средний возраст всех пользователей: " + User.getAverageAgeOfAllUsers());
23    System.out.println("  средний возраст всех пользователей MALE: " + User.getAverageAgeOfAllUsers(Sex.MALE));
24    System.out.println("средний возраст всех пользователей FEMALE: " + User.getAverageAgeOfAllUsers(Sex.FEMALE));
25    System.out.println("=====");
26 }
```

Вывод в консоль получим примерно такой, а дальше сравниваем получили ли мы нормальную работу. Можно конечно углубиться, написать логику сравнения, и посмотреть, что скажет наше вычисление, при том что мы не будем уверены, что все смогли предусмотреть.

```
1 //output
2 Все пользователи:
3 User{id=1, name='Евгений', age=35, sex=MALE}
4 User{id=2, name='Марина', age=34, sex=FEMALE}
5 User{id=3, name='Алина', age=7, sex=FEMALE}
6 Все пользователи: MALE
7 User{id=1, name='Евгений', age=35, sex=MALE}
8 Все пользователи: FEMALE
9 User{id=2, name='Марина', age=34, sex=FEMALE}
10 User{id=3, name='Алина', age=7, sex=FEMALE}
11 =====
12     всех пользователей: 3
13     всех пользователей MALE: 1
14     всех пользователей FEMALE: 2
15 =====
16     общий возраст всех пользователей: 76
17     общий возраст всех пользователей MALE: 35
18     общий возраст всех пользователей FEMALE: 41
19 =====
20     средний возраст всех пользователей: 25
21     средний возраст всех пользователей MALE: 35
22     средний возраст всех пользователей FEMALE: 20
23 
```

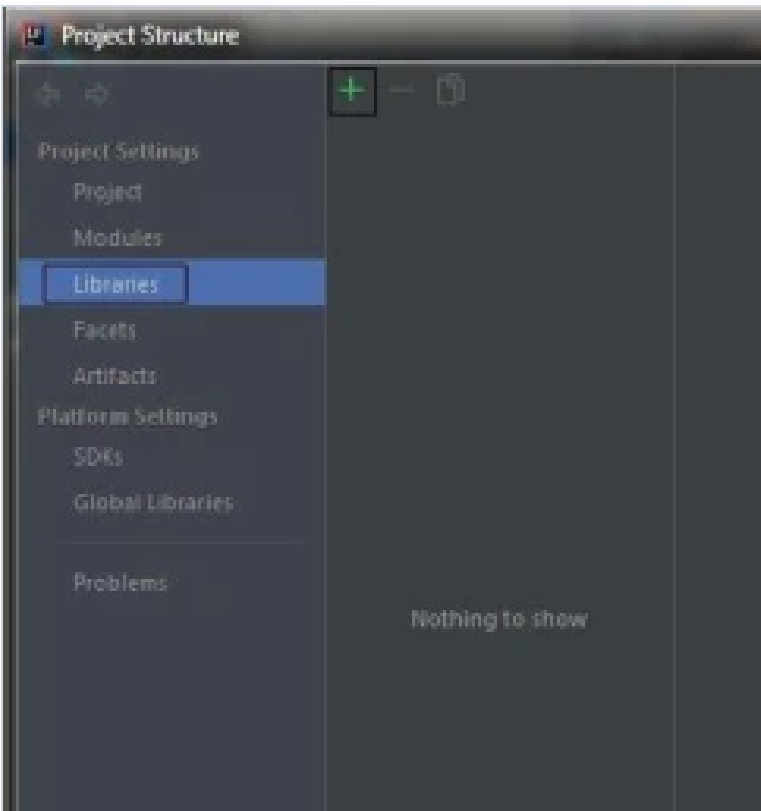
25	Process finished <b>with</b> exit code 0
----	--

Нас этот исход не устраивает, долой тесты **main**, нам нужен **JUnit**.

## Как подключить JUnit к проекту

Возникает вопрос, как его подключить к проекту. Для знающих вариант с **Maven** брать не буду, так как это совсем другая история. ;)

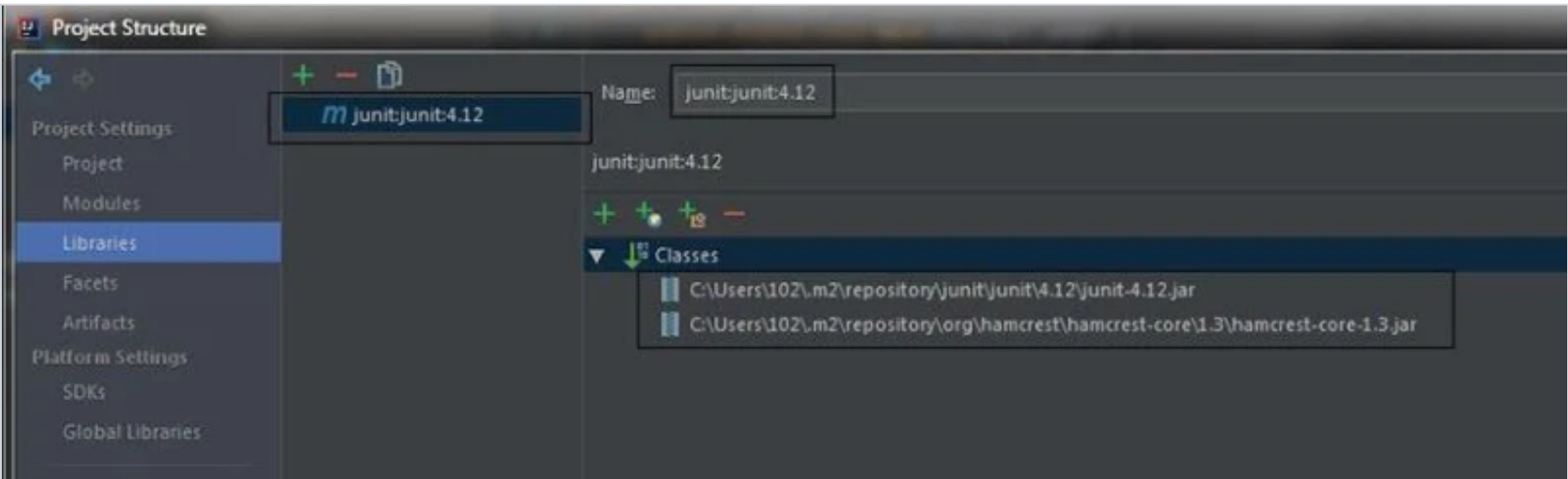
Открываем структуру проекта Ctrl + Alt + Shift + S -> Libraries -> жмем + (New Project Library) -> выбираем from Maven



далее видим такое окно, в строку поиска вводим “ junit:junit:4.12 ” ждем пока найдет -> OK! -> OK!



