JUnit

**JUnit -** это фреймворк для языка программирования Java, предназначенный для автоматического тестирования программ. Его основное назначение — unit-тестирование, то есть такое, когда по отдельности проверяется функциональность каждого компонента программы. Юнит-тестирование еще называют модульным.

**JUnit5 делится на 3 модуля:**

* **Платформа JUnit** служит основой для запуска тестовых фреймворков на JVM. В нем определяется TestEngine API для разработки платформы тестирования, которая выполняется на платформе. Кроме того, платформа предоставляет консольный лаунчер для запуска платформы из командной строки и движок JUnit Platform Suite для запуска пользовательского набора тестов с использованием одного или нескольких движков тестирования на платформе.
* **JUnit Jupiter** - это комбинация модели программирования и модели расширения для написания тестов и расширений в JUnit 5. Подпроект Jupiter предоставляет TestEngine для запуска тестов на основе Jupiter на платформе.
* **JUnit Vintage** предоставляет TestEngine для запуска тестов на основе JUnit 3 и JUnit 4 на платформе. Для этого требуется, чтобы JUnit 4.12 или более поздняя версия присутствовала в пути к классу или пути к модулю.

**Следующий пример дает представление о минимальных требованиях для написания теста в JUnit:**

**import static org.junit.jupiter.api.Assertions.assertEquals;**

**import example.util.Calculator;**

**import org.junit.jupiter.api.Test;**

**class MyFirstJUnitJupiterTests {**

**private final Calculator calculator = new Calculator();**

**@Test**

**void addition() {**

**assertEquals(2, calculator.add(1, 1));**

**}**

**}**

Аннотации

**@Test** - обозначает, что метод является методом тестирования. В отличие от @Test аннотации JUnit 4, в этой аннотации не объявляются никакие атрибуты, поскольку тестовые расширения в JUnit Jupiter работают на основе своих собственных специальных аннотаций. Такие методы наследуются, если они не переопределены.

**@ParameterizedTest** - обозначает, что метод является параметризованным тестом. Такие методы наследуются, если они не переопределены.

**@ValueSource –** используется для указания массива при использовании параметризированных методов. Следующие типы литеральных значений поддерживаются:

* short
* byte
* int
* long
* float
* double
* char
* boolean
* java.lang.String
* java.lang.Class

**@NullSource** - предоставляет единственный null аргумент для аннотированного @ParameterizedTest метода. Не может использоваться для параметра, имеющего примитивный тип.

**@EmptySource** - предоставляет единственный пустой аргумент аннотированному @ParameterizedTest методу для параметров следующих типов: java.lang.String, java.util.Listjava.util.Set, java.util.Map int[], примитивные массивы (например, char[][], String[] и т.д.), объектные массивы (например,Integer[][], ,, и т.д.). Подтипы поддерживаемых типов не поддерживаются.

**@EnumSource** - предоставляет удобный способ использования Enum констант при использовании параметризированных методов.

**@MethodSource** - позволяет ссылаться на один или несколько заводских методов тестового класса или внешних классов при использовании параметризированных методов.

**@CsvSource** - позволяет отображать списки аргументов в виде значений, разделенных запятым при использовании параметризированных методов.

**@CsvFileSource** - позволяет использовать файлы значений, разделенных запятыми (CSV), из classpath или локальной файловой системы при использовании параметризированных методов.

**@ArgumentsSource** может использоваться для указания пользовательского, многократно используемого ArgumentsProvider при использовании параметризированных методов.

**@NullAndEmptySource** - составленная аннотация, которая сочетает в себе функциональность @NullSource и @EmptySource.

**@RepeatedTest** - обозначает, что метод является шаблоном теста для повторного теста. Такие методы наследуются, если они не переопределены.

**@TestFactory** - обозначает, что метод является тестовой фабрикой для динамических тестов. Такие методы наследуются, если они не переопределены.

**@TestTemplate** - обозначает, что метод является шаблоном для тестовых случаев, предназначенным для многократного вызова в зависимости от количества контекстов вызова, возвращаемых зарегистрированными поставщиками. Такие методы наследуются, если они не переопределены.

