JUnit

Модульное тестирование

JUnit 4 — это популярный фреймворк для выполнения модульного тестирования на Java.

Модульное (unit) — тестирование — заключается в тестировании только части некоторого функционала, а не всего продукта в целом. Проект разбивается на мелкие детали и проверяются конкретные его части, например, правильность работы конкретного метода.

Правила создания тестов

1) Тестовый класс должен размещаться в папке test, специально созданной для хранения тестов.
2) Название класса должно соответствовать одной из масок: *Test, Test*, *TestCase
3) Методы должны иметь возвращаемый тип void в сигнатуре и аннотацию @Test, которая определяет, что метод является тестовым.
4) Для проверки результата используются специальные методы assertEquals

Аннотации

@Test - определяет что метод method() является тестовым.

@Before - указывает на то, что метод будет выполняться перед КАЖДЫМ тестируемым методом @Test.

@After - указывает на то, что метод будет выполняться после КАЖДОГО тестируемого метода @Test.

@BeforeClass - указывает на то, что метод будет выполняться в начале всех тестов, а точнее в момент запуска тестов(ПЕРЕД ВСЕМИ тестами @Test).

@AfterClass - указывает на то, что метод будет выполняться ПОСЛЕ ВСЕХ тестов.

@Ignore - говорит, что метод будет проигнорирован в момент проведения тестирования.

@Test (expected = Exception.class) - казывает на то, что в данном тестовом методе вы преднамеренно ожидается Exception.

@Test (timeout = x)- указывает, что тестируемый метод не должен занимать больше чем x миллисекунд.

Методы

assertTrue(boolean condition) – проверяет, что логическое условие истинно.

assertEquals(expected, actual) – проверяет, что два значения совпадают (для массивов проверяются ссылки, а не содержание массивов).

assertNull(object) – проверяет, что объект является пустым null.

assertNotNull(object) – проверяет, что объект не является пустым null.

assertSame(expected, actual) – проверяет, что обе переменные относятся к одному объекту.

assertNotSame(expected, actual) – проверяет, что обе переменные относятся к разным объектам.

fail(String) – указывает на то чтобы тестовый метод завалился при этом выводя текстовое сообщение.

```java
public class CalculationUtils {

    public static int add(int a, int b) {
        return a + b;
    }

    public static int sub(int a, int b) {
        return a - b;
    }
}
```

С помощью @Before будет выполняться инициализация полей перед каждым тестовым методом

аннотация @Test говорит о том, что это тестовый метод

передаем в метод для сравнения число 13

если наш метод правильно складывает числа, то ожидаемый результат работы этого метода будет 12.

```java
public class CalculationUtilsTest {
    int x;
    int y;

    @Before
    public void setUp() {
        x = 5;
        y = 7;
    }

    @Test
    public void testAdd() {
        int result = 13;
        Assert.assertEquals( message: "5 + 7 must be equal 12", result, CalculationUtils.add(x, y));
    }
    @Test
    public void testSub() {
        int result = -2;
        Assert.assertEquals(result, CalculationUtils.sub(x, y));
    }
}
```

```java
@Test(expected = IllegalArgumentException.class)
public void testCreateAirplanesList_wrongInputData() {

    AirlineService airlineService = new AirlineServiceImpl();
    List<Airplane> airplanes = airlineService.createAirplanesList( filename: "airplanesWrongData.txt");

    assertTrue(airplanes.get(0) instanceof CargoAirplane);
    assertEquals(airplanes.get(0).getSideNumber(), actual: "GH12");
    assertEquals(airplanes.get(0).getManufacturer(), actual: "Sukhoi");
    assertEquals(airplanes.get(0).getModel(), actual: "C4312");
    assertEquals(airplanes.get(0).getAirplaneType(), AirplaneType.CARGO);
    assertEquals(airplanes.get(0).getCrewQuantity(), actual: 2);
    assertEquals(airplanes.get(0).getMaxSpeed(), actual: 850);
    assertEquals(airplanes.get(0).getMaxAltitude(), actual: 10000);
    assertEquals(airplanes.get(0).getMaxFlightRange(), actual: 5000);
    assertEquals(airplanes.get(0).getFuelSupply(), actual: 245);
    assertEquals(airplanes.get(0).getFuelConsumption(), actual: 170, delta: 0.01);
    assertEquals(((CargoAirplane)airplanes.get(0)).getCargoCapacity(), actual: 220);
}
```

В этом примере я тестирую, что будет выброшено исключение. Для этого указано (expected = и нужная нам ошибка))