Должен получиться такой результат

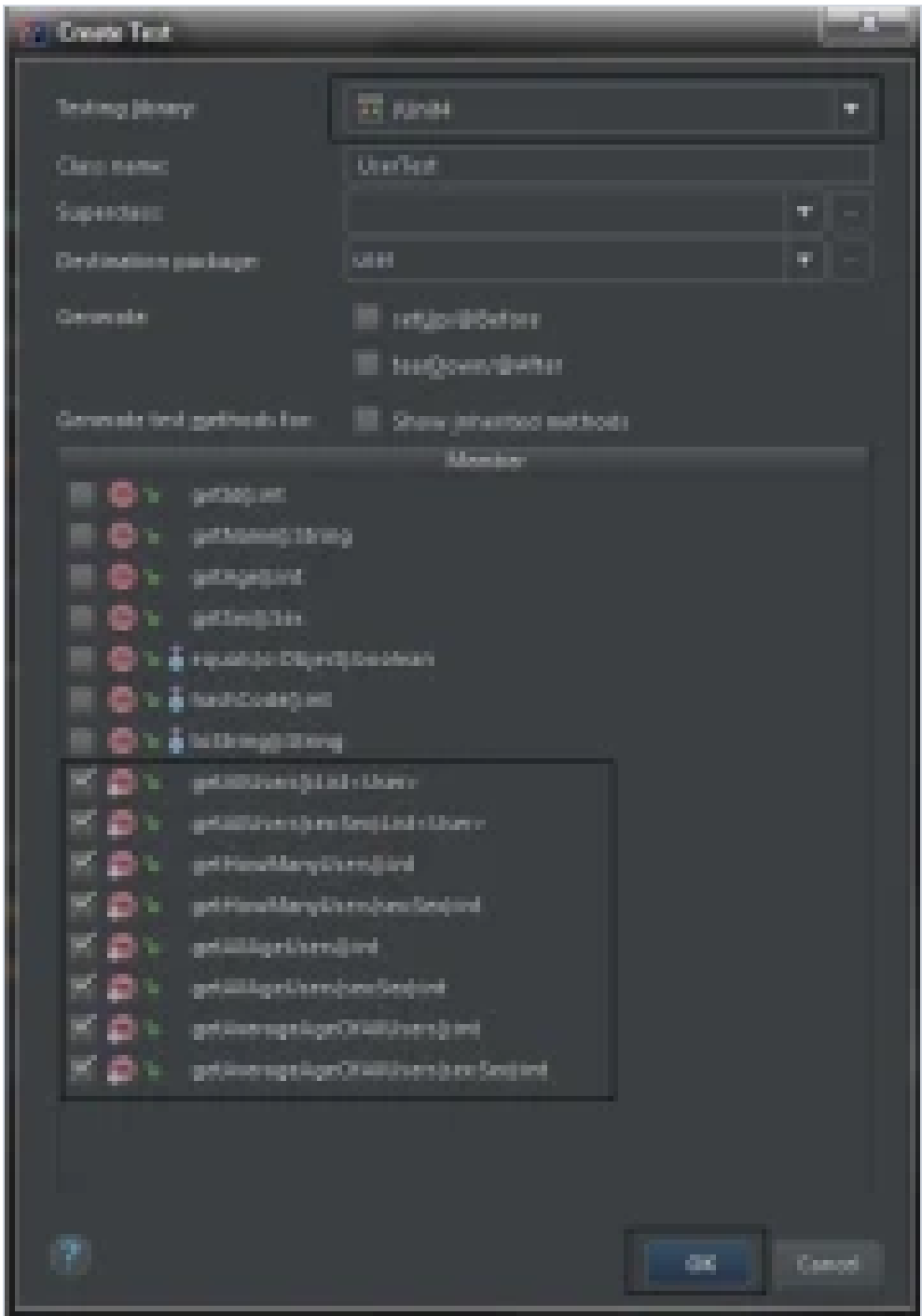




Жмем OK, поздравлю JUnit добавлен к проекту. Едем дальше.

Теперь нам нужно создать тесты для нашего Java класса, ставим курсор на название класса `User` -> жмем Alt + Enter -> выбираем create Test.