**@TestClassOrder** - используется для настройки порядка выполнения тестового класса для @Nested тестовых классов в аннотированном тестовом классе. Такие аннотации наследуются.

**@TestMethodOrder** - используется для настройки порядка выполнения метода тестирования для аннотированного тестового класса; аналогично JUnit 4 @FixMethodOrder. Такие аннотации наследуются.

**@TestInstance** - используется для настройки жизненного цикла экземпляра теста для аннотированного тестового класса. Такие аннотации наследуются.

**@DisplayName** - объявляет пользовательское отображаемое имя для тестового класса или метода тестирования. Такие аннотации не наследуются.

**@DisplayNameGeneration** - объявляет пользовательский генератор отображаемых имен для тестового класса. Такие аннотации наследуются.

* **Standard -** соответствует стандартному поведению генерации отображаемых имен, существующему с момента выпуска JUnit Jupiter 5.0.
* **Simple -** удаляет завершающие скобки для методов без параметров.
* **ReplaceUnderscores -** заменяет подчеркивания пробелами.
* **IndicativeSentences -** генерирует полные предложения путем объединения названий теста и входящих в него классов.

**вкратце, отображаемое имя для тестового класса или метода определяется в соответствии со следующими правилами приоритета:**

* значение @DisplayName аннотации, если она присутствует
* путем вызова, DisplayNameGenerator указанного в @DisplayNameGeneration аннотации, если таковой имеется
* путем вызова параметра по умолчанию, DisplayNameGenerator настроенного с помощью параметра конфигурации, если он присутствует
* позвонив org.junit.jupiter.api.DisplayNameGenerator.Standard

**@BeforeEach** - обозначает, что аннотированный метод должен выполняться перед каждым @Test, @RepeatedTest, @ParameterizedTest или @TestFactory методом в текущем классе; аналогично JUnit 4 @Before. Такие методы наследуются, если только они не переопределены или заменены (т. е. заменены только на основе подписи, независимо от правил видимости Java).

**@AfterEach** - обозначает, что аннотированный метод должен выполняться после каждого @Test, @RepeatedTest, @ParameterizedTest или @TestFactory метода в текущем классе; аналогично JUnit 4 @After. Такие методы наследуются, если только они не переопределены или заменены (т. е. заменены только на основе подписи, независимо от правил видимости Java).

**@BeforeAll** - обозначает, что аннотированный метод должен выполняться перед всеми @Test, @RepeatedTest, @ParameterizedTest и @TestFactory методами в текущем классе; аналогично методам JUnit 4 @BeforeClass. Такие методы наследуются – если только они не скрыты, переопределены или заменены (т. Е. заменяются только на основе подписи, независимо от правил видимости Java) – и должны быть, static если не используется жизненный цикл экземпляра теста для каждого класса.

**@AfterAll** - обозначает, что аннотированный метод должен выполняться после всех @Test, @RepeatedTest, @ParameterizedTest и @TestFactory методов в текущем классе; аналогично методам JUnit 4 @AfterClass. Такие методы наследуются – если только они не скрыты, переопределены или заменены (т. Е. заменяются только на основе подписи, независимо от правил видимости Java) – и должны быть, static если не используется жизненный цикл экземпляра теста для каждого класса.

**@Nested** - обозначает, что аннотированный класс является нестатическим вложенным тестовым классом. На Java с 8 по Java 15 методы @BeforeAll и @AfterAll не могут использоваться непосредственно в @Nested тестовом классе, если не используется "жизненный цикл экземпляра теста для каждого класса". Начиная с Java 16, методы @BeforeAll и @AfterAll могут быть объявлены как static в @Nested тестовом классе в любом режиме жизненного цикла тестового экземпляра. Такие аннотации не наследуются.

**@Tag** - используется для объявления тегов для фильтрации тестов либо на уровне класса, либо на уровне метода; аналогично тестовым группам в TestNG или категориям в JUnit 4. Такие аннотации наследуются на уровне класса, но не на уровне метода.

**@Disabled** - используется для отключения тестового класса или метода тестирования; аналогично JUnit 4 @Ignore. Такие аннотации не наследуются.

