**Junit 5**

видео - <https://www.youtube.com/watch?v=y5E2QUyUBjQ>

Чтобы в IDEA IntelliJ автоматически создать шаблон тестирования для конкретного класса нужно – находясь в открытом классе зайти в верхнюю закладку Navigate, а в ней выбрать закладку Test.

1. **Если нужно на 5 версии запустить тесты 4 версии**

В Maven дописать блок:



1. **Для использования нового функционала 5-й версии**

В Maven дописать блок:



1. **Для возможности запуска тестов 5 версии из Maven**

В Maven дописать блок:



1. **Если мы хотим вместо названия метода выводить свою надпись, то нужно внизу дописать**

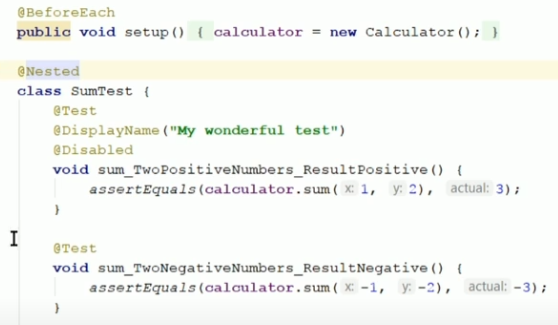


1. **Вместо *@Ignore* нужно писать *@Disabled*.**

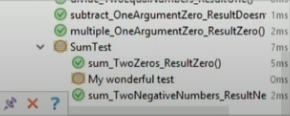
Если тест просто закомментировать, то не покажет, что такой тест есть но он не был запущен.

1. **Группировка тестов**

Внутри класса написать еще один класс и добавить перед ним аннотацию



и тогда при выполнении мы увидим в окошке



1. **Для параллельного выполнения тестов**

нужно дописать после строки названия метода



Если на компьютере с 4 ядрами написать не >1, а >4 , то при выполнении он этот тест покажет как @Disabled

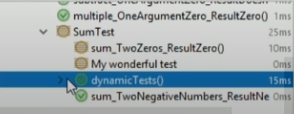
1. **Написание динамических тестов**

Для обозначения этого используется аннотация *@TestFactory* и тип *Stream<DynamicTest>*

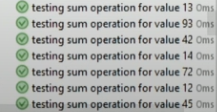


**,**

при выполнении мы увидим

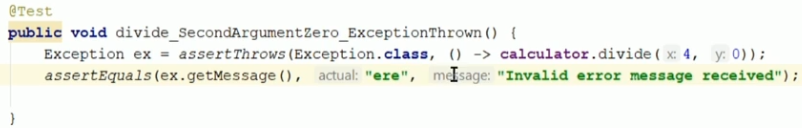


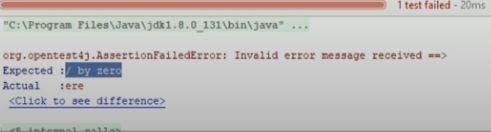
а в нем все 100 тестов



1. **Проверка выбрасывания исключений**

Вначале проверяем выпадает ли при выполнении метода данное исключение или нет, а потом дополнительно*(если хотим)* проверяем текст выпавшего исключения и в случае если текст не такой, то выводим сообщение

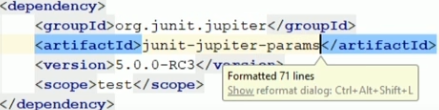




, только тут нужно будет проверять собственные исключения, которые будут выпадать в придуманных самостоятельно случаях не стандартных для языка Java. Иначе это проверка правильности работы самого языка Java *(что называется “А не слишком ли много вы на себя берете?”)*.

1. **Для написания метода тестирующего одну и туже функцию с разными параметрами**

В Maven дописать блок:

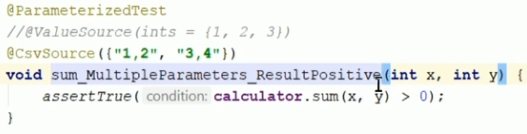


Дальше вторая аннотация будет обозначать тип “базы данных”, из которой будут использоваться данные в качестве подставляемых параметров.

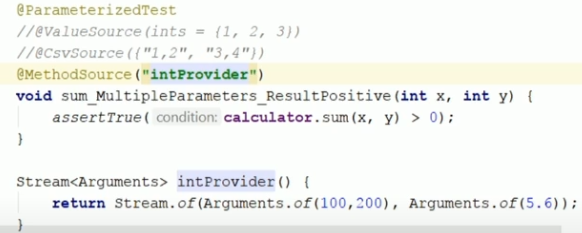
@ValueSource - если наш тестовый метод принимает только один параметр метода, который является либо String, либо примитивным типом (int, long или double).

@CsvSource - если нам нужно передать несколько параметров метода нашему параметризованному тесту.

Когда заранее известны параметры, то писать их блоками в кавычках



Когда не известны



при этом для того, чтобы это сработало метод intProvider() должен быть не как тут, а static (т.е. “static Stream<Arguments> intProvider()”) и размещать его нужно в главном классе, а не в подклассах???.

Когда данные генерируются в отдельном классе, то:

**class** **StringsUnitTest** {

@ParameterizedTest @MethodSource("com.baeldung.parameterized.StringParams#blankStrings")

**void** **isBlank\_ShouldReturnTrueForNullOrBlankStringsExternalSource**(String input) { assertTrue(Strings.isBlank(input));

}

}

**public** **class** **StringParams** {

**static** Stream<String> **blankStrings**() {

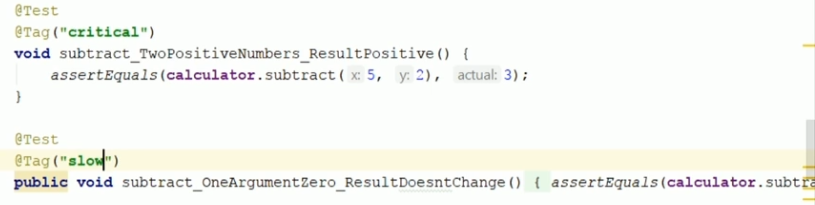
**return** Stream.of(null, "", " ");

}

}

1. **Для запуска теста по определенным группам**

используется аннотация @Tag

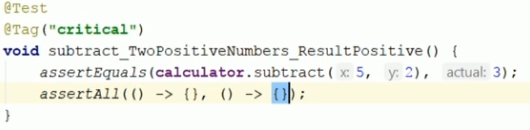


И потом в Maven дописать блок, где например мы включаем @Tag(“critical”) и исключаем @Tag(“slow”):



1. **Если необходимо одновременно сделать несколько разных проверок одного и того же**

Для того, чтобы при невыполнении первой же проверки остальные проверки все-таки сработали (***в отличие от assertEquals()***) нужно использовать assertAll():



1. **Для работы со/в Spring смотреть видео с 39 по 41 минуты.**