Мы должны увидеть окно, в котором нам нужно выбрать библиотеку JUnit4 -> выбрать методы которые собираемся тестировать -> OK



Идея сама создаст класс `UserTest`, это и есть класс, в котором мы будем покрывать наш код тестами. Приступим:

## Наш первый @Test

Создадим наш первый **@Test** метода `getAllUsers()` – это метод который должен вернуть всех пользователей. Тест будет выглядеть примерно так:

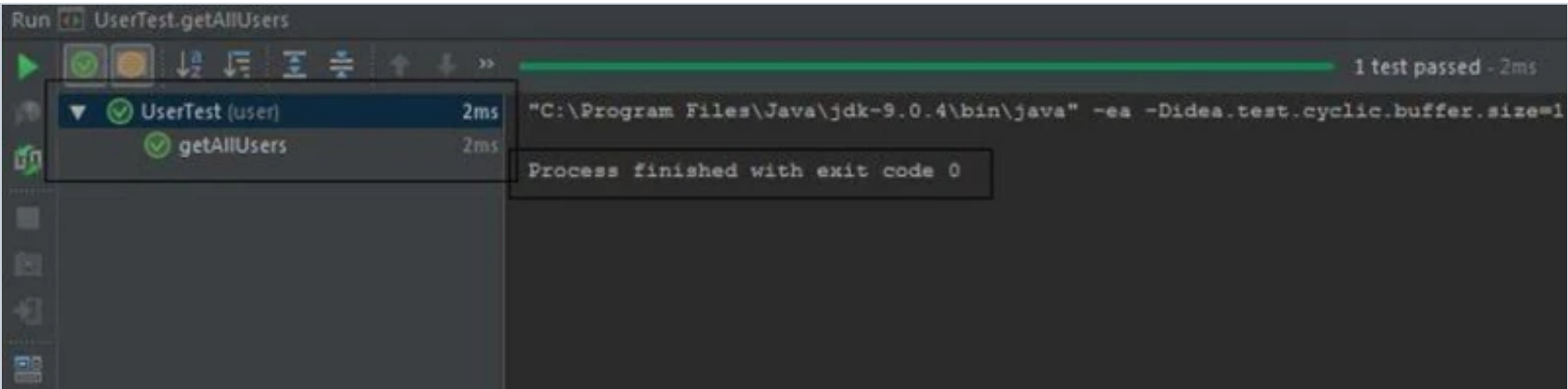
```
1  @Test
2  public void getAllUsers() {
3      //создаем тестовые данные
4      User user = new User("Евгений", 35, Sex.MALE);
5      User user1 = new User("Марина", 34, Sex.FEMALE);
6      User user2 = new User("Алина", 7, Sex.FEMALE);
7
8      //создаем список expected и заполняем его данными нашего метода
9      List<User> expected = User.getAllUsers();
10
11     //создаем список actual в него помещаем данные для сравнения
12     //то что мы предполагаем метод должен вернуть
13     List<User> actual = new ArrayList<>();
14     actual.add(user);
```

НАЧАТЬ ОБУЧЕНИЕ

```
16      actual.add(user2);
17
18      //запускаем тест, в случае если список expected и actual не будут равны
19      //тест будет провален, о результатах теста читаем в консоли
20      Assert.assertEquals(expected, actual);
21  }
```

Тут мы создаем несколько тестовых пользователей -> создаем список `expected` в который поместим пользователей которых нам вернет метод `getAllUsers()` -> создадим список `actual` в который поместим пользователей которых мы предполагаем что метод `getAllUsers()` `Assert.assertEquals(actual, expected)` ему мы и передадим списки, инспектируемый и актуальный. Этот метод проверит объекты в предоставленных списках и выдаст результат теста. Метод будет сравнивать все поля объектов, даже пройдетя по полям родителей, если есть наследование.

