

Первые тесты с помощью JUnit

JSP & Servlets
3 уровень, 1 лекция

ОТКРЫТА

Подключаем фреймворк JUnit

Для тестирования Java-кода у нас есть шикарный фреймворк под названием **JUnit**. Он отлично работает, постоянно обновляется, очень популярен и, конечно, с ним очень плотно интегрирована IntelliJ IDEA.

Сейчас все пользуются пятой версией этого фреймворка — **JUnit 5**, хотя во многих проектах ты еще можешь встретить его четвертую версию. Они не сильно отличаются, но мы все равно рассмотрим самую последнюю. Думаю, к тому времени, когда ты начнешь активно писать тесты, ты одобришь мой выбор.

Итак, как добавить в проект JUnit? После изучения Maven это будет просто: просто добавь этот код в свой pom.xml:

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.8.1</version>
  <scope>test</scope>
</dependency>
```

Кстати, если ты хочешь, чтобы Maven на этапе сборки (**test stage**) запустил твои тесты, то в pom.xml нужно добавить еще один фрагмент:

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.19.1</version>
  <dependencies>
    <dependency>
      <groupId>org.junit.platform</groupId>
      <artifactId>junit-platform-surefire-provider</artifactId>
      <version>1.3.2</version>
    </dependency>
  </dependencies>
</plugin>
```

Аннотация @Test

Допустим, у тебя есть класс, который ты хочешь протестировать. Как это лучше всего сделать? Давай начнем с какого-нибудь примера, а то сложно тестировать абстрактный класс :)

Предположим, у нас есть класс **Calculator**, которые умеет выполнять 4 базовые операции: сложение, вычитание, умножение и деление. Напишем его:

```
1 class Calculator {
2     public int add(int a, int b) {
3         return a+b;
4     }
5 }
```

```
6         public int sub(int a, int b) {
7             return a-b;
8         }
9
10        public int mul (int a, int b) {
11            return a*b;
12        }
13
14        public int div(int a, int b) {
15            return a/b;
16        }
17    }
```

Мы хотим протестировать методы этого класса. Мало ли, в будущем наменяют чего-нибудь и все перестанет работать. Как нам написать для него тесты?

Нам нужно создать еще один класс. Для этого обычно берут то же имя и добавляют суффикс `Test`. Для каждого метода нужно добавить хотя бы один тестовый метод. Пример:

```
1    class CalculatorTest {
2        @Test
3        public void add() {
4        }
5        @Test
6        public void sub() {
7        }
8        @Test
9        public void mul() {
10       }
11       @Test
12       public void div() {
13       }
14   }
```

Раньше существовало требование, чтобы имя метода начиналось со слова **test**, но теперь это не нужно.

Примеры тестов JUnit

Давай напишем несколько примеров, в которых протестируем методы нашего класса `CalculatorTest`:

```
1    class CalculatorTest {
2        @Test
3        public void add() {
4            Calculator calc = new Calculator();
5            int result = calc.add(2, 3);
6            assertEquals(5, result);
7        }
8
9        @Test
10       public void sub() {
11           Calculator calc = new Calculator();
12           int result = calc.sub(10, 10);
13           assertEquals(0, result);
14       }
15
16       @Test
```

```
17     public void mul() {
18         Calculator calc = new Calculator();
19         int result = calc.mul(-5, -3);
20         assertEquals(15, result);
21     }
22
23     @Test
24     public void div() {
25         Calculator calc = new Calculator();
26         int result = calc.div(2, 3);
27         assertEquals(0, result);
28     }
29 }
```

Вот так выглядит типичный тест с использованием JUnit. Из интересного: тут используется специальный метод `assertEquals()`: он сравнивает переданные ему параметры, о чем говорит слово equals в его названии.

Если параметры, переданные в `assertEquals()`, неравны, то метод кинет исключение и тест будет считаться проваленным. Это исключение не мешает выполнению остальных тестов.



Комментарии (5)

популярные

новые

старые

JavaCoder

Введите текст комментария

Владимир Чугуевец

Уровень 61, Краснодар, Россия

18 сентября, 22:57

...

Этот плагин точно должен содержать зависимость? у меня idea ругается

Кстати, если ты хочешь, чтобы Maven на этапе сборки (test stage) запустил твои тесты, то в pom.xml нужно добавить еще один фрагмент:

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.19.1</version>
  <dependencies>
    <dependency>
      <groupId>org.junit.platform</groupId>
      <artifactId>junit-platform-surefire-provider</artifactId>
      <version>1.3.2</version>
    </dependency>
  </dependencies>
</plugin>
```

Ответить

0

+

Stas S

Уровень 75, Гродно, Беларусь

1 сентября, 09:39

...

От куда информация про необходимость написания хотя бы 1 теста для каждого метода ?

Если все делать по примеру из лекции, получаем
WARNING: "The junit-platform-surefire-provider has been deprecated"
Зачем нам устаревший плагин ? Проще воспользоваться этим

1

<plugin>

2

<artifactId>maven-surefire-plugin</artifactId>

3

<version>2.22.2</version>

4

</plugin>