**@Timeout** - используется для сбоя теста, тестовой фабрики, тестового шаблона или метода жизненного цикла, если его выполнение превышает заданную продолжительность. Такие аннотации наследуются.

**@ExtendWith** - используется для декларативной регистрации расширений. Такие аннотации наследуются.

**@RegisterExtension** - используется для регистрации расширений программно с помощью полей. Такие поля наследуются, если они не затемнены.

**@TempDir** - используется для предоставления временного каталога посредством ввода поля или параметра в метод жизненного цикла или метод тестирования; находится в org.junit.jupiter.api.io пакете.

**@EnabledOnOs и @DisabledOnOs –** используется для включения и отключения контейнеров и тестов в конкретной операционной системы.

**@EnabledOnJre и @DisabledOnJre –** используется для включения или отключения в определенных версиях Java Runtime Environment.

**@EnabledForJreRange и @DisabledForJreRange** – используется для включены или отключены в определенном диапазоне версий JRE.

**@EnabledInNativeImage и @DisabledInNativeImage -** используетсядлявключения или отключения контейнера или теста в собственном изображении GraalVM.

**@EnabledIfSystemProperty и @DisabledIfSystemProperty** - **-** используетсядлявключения или отключения контейнера или теста на основе значения named системного свойства. Значение, предоставляемое через matches атрибут, будет интерпретироваться как регулярное выражение.

**@EnabledIfEnvironmentVariable и @DisabledIfEnvironmentVariable** - используетсядлявключения или отключения контейнера или теста на основе значения named переменной среды из базовой операционной системы. Значение, предоставляемое через matches атрибут, будет интерпретироваться как регулярное выражение.

**@EnabledIf и @DisabledIf -** используетсядлявключения или отключения контейнера или теста на основе метода условий. Метод condition должен иметь boolean возвращаемый тип и может принимать либо без аргументов, либо один ExtensionContext аргумент.

**@TestMethodOrder** – используется для управления порядком выполнения тестов.

* **MethodOrderer.DisplayName** сортирует методы тестирования в алфавитно-цифровом виде на основе их отображаемых имен.
* **MethodOrderer.MethodName** сортирует методы тестирования в алфавитно-цифровом виде на основе их названий и формальных списков параметров.
* **MethodOrderer.OrderAnnotation**: сортирует методы тестирования численно на основе значений, указанных в @Order аннотации.
* **MethodOrderer.Random**: псевдослучайно упорядочивает методы тестирования и поддерживает настройку пользовательского начального.
* **MethodOrderer.Alphanumeric**: сортирует методы тестирования в алфавитно-цифровом виде на основе их названий и формальных списков параметров; устарел в пользу MethodOrderer.MethodName, будет удален в версии 6.0.

**@TestClassOrder** – используется для управления порядком выполнения тестовых классов. Позволяет локально настроить порядок выполнения тестового класса для @Nested тестовых классов. объявите @TestClassOrder аннотацию во вложенном классе для @Nested тестовых классов, которые вы хотите заказать, и укажите ссылку на класс ClassOrderer реализации, которую вы хотели бы использовать непосредственно в @TestClassOrder аннотации. Настроенное ClassOrderer будет рекурсивно применено к @Nested тестовым классам и их @Nested тестовым классам.

**@RepeatedTest** – используется для повторения теста заданное количество раз.

**@ConvertWith –** используется для явного преобразования аргументов.

**@ResourceLock** - позволяет объявить, что тестовый класс или метод использует определенный общий ресурс, которому требуется синхронизированный доступ для обеспечения надежного выполнения теста. Общий ресурс идентифицируется уникальным именем, которое является String. Имя может быть определено пользователем или одной из предопределенных констант в Resources: SYSTEM\_PROPERTIES, SYSTEM\_OUT, SYSTEM\_ERR, LOCALE TIME\_ZONE.

**Аннотации JUnit Jupiter** можно использовать как мета-аннотации. Это означает, что вы можете определить свою собственную составленную аннотацию, которая автоматически унаследует семантику своих мета-аннотаций.