Запускаем первый тест...

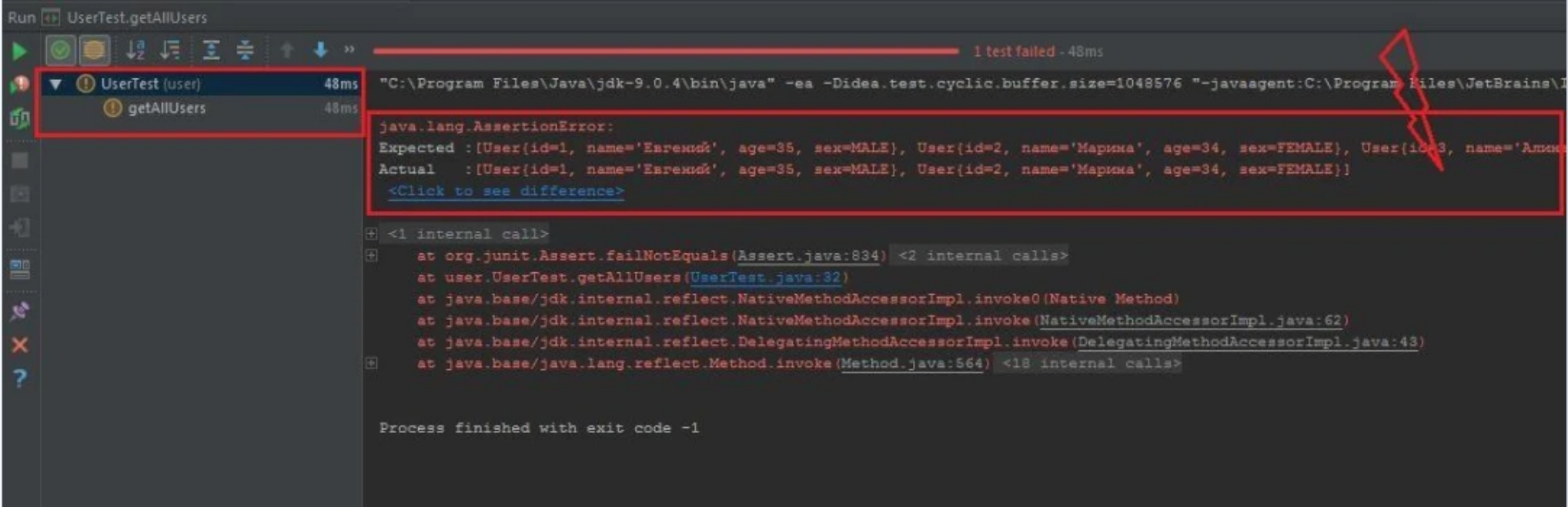


Тест выполнен успешно. Теперь попробуем сделать так, чтобы тест был провален, для этого нам нужно изменить один из списков теста, сделаем это путем, закомментировав добавление одного пользователя в список `actual`,

```
1  @Test
2  public void getAllUsers() {
3      //создаем тестовые данные
4      User user = new User("Евгений", 35, Sex.MALE);
5      User user1 = new User("Марина", 34, Sex.FEMALE);
6      User user2 = new User("Алина", 7, Sex.FEMALE);
7
8      //создаем список expected и заполняем его данными нашего метода
9      List<User> expected = User.getAllUsers();
10
11     //создаем список actual в него помещаем данные для сравнения
12     //то что мы предполагаем метод должен вернуть
13     List<User> actual = new ArrayList<>();
14     actual.add(user);
15     actual.add(user1);
16     //actual.add(user2);
17
18     //запускаем тест, в случае если список expected и actual не будут равны
19     //тест будет провален, о результатах теста читаем в консоли
20     Assert.assertEquals(expected, actual);
21 }
```

запускаем тест и видим следующее:

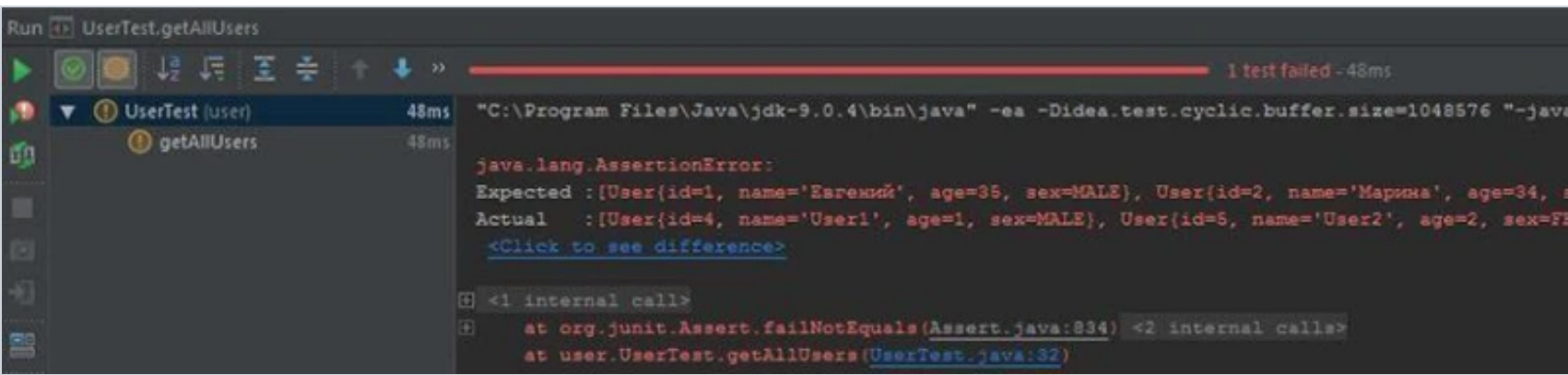




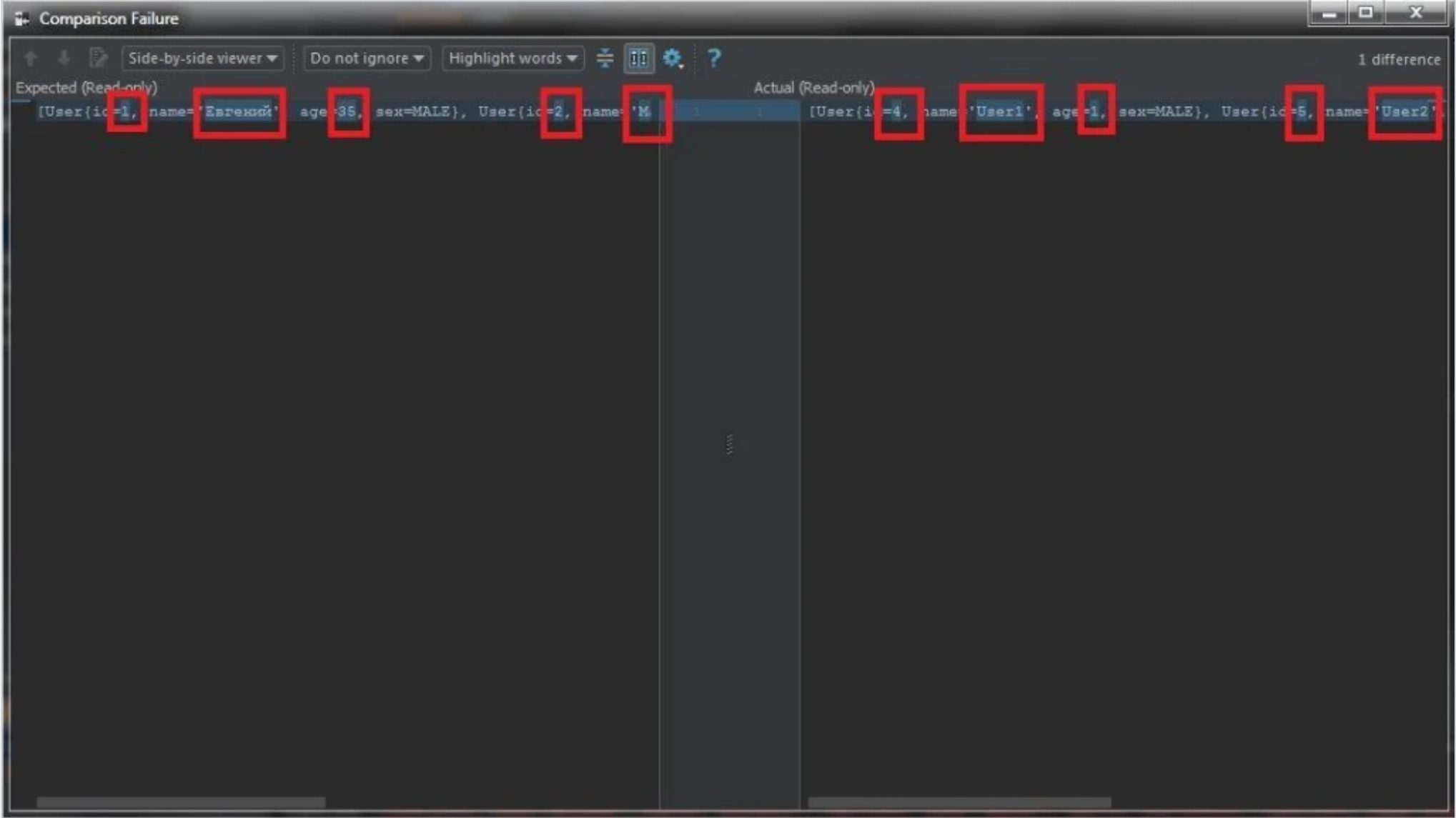
Теперь мы можем немного разобрать причину провала теста. Тут мы видим, что в инспектируемом списке больше пользователей чем в актуальном. Это и есть причина провала. А в main мы можем проверить так? JUnit : main = 1 : 0. Давайте посмотрим как будет выглядеть тест, если в нем будут полностью разные объекты, сделаем это так:

