**Материалы:**

1. <https://junit.org/junit5/docs/current/user-guide/#overview>

**Возможности:**

1. Использование мета-аннотаций и составных аннотаций для тестов.
2. Генерация имени для тестового класса, методов.
3. Создание собственного генератора имен и использования его для генерации имен. Возможность задания генератора имен по умолчанию.
4. Создание методов жизненного цикла (Обозначаются аннотациями - @BeforeAll, @AfterAll, @BeforeEach, or @AfterEach.
5. Использование Assertions – утверждений для проверки функций тестируемых классов. Использование утверждений сторонних библиотек (Например, hamcrest). С помошью утверждений можно проверить:

* Равенство двух примитивных значений.
* Равенство двух массивов.
* Равенство двух Iterable объектов.
* Является ли выражение истинным или ложным.
* Равен ли аргумент null.
* Ссылаются ли обе переменных на один и тот же объект.
* Группу утверждений (Проверяет все утверждения и только потом выводит результат).
* Выброс исключения.
* Превышение временного интервала (имеются два различных метода – один запускается в том же потоке, в которой исполняется код, вызывающий тестовый метод + при превышении таймаута выполнение теста не прерывается, второй запускается в собственном потоке и прерывание выполнения происходит по таймауту).
* Различные утверждения сторонних библиотек.
* …

Кроме того можно “завалить тест” искусственно с помощью метода “fail”.

1. Выполнение тестов при определенных допущениях – assumptions, иначе тест отменяется (“aborted”).
2. Исключение тестового класса или отдельных его тестовых методов из процесса тестирования (Disabling). Достигается через:

* Ааннотацию [**@Disabled**](https://junit.org/junit5/docs/current/api/org/junit/jupiter/api/Disabled.html);
* Аннотации условного выполнения тестов ([**Conditional Test Execution**](https://junit.org/junit5/docs/current/user-guide/#writing-tests-conditional-execution));
* **Custom** [**ExecutionCondition**](https://junit.org/junit5/docs/current/user-guide/#extensions-conditions).

1. Использование тегов для тестов и возможность запуска тестов по ним (Только тесты с определенными тегами будут запущены, или наоборот проигнорированы).
2. Задание порядка выполнения тестов (Можно реализовывать свои классы или использовать готовые).
3. Изменение жизненного цикла выполнения тестового класса. (Можно создать как один общий Instance тестового класса для всех методов, так и отдельный для каждого из тестовых методов).
4. Создание вложенных тестовых классов (Nested).
5. Внедрение зависимостей в конструкторы класса и тестовые методы (С помощью классов реализующих интерфейс ParameterResolver). Три встроенные автоматически зарегистрированные реализации:

* [TestInfoParameterResolver](https://github.com/junit-team/junit5/tree/r5.5.1/junit-jupiter-engine/src/main/java/org/junit/jupiter/engine/extension/TestInfoParameterResolver.java);
* [RepetitionInfoParameterResolver](https://github.com/junit-team/junit5/tree/r5.5.1/junit-jupiter-engine/src/main/java/org/junit/jupiter/engine/extension/RepetitionInfoParameterResolver.java);
* [TestReporterParameterResolver](https://github.com/junit-team/junit5/tree/r5.5.1/junit-jupiter-engine/src/main/java/org/junit/jupiter/engine/extension/TestReporterParameterResolver.java).

Регистрация собственных классов через аннотацию @ExtendWith(\*.class)

**Замечания:**

При использовании Maven имена тестовых классов должны соответствовать маскам:

1. Test\*
2. \*Test
3. \*TestCase

или же соответствующий плагин (maven-surefire-plugin) должен быть перенастроен.

С помощью плагина “ maven-surefire-report-plugin” можно сформировать отчет по итогам тестирования.