**Руководство по JUnit:**

**Свойства JUnit**

Фреймворка JUnit состоит из следующих ключевых групп:

* **Fixtures (каркас)**
* **Test Suites (группа тестов)**
* **Test runners (сущности, которые выполняют тесты)**
* **Классы JUnit**

Разберём каждую из них отдельно:

**Fixtures**

Это фиксированное состояние множества (набора) объектов, которые служат базисом для выполнения тестов. Их цель – гарантировать, что существует хорошо известное и фиксированное окружение в котором тесты выполняются таким образом, что результаты повторяемы. Он включает в себя следующие методы:

* **setUp()**  
  Выполняется перед каждым запуском тестов
* **tearDown()**Выполняется после каждого тестового метода

**Test Suite**

Это группа, состоящая из нескольких тестов, которые запускаются вместе. Для запуска групповых тестов используются аннотации **@RunWith** и **@Suite**.

**Test Runners**

Используются для выполнения тестовых случаев.

**Классы JUnit**

Классы JUnit играют огромную роль  и используются для написания тестов и их выполнения. Наиболее важные классы указаны ниже:

* **Assert**Содержит множество методов утверждений.
* **TestCase**Содержит тестовые случаи, который определяют каркас для выполнения нескольких тестов.
* **TestResult**Содержит методы для хранения данных, полученных в результате выполнения тестовых случаев.

**Класс Assert**

Данный класс содержит набор методов-утверждений, которые крайне помогают разработчикам при написании тестов. Записываются только те утверждения, которые оказываются ложными (false). Ниже приведён список наиболее важных методов данного класса:

|  |  |
| --- | --- |
| № | Метод и описание |
| 1 | **void assertEquals(boolean expected, boolean actual)**  Проверяет, равенство значений двух примитивных типов данных. |
| 2 | **void assertFalse(boolean condition)**  Проверяет, является ли утверждение ложным. |
| 3 | **void assertNotNull(Object object)**  Проверяет, что объект на является **null** |
| 4 | **void assertNull(Object object)**  Проверяет, что данный объект не является типом **null** |
| 5 | **void assertTrue(boolean condition)**  Проверяет, что указанное выражение является истинным (true) |
| 6 | **void fail()**  Проваливает тест без вывода сообщений. |

**Класс TestCase**

Класс **org.junit.TestCase**определяет ряд каркас (fixture) для выполнения нескольких тестов. Наиболее использованые методы данного класса приведены ниже:

|  |  |
| --- | --- |
| № | Методы и описание |
| 1 | **int countTestCases()**  Возвращает количество выполнняемых тестовых случаев (test cases). |
| 2 | **TestResult createResult()**  Создаёт экземпялр класса TestResult по умолчанию. |
| 3 | **String getName()**  Вовращает имя тестового случая |
| 4 | **TestResult run()**  Выполняет тест, собирая результаты с помощью стандартного объекта TestResult. |
| 5 | **void run(TestResult result)**  Выполняет тестовый случай и собирает результаты в экземпляре TestResult |
| 6 | **void setName(String name)**  Задаёт имя тестовому случаю |
| 7 | **void setUp()**  Устанавливает каркас (fixture), например, открывает сетевое соединение |
| 8 | **void tearDown()**  Уничтожает каркас (fixture), например, закрывает сетевое соединение |

С версии JUnit4 тесты создаются с помощью аннотаций.

Список аннотаций:

**@Test**

**@Before @After**

**@BeforeClass @AfterClass**

**@Ignore**

**@Rule**

**@Parameterized**

Бывает такое, что для выполнения каждого тестового сценария вам необходим некоторый контекст, например, заранее созданные экземпляры классов. А после выполнения нужно освободить зарезервированные ресурсы. В этом случае вам понадобятся аннотации @Before и @After.

Метод помеченный @Before будет выполняться перед каждым тестовым случаем, а метод, помеченный @After - после каждого тестового случая.

Если же инициализацию и освобождение ресурсов нужно сделать всего один раз - соответственно до и после всех тестов - то используйте пару аннотаций @BeforeClass и @AfterClass.

@Test — указывает на тестирующий метод.

Если вы хотите указать, что определенный тест необходимо пропустить, то пометьте его аннотацией @Ignore. Хотя можно просто удалить аннотацию @Test.

@Rule – здесь можно указать правила для всех тестов (нужно унаследовать TestRule, пример:

**public TestRule timeout = new Timeout(100);**)

@Parameterized – здесь можно указать параметры, при указании параметров – нужно добавить перед классом аннотацию **@RunWith(Parameterized.class)**, она укажет классу что нужно начинать работу теста с указанными параметрами. Пример:

**@RunWith(Parameterized.class)**

**public class MyMathTest{**

**int a, b, c;**

public **MyMathTest(int a, int b, int c){**

**this.a=a;**

**this.b=b;**

**this.c=c;**

**}**

**public void testAdd() trows InterruptedException{**

**int result = MyMath.add(a, b);**

**assertEquals(c, result);**

**}**

**@Parameterized.Parameters**

**Public static Collection numbers(){**

**return Arrays.asList(new Object [] [] {{1,2,3}, {5,5,10}, {10,5,15}});**

**}**

Здесь указывается многократность выполнения тестов. К примеру тестируется сумирование, первая цифра сумируется со второй, третья это ожидаемый результат (1+2=3, 5+5=10, 10+5=15).

**“@RunWith” и “@Suite”** аннотации используются для запуска тест сьюта. Класс, написанный ниже, означает, что классы “JunitTest1” и “JunitTest2” будут запущены после запуска JunitTest5. Все остальные декларации будут внутри класса.

**@RunWith(Suite.class)**

**@Suite.SuiteClasses({**

**JunitTest1.class,**

**JunitTest2.class**

**})**

**public class JunitTest5 {**

**}**

У аннотации могут быть проставлены такие параметры:

**expected** – здесь можно указать какое исключение мы ожидаем

(пример: **@Test (expected = AriphmeticException.class)**

**timeout** - через какое время в миллисекундах прекратить выполнение теста и засчитать его как неуспешный.

Создание теста в IDE:

Правой клавишей мыши кликаем на классе, выбираем создание теста.

Выбираем нужное:

1. методы доступа
2. генерацию кода
3. генерацию комента

Хорошее видео <https://www.youtube.com/watch?reload=9&v=z9jEVLCF5_w>