```
1  @Test
2  public void getAllUsers() {
3      //создаем тестовые данные
4      User user = new User("Евгений", 35, Sex.MALE);
5      User user1 = new User("Марина", 34, Sex.FEMALE);
6      User user2 = new User("Алина", 7, Sex.FEMALE);
7
8      //создаем список expected и заполняем его данными нашего метода
9      List<User> expected = User.getAllUsers();
10
11     //создаем список actual в него помещаем данные для сравнения
12     //то что мы предполагаем метод должен вернуть
13     List<User> actual = new ArrayList<>();
14     actual.add(new User("User1", 1, Sex.MALE));
15     actual.add(new User("User2", 2, Sex.FEMALE));
16     actual.add(new User("User3", 3, Sex.MALE));
17
18     //запускаем тест, в случае если список expected и actual не будут равны
19     //тест будет провален, о результатах теста читаем в консоли
20     Assert.assertEquals(expected, actual);
21 }
```

вот что будет в консоли:



тут сразу видно что в сравниваемых списках разные пользователи, еще мы можем кликнуть на <Click to see difference> получим такое окно, где можно посмотреть подробно с какими данными у нас проблема. **IDEA** подсветит все поля в которых есть различия.

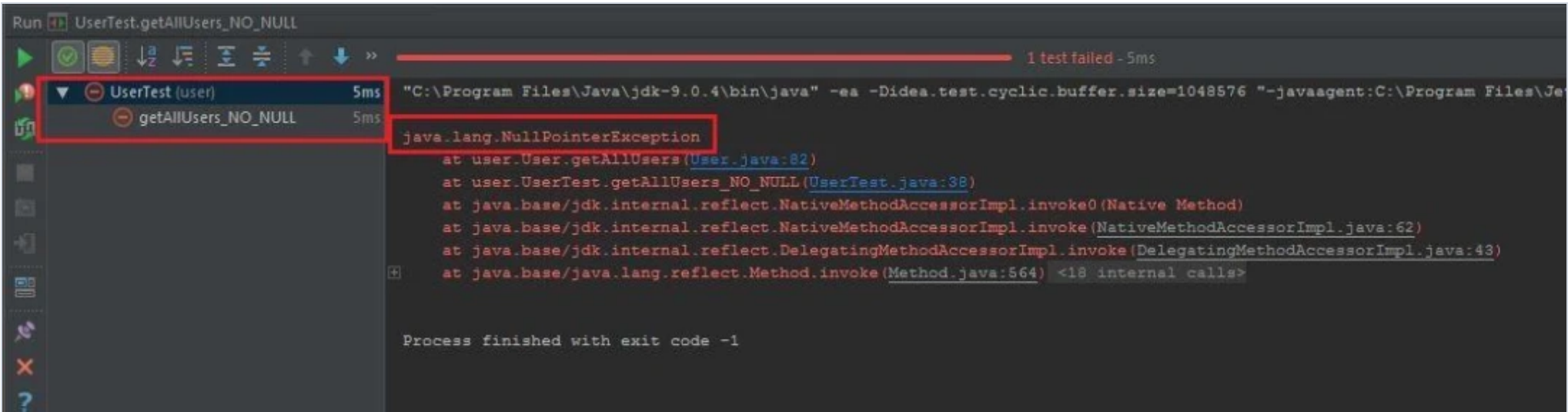


main такое может? — нет. JUnit : main = 2 : 0

Ну что, пойдём дальше у нас ещё куча методов, которые нужно покрыть тестами ), но подождите, а ведь будет не плохо, проверить, а не будет ли нам метод `getAllUsers()` возвращать `null`, ведь примерно так нас на задачах **JavaRush** ловит валидатор ). Сделаем это, делов то на три копейки ...