Конфигурации

**junit.jupiter.execution.parallel.config.strategy** – используется для выбора стратегии.

* **dynamic -** вычисляет желаемый параллелизм на основе количества доступных процессоров / ядер, умноженного на junit.jupiter.execution.parallel.config.dynamic.factor параметр конфигурации (по умолчанию имеет значение 1).
* **fixed -** использует обязательный junit.jupiter.execution.parallel.config.fixed.parallelism параметр конфигурации в качестве желаемого параллелизма. Необязательный junit.jupiter.execution.parallel.config.fixed.max-pool-size параметр конфигурации может использоваться для ограничения максимального количества потоков.
* **custom -** позволяет указать пользовательскую ParallelExecutionConfigurationStrategy реализацию с помощью обязательного junit.jupiter.execution.parallel.config.custom.class параметра конфигурации для определения желаемой конфигурации.

**junit.jupiter.execution.parallel.enabled** - включить параллельное выполнение теста: true,

false.Значение по умолчанию false.

**junit.jupiter.execution.parallel.mode.default** - режим выполнения узлов в тестовом дереве по умолчанию: concurrent, same\_thread. Значение по умолчанию same\_thread.

**junit.jupiter.execution.parallel.mode.classes.default** - режим выполнения классов верхнего уровня по умолчанию: concurrent, same\_thread. Значение по умолчанию same\_thread.

**junit.jupiter.execution.parallel.config.strategy** - стратегия выполнения для желаемого параллелизма и максимального размера пула: dynamic, fixed, custom.Значение по умолчанию dynamic.

**junit.jupiter.execution.parallel.config.dynamic.factor** - коэффициент, который необходимо умножить на количество доступных процессоров / ядер, чтобы определить желаемый параллелизм для dynamic стратегии настройки: положительное десятичное число. Значение по умолчанию 1.0.

**junit.jupiter.execution.parallel.config.fixed.parallelism** - желаемый параллелизм для fixed стратегии конфигурирования: положительное целое число.

**junit.jupiter.execution.parallel.config.fixed.max-pool-size** - желаемый максимальный размер пула базового пула fork-join для fixed стратегии настройки: положительное целое число, должно быть больше или равно junit.jupiter.execution.parallel.config.fixed.parallelism. Значение по умолчанию 256 + значение junit.jupiter.execution.parallel.config.fixed.parallelism.

**junit.jupiter.execution.parallel.config.fixed.saturate** - отключите насыщение базового пула fork-join для fixed стратегии настройки: true, false. Значение по умолчанию true.

**junit.jupiter.execution.parallel.config.custom.class** - полное имя класса ParallelExecutionConfigurationStrategy, которое будет использоваться для custom стратегии конфигурирования например, org.example.CustomStrategy.

Assert

**Assert** – это класс который предоставляет набор методов утверждения, полезных для написания тестов. Только неудачные утверждения записываются. Эти методы могут быть использованы напрямую: Assert.assertEquals(...) однако они читаются лучше, если они на которые ссылаются через статический импорт.

**Методы:**

**assertArrayEquals(boolean[] expecteds, boolean[] actuals)** - утверждает, что два логических массива равны.

**assertArrayEquals(byte[] expecteds, byte[] actuals)** - утверждает, что два байтовых массива равны.

**assertArrayEquals(char[] expecteds, char[] actuals)** - утверждает, что два массива символов равны.

**assertArrayEquals(double[] expecteds, double[] actuals, double delta)** - утверждает, что два двойных массива равны.

**assertArrayEquals(float[] expecteds, float[] actuals, float delta)** - утверждает, что два массива с плавающей запятой равны.

**assertArrayEquals(int[] expecteds, int[] actuals)** - утверждает, что два массива int равны.

**assertArrayEquals(long[] expecteds, long[] actuals)** - утверждает, что два длинных массива равны.

**assertArrayEquals(Object[] expecteds, Object[] actuals)** - утверждает, что два массива объектов равны.

**assertArrayEquals(short[] expecteds, short[] actuals)** - утверждает, что два коротких массива равны.

**assertArrayEquals(String message, boolean[] expecteds, boolean[] actuals)** - утверждает, что два логических массива равны.

