JUnit. Расширяем тестовый фреймворк

Цели: Познакомиться с тестовыми фреймворками

План урока:

- JUnit
- TestNG
- Аннотации
- Отчеты

JUnit

JUnit — фреймворк автоматического тестирования программного обеспечения на языке Java.

Созданный Кентом Беком и Эриком Гаммой, JUnit принадлежит семье фреймворков xUnit для разных языков программирования, берущей начало в SUnit Кента Бека для Smalltalk. JUnit породил экосистему расширений — JMock, EasyMock, DbUnit, HttpUnit и т. д.

Библиотека **JUnit** была портирована на другие языки, включая PHP (PHPUnit), C# (NUnit), Python (PyUnit), Fortran (fUnit), Delphi (DUnit), Free Pascal (FPCUnit), Perl (Test::Unit), C++ (CPPUnit), Flex (FlexUnit), JavaScript (JSUnit).

JUnit аннотации

`@BeforeEach` - Аннотированный метод будет запускаться перед каждым тестовым методом в тестовом классе.

`@AfterEach` - Аннотированный метод будет запускаться после каждого тестового метода в тестовом классе.

`@BeforeAll` - Аннотированный метод будет запущен перед всеми тестовыми методами в тестовом классе. Этот метод должен быть статическим.

`@AfterAll` - Аннотированный метод будет запущен после всех тестовых методов в тестовом классе. Этот метод должен быть статическим.

`@Test` - Он используется, чтобы пометить метод как тест junit.

`@DisplayName` - Используется для предоставления любого настраиваемого отображаемого имени для тестового класса или тестового метода

`@Disable` - Он используется для отключения или игнорирования тестового класса или тестового метода из набора тестов.

`@Nested` - Используется для создания вложенных тестовых классов

`@Tag` - Пометьте методы тестирования или классы тестов тегами для обнаружения и фильтрации тестов.

`@TestFactory` - Отметить метод - это тестовая фабрика для динамических тестов.

Документация:

https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests

Параметризованные тесты

Параметризованный тест должен выполнять **один и тот же тест снова и снова, используя разные значения**. Это помогает разработчику сэкономить время при выполнении одного и того же теста, который отличается только своими входами и ожидаемыми результатами.

Используя параметризованный тест, можно настроить метод тестирования, который извлекает данные из некоторого источника данных.

```
public class Numbers {
    public static boolean isOdd(int number) {
        return number % 2 != 0;
    }
}
@ParameterizedTest
@ValueSource(ints = {1, 3, 5, -3, 15, Integer.MAX_VALUE}) // six numbers
void isOdd_ShouldReturnTrueForOddNumbers(int number) {
    assertTrue(Numbers.isOdd(number));
}
```

Расширения

```
@ExtendsWith(FileDeleter.class)
public class Numbers {
    public static boolean isOdd(int number) {
        return number % 2 != 0;
    }
}
```

```
public class FileDeleter implements AfterEachCallback {

    @Override
    public void afterEach(ExtensionContext context) throws Exception {
        Files.deleteIfExists(Path.of("data.json"));
    }
}
```

Вопросы?