```
1  @Test
2  public void getAllUsers_NO_NULL() {
3      //добавим проверку на null
4      List<User> expected = User.getAllUsers();
5      Assert.assertNotNull(expected);
6  }
```

Да, да примерно так валидатор ловит наш говно код на `null` ;) Теперь запустим этот тест, и посмотрим, что он нам покажет. А покажет он ошибку, как ???? как же тут можно было допустить ошибку теста)))

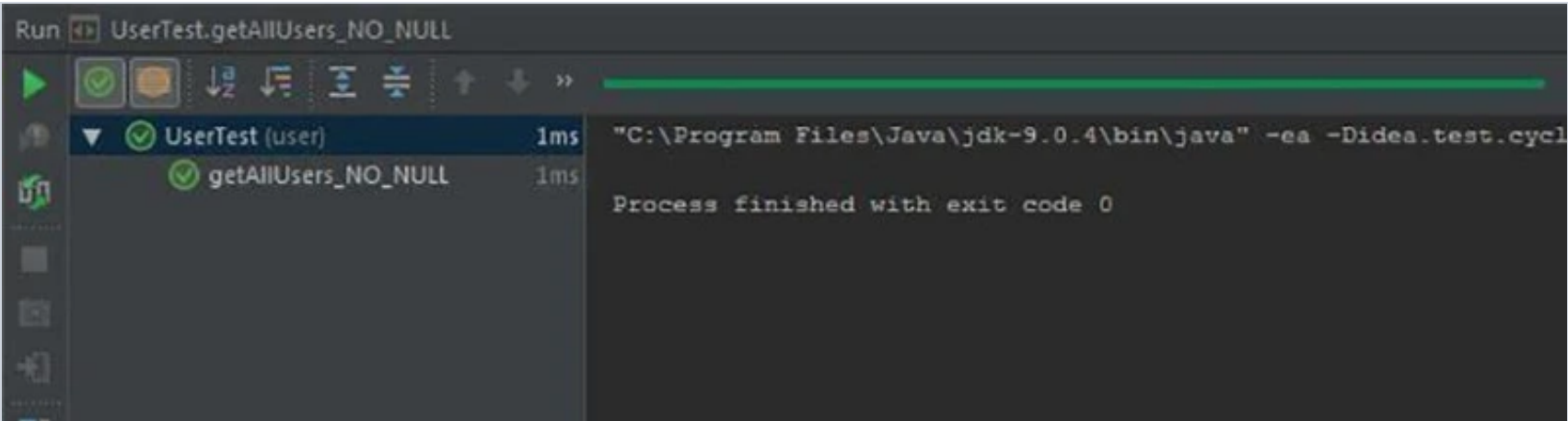


И тут мы можем пожинать первые плоды покрытия своего кода тестами. Как вы помните, поле `allUsers` мы инициализировали в конструкторе, и значит при вызове метода `getAllUsers()`, мы обратимся к объекту, который ещё не был инициализирован. Будем править, уберём инициализацию из конструктора, и сделаем её при объявлении поля.

```
1  private static Map<Integer, User> allUsers = new HashMap<>();
2
```

```
5         this.age = age;
6         this.sex = sex;
7
8         if (!hasUser()) {
9             countId++;
10            this.id = countId;
11            allUsers.put(id, this);
12        }
13    }
```

Запустим тест, теперь все хорошо.



не думаю что в main легко будет отловить NPE, думаю вы согласитесь что счет JUnit : main = 3 : 0

Дальше я все методы покрою тестами, и дам вам посмотреть, как это будет выглядеть...

Теперь класс тестов у нас выглядит так:

```
1 package user;
2
3 import org.junit.Assert;
4 import org.junit.Test;
5
6 import java.util.ArrayList;
7 import java.util.List;
8
9 import static org.junit.Assert.*;
10
11 public class UserTest {
12
13     @Test
14     public void getAllUsers() {
15         //создаем тестовые данные
16         User user = new User("Евгений", 35, Sex.MALE);
17         User user1 = new User("Марина", 34, Sex.FEMALE);
18         User user2 = new User("Алина", 7, Sex.FEMALE);
19
20         //создаем список expected и заполняем его данными нашего метода
21         List<User> expected = User.getAllUsers();
22
23         //создаем список actual в него помещаем данные для сравнения
24         //то что мы предполагаем метод должен вернуть
25         List<User> actual = new ArrayList<>();
26         actual.add(user);
```

```
29
30     //запускаем тест, в случае если список expected и actual не будут равны
31     //тест будет провален, о результатах теста читаем в консоли
32     Assert.assertEquals(expected, actual);
33 }
34
35 @Test
36 public void getAllUsers_NO_NULL() {
37     //добавим проверку на null
38     List<User> expected = User.getAllUsers();
39     Assert.assertNotNull(expected);
40 }
41
42 @Test
43 public void getAllUsers_MALE() {
44     User user = new User("Евгений", 35, Sex.MALE);
45     User user1 = new User("Марина", 34, Sex.FEMALE);
46     User user2 = new User("Алина", 7, Sex.FEMALE);
47
48     List<User> expected = User.getAllUsers(Sex.MALE);
49
50     List<User> actual = new ArrayList<>();
51     actual.add(user);
52
53     Assert.assertEquals(expected, actual);
54 }
55
56 @Test
57 public void getAllUsers_MALE_NO_NULL() {
58     //добавим проверку на null
59     List<User> expected = User.getAllUsers(Sex.MALE);
60     Assert.assertNotNull(expected);
61 }
62
63 @Test
64 public void getAllUsers_FEMALE() {
65     User user = new User("Евгений", 35, Sex.MALE);
66     User user1 = new User("Марина", 34, Sex.FEMALE);
67     User user2 = new User("Алина", 7, Sex.FEMALE);
68
69     List<User> expected = User.getAllUsers(Sex.FEMALE);
70
71     List<User> actual = new ArrayList<>();
72     actual.add(user1);
73     actual.add(user2);
74
75     Assert.assertEquals(expected, actual);
76 }
77
78 @Test
79 public void getAllUsers_FEMALE_NO_NULL() {
80     //добавим проверку на null
81     List<User> expected = User.getAllUsers(Sex.FEMALE);
82     Assert.assertNotNull(expected);
```



```
85 @Test
86 public void getHowManyUsers() {
87     User user = new User("Евгений", 35, Sex.MALE);
88     User user1 = new User("Марина", 34, Sex.FEMALE);
89     User user2 = new User("Алина", 7, Sex.FEMALE);
90
91     int expected = User.getHowManyUsers();
92
93     int actual = 3;
94
95     Assert.assertEquals(expected, actual);
96 }
97
98 @Test
99 public void getHowManyUsers_MALE() {
100     User user = new User("Евгений", 35, Sex.MALE);
101     User user1 = new User("Марина", 34, Sex.FEMALE);
102     User user2 = new User("Алина", 7, Sex.FEMALE);
103
104     int expected = User.getHowManyUsers(Sex.MALE);
105
106     int actual = 1;
107
108     Assert.assertEquals(expected, actual);
109 }
110
111 @Test
112 public void getHowManyUsers_FEMALE() {
113     User user = new User("Евгений", 35, Sex.MALE);
114     User user1 = new User("Марина", 34, Sex.FEMALE);
115     User user2 = new User("Алина", 7, Sex.FEMALE);
116
117     int expected = User.getHowManyUsers(Sex.FEMALE);
118
119     int actual = 2;
120
121     Assert.assertEquals(expected, actual);
122 }
123
124 @Test
125 public void getAllAgeUsers() {
126     User user = new User("Евгений", 35, Sex.MALE);
127     User user1 = new User("Марина", 34, Sex.FEMALE);
128     User user2 = new User("Алина", 7, Sex.FEMALE);
129
130     int expected = User.getAllAgeUsers();
131
132     int actual = 35 + 34 + 7;
133
134     Assert.assertEquals(expected, actual);
135 }
136
137 @Test
138 public void getAllAgeUsers_MALE() {
```

```
141         User user2 = new User("Алина", 7, Sex.FEMALE);
142
143         int expected = User.getAllAgeUsers(Sex.MALE);
144
145         int actual = 35;
146
147         Assert.assertEquals(expected, actual);
148     }
149
150     @Test
151     public void getAllAgeUsers_FEMALE() {
152         User user = new User("Евгений", 35, Sex.MALE);
153         User user1 = new User("Марина", 34, Sex.FEMALE);
154         User user2 = new User("Алина", 7, Sex.FEMALE);
155
156         int expected = User.getAllAgeUsers(Sex.FEMALE);
157
158         int actual = 34 + 7;
159
160         Assert.assertEquals(expected, actual);
161     }
162 }
```

Да не маленький получился, а что же будет при работе с большими проектами. Что же тут можно сократить, оценив все можно заметить, что тестовые данные мы создаем в каждом тесте, и тут нам на помощь приходят аннотации. Возьмем `@Before` — Аннотация `@Before` указывает на то, что метод будет выполняться перед каждым тестируемым методом `@Test`.