**assertArrayEquals(String message, byte[] expecteds, byte[] actuals)** - утверждает, что два байтовых массива равны.

**assertArrayEquals(String message, char[] expecteds, char[] actuals)** - утверждает, что два массива символов равны.

**assertArrayEquals(String message, double[] expecteds, double[] actuals, double delta)** - утверждает, что два двойных массива равны.

**assertArrayEquals(String message, float[] expecteds, float[] actuals, float delta)** - утверждает, что два массива с плавающей запятой равны.

**assertArrayEquals(String message, int[] expecteds, int[] actuals)** - утверждает, что два массива int равны.

**assertArrayEquals(String message, long[] expecteds, long[] actuals)** - утверждает, что два длинных массива равны.

**assertArrayEquals(String message, Object[] expecteds, Object[] actuals)** - утверждает, что два массива объектов равны.

**assertArrayEquals(String message, short[] expecteds, short[] actuals) -** утверждает, что два коротких массива равны.

**assertEquals(double expected, double actual, double delta)** - утверждает, что два удвоения равны в пределах положительной дельты.

**assertEquals(float expected, float actual, float delta)** - утверждает, что два значения с плавающей точкой равны в пределах положительной дельты.

**assertEquals(long expected, long actual)** - утверждает, что две длины равны.

**assertEquals(Object expected, Object actual)** - утверждает, что два объекта равны.

**assertEquals(String message, double expected, double actual, double delta)** - утверждает, что два удвоения равны в пределах положительной дельты.

**assertEquals(String message, float expected, float actual, float delta)** - утверждает, что два значения с плавающей точкой равны в пределах положительной дельты.

**assertEquals(String message, long expected, long actual)** - утверждает, что две длины равны.

**assertEquals(String message, Object expected, Object actual)** - утверждает, что два объекта равны.

**assertFalse(boolean condition)** - утверждает, что условие является ложным.

**assertFalse(String message, boolean condition)** - утверждает, что условие является ложным.

**assertNotEquals(double unexpected, double actual, double delta)** - утверждает, что два удвоения не равны в пределах положительной дельты.

**assertNotEquals(float unexpected, float actual, float delta)** - утверждает, что два значения с плавающей точкой не равны в пределах положительной дельты.

**assertNotEquals(long unexpected, long actual)** - утверждает, что две длины не равны.

**assertNotEquals(Object unexpected, Object actual)** - утверждает, что два объекта не равны.

**assertNotEquals(String message, double unexpected, double actual, double delta)** - утверждает, что два удвоения не равны в пределах положительной дельты.

**assertNotEquals(String message, float unexpected, float actual, float delta) -** утверждает, что два значения с плавающей точкой не равны в пределах положительной дельты.

**assertNotEquals(String message, long unexpected, long actual)** - утверждает, что две длины не равны.

**assertNotEquals(String message, Object unexpected, Object actual)** - утверждает, что два объекта не равны.

**assertNotNull(Object object)** - утверждает, что объект не равен нулю.

**assertNotNull(String message, Object object)** - утверждает, что объект не равен нулю.

**assertNotSame(Object unexpected, Object actual)** - утверждает, что два объекта не ссылаются на один и тот же объект.

**assertNotSame(String message, Object unexpected, Object actual)** - утверждает, что два объекта не ссылаются на один и тот же объект.

**assertNull(Object object)** - утверждает, что объект равен нулю.

**assertNull(String message, Object object)** - утверждает, что объект равен нулю.

**assertSame(Object expected, Object actual)** - утверждает, что два объекта ссылаются на один и тот же объект.

**assertSame(String message, Object expected, Object actual)** - утверждает, что два объекта ссылаются на один и тот же объект.

**assertThrows(Class<T> expectedThrowable, ThrowingRunnable runnable) -** утверждает, что runnable выдает исключение типа expectedThrowable при выполнении.

**assertThrows(String message, Class<T> expectedThrowable, ThrowingRunnable runnable)** - утверждает, что runnable выдает исключение типа expectedThrowable при выполнении.

**assertTrue(boolean condition)** - утверждает, что условие истинно.

**assertTrue(String message, boolean condition)** - утверждает, что условие истинно.

**fail()** - сбой теста без сообщения.

**fail(String message)** - сбой теста с данным сообщением.