Вот так теперь будет выглядеть наш класс тестов с аннотацией `@Before`:

```
1 package user;
2
3 import org.junit.Assert;
4 import org.junit.Before;
5 import org.junit.BeforeClass;
6 import org.junit.Test;
7
8 import java.util.ArrayList;
9 import java.util.List;
10
11 import static org.junit.Assert.*;
12
13 public class UserTest {
14     private User user;
15     private User user1;
16     private User user2;
17
18     @Before
19     public void setUp() throws Exception {
20         user = new User("Евгений", 35, Sex.MALE);
21         user1 = new User("Марина", 34, Sex.FEMALE);
22         user2 = new User("Алина", 7, Sex.FEMALE);
23     }
24 }
```



```
27         List<User> expected = User.getAllUsers();
28
29         List<User> actual = new ArrayList<>();
30         actual.add(user);
31         actual.add(user1);
32         actual.add(user2);
33
34         Assert.assertEquals(expected, actual);
35     }
36
37     @Test
38     public void getAllUsers_NO_NULL() {
39         List<User> expected = User.getAllUsers();
40         Assert.assertNotNull(expected);
41     }
42
43     @Test
44     public void getAllUsers_MALE() {
45         List<User> expected = User.getAllUsers(Sex.MALE);
46
47         List<User> actual = new ArrayList<>();
48         actual.add(user);
49
50         Assert.assertEquals(expected, actual);
51     }
52
53     @Test
54     public void getAllUsers_MALE_NO_NULL() {
55         //добавим проверку на null
56         List<User> expected = User.getAllUsers(Sex.MALE);
57         Assert.assertNotNull(expected);
58     }
59
60     @Test
61     public void getAllUsers_FEMALE() {
62         List<User> expected = User.getAllUsers(Sex.FEMALE);
63
64         List<User> actual = new ArrayList<>();
65         actual.add(user1);
66         actual.add(user2);
67
68         Assert.assertEquals(expected, actual);
69     }
70
71     @Test
72     public void getAllUsers_FEMALE_NO_NULL() {
73         //добавим проверку на null
74         List<User> expected = User.getAllUsers(Sex.FEMALE);
75         Assert.assertNotNull(expected);
76     }
77
78     @Test
79     public void getHowManyUsers() {
80         int expected = User.getHowManyUsers();
```

```
83
84     Assert.assertEquals(expected, actual);
85 }
86
87 @Test
88 public void getHowManyUsers_MALE() {
89     int expected = User.getHowManyUsers(Sex.MALE);
90
91     int actual = 1;
92
93     Assert.assertEquals(expected, actual);
94 }
95
96 @Test
97 public void getHowManyUsers_FEMALE() {
98     int expected = User.getHowManyUsers(Sex.FEMALE);
99
100    int actual = 2;
101
102    Assert.assertEquals(expected, actual);
103 }
104
105 @Test
106 public void getAllAgeUsers() {
107     int expected = User.getAllAgeUsers();
108
109     int actual = 35 + 34 + 7;
110
111     Assert.assertEquals(expected, actual);
112 }
113
114 @Test
115 public void getAllAgeUsers_MALE() {
116     int expected = User.getAllAgeUsers(Sex.MALE);
117
118     int actual = 35;
119
120     Assert.assertEquals(expected, actual);
121 }
122
123 @Test
124 public void getAllAgeUsers_FEMALE() {
125     int expected = User.getAllAgeUsers(Sex.FEMALE);
126
127     int actual = 34 + 7;
128
129     Assert.assertEquals(expected, actual);
130 }
131 }
```

Ну как вам, уже веселее и легче читать ;)

Вот список аннотаций для JUnit с ними однозначно жить проще.

1	<code>@Test</code> – определяет что метод <code>method()</code> является тестовым.
2	<code>@Before</code> – указывает на то, что метод будет выполняться перед каждым тестируемым методом <code>@Test</code> .
3	<code>@After</code> – указывает на то что метод будет выполняться после каждого тестируемого метода <code>@Test</code>
4	<code>@BeforeClass</code> – указывает на то, что метод будет выполняться в начале всех тестов,
5	а точнее в момент запуска тестов(перед всеми тестами <code>@Test</code> ).
6	<code>@AfterClass</code> – указывает на то, что метод будет выполняться после всех тестов.
7	<code>@Ignore</code> – говорит, что метод будет проигнорирован в момент проведения тестирования.
8	(expected = <code>Exception.class</code> ) – указывает на то, что в данном тестовом методе
9	вы преднамеренно ожидаете <code>Exception</code> .
10	(timeout = <code>100</code> ) – указывает, что тестируемый метод не должен занимать больше чем <code>100</code> миллисекунд.

Основные методы класса `Assert` для проверки:

1	<code>fail(String)</code> – указывает на то что бы тестовый метод завалился при этом выводя текстовое сообщение.
2	<code>assertTrue([message], boolean condition)</code> – проверяет, что логическое условие истинно.
3	<code>assertEquals([String message], expected, actual)</code> – проверяет, что два значения совпадают.
4	Примечание: для массивов проверяются ссылки, а не содержание массивов.
5	<code>assertNull([message], object)</code> – проверяет, что объект является пустым <code>null</code> .
6	<code>assertNotNull([message], object)</code> – проверяет, что объект не является пустым <code>null</code> .
7	<code>assertSame([String], expected, actual)</code> – проверяет, что обе переменные относятся к одному объекту.
8	<code>assertNotSame([String], expected, actual)</code> – проверяет, что обе переменные относятся к разным объектам

Вот так мы можем добавить зависимость JUnit 4.12 в Maven

1	<code>&lt;dependency&gt;</code>
2	<code>  &lt;groupId&gt;junit&lt;/groupId&gt;</code>
3	<code>  &lt;artifactId&gt;junit&lt;/artifactId&gt;</code>
4	<code>  &lt;version&gt;4.12&lt;/version&gt;</code>
5	<code>  &lt;scope&gt;test&lt;/scope&gt;</code>
6	<code>&lt;/dependency&gt;</code>

продолжение тут -> [JUnit part II](#)

JavaCoder

Введите текст комментария

Elidriel

Уровень 35, Воронеж

27 марта, 16:06

...

кто будет пробовать повторить, юзайте вместо @Before @After эти аннотации - @BeforeEach @AfterEach

Ответить

-

0

+

Sobolev Evgeny

мужик в гараже

28 марта, 07:18

...

Это аннотации пятого junit

Ответить

-

0

+

Elidriel

Уровень 35, Воронеж

28 марта, 09:25

...

у меня в pom.xml такое -

1 <dependency>  
2 <groupId>junit</groupId>  
3 <artifactId>junit</artifactId>  
4 <version>4.13.1</version>  
5 <scope>test</scope>  
6 </dependency>

у товарища, написавшего статью стоит версия 4.12  
так что про пятый junit не скажу, но при 4.13.1 чистый @Before у меня не виделся программой

Ответить

-

0

+

hidden #2595317

Уровень 45

21 февраля, 19:53

...

1 еще мы можем кликнуть на <Click to see difference> получим  
2 такое окно, где можно посмотреть подробно с какими данными у нас проблема.

Не нашел куда в итоге нажимать.

Ответить

-

0

+

Anonymous #2957882

Уровень 40, Днепр, Ukraine

3 июля, 17:18

...

Ответить

-

0

+

hidden #2595317

Уровень 45

21 февраля, 19:42

...

1 создадим список actual в который поместим пользователей которых  
2 мы предполагаем что метод getAllUsers().assertEquals(actual,  
3 expected) ему мы и передадим списки, инспектируемый и актуальный.

Надо поправить смысловое содержание.

Ответить

-

0

+

fedyaka

Уровень 36, Кострома, Россия

20 февраля, 12:07

...

Но как ты с таким @Before будешь проверять на NOT\_NULL? ведь массив уже создается 😞  
Но спасибо, всё очень доступно и понятно :)

Ответить

-

+1

+

Alexey Pavlovsky

from в Krasnodar

14 января, 10:07

...

перестал читать, когда увидел логику в конструкторе...жесть...

Плюсую хорошая статья

Ответить

0

Модератор

Уровень 51, Молдова

10 декабря 2021, 03:23

...

а что за первая фотка с junit курсор надо ставить? ты что на пирожок с капустой фоткал?

Ответить

+4

LuneFox

инженер по сопровождению в BIFIT

EXPERT

15 января, 07:47

...

Секретные разработки специального алгоритма сжатия по методу Дж. Шакала, перед сохранением в .jpg сначала создаётся промежуточный .shkl :)

Ответить

0

Yarik

Таксист в Яндекс.Такси

30 ноября 2021, 13:09

...

Только после этой статьи задумался что же такое Assert-ы. Оказывается очень полезная штука.  
[Assert. Что это?](#)

Ответить

0

PaiMei in J#

Grand Master в Eagles' Claw

29 октября 2021, 11:39

...

В лекции создается класс User, у которого, в числе прочего, есть поле sex, типа Sex, только вот нигде не увидел описания класса Sex🤔 Мб в данном случае было бы проще сделать поле sex, типа boolean?

Ответить

+1

Destr

Уровень 28, Россия

29 октября 2021, 23:57

...

Подразумевается, видимо, что раз дошли до JUnit, то уже достаточно грамотные для того, чтоб создать "enum Sex" самостоятельно.

Ответить

+2

Anonymous #2950693

Уровень 1, Russian Federation

10 июля, 17:56

...

А вдруг не "enum" а что другое, "подразумевать" как то не относиться к точным понятиям, допустим я знаю про enum , но в то же время я знаю что еще плохо знаю Java, но все равно зачем плодить сомнения, лучше бы написали определение переменной.

Ответить

+1

LindX

Java Developer в JM

28 октября 2021, 03:37

...

Скриншоты вывода в консоль - это мощно

Ответить

+1

С Показать еще комментарии

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

НАЧАТЬ ОБУЧЕНИЕ

