**@TestInstance в JUnit 5**

13 августа 2020 г. · 4 мин. чтения

**1. Введение**[**​**](https://for-each.dev/lessons/b/-junit-testinstance-annotation#1-%D0%B2%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5)

Тестовые классы часто содержат переменные-члены, относящиеся к тестируемой системе, макетам или ресурсам данных, используемым в тесте. **По умолчанию и JUnit 4, и 5 создают новый экземпляр тестового класса перед запуском каждого тестового метода.**Это обеспечивает четкое разделение состояний между тестами.

В этом руководстве мы узнаем, как [JUnit 5](https://for-each.dev/lessons/b/-junit-5) позволяет нам изменять жизненный цикл тестового класса с помощью аннотации @TestInstance . Мы также увидим, как это может помочь нам в управлении большими ресурсами или более сложными отношениями между тестами.

**2. Жизненный цикл тестирования по умолчанию**[**​**](https://for-each.dev/lessons/b/-junit-testinstance-annotation#2-%D0%B6%D0%B8%D0%B7%D0%BD%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9-%D1%86%D0%B8%D0%BA%D0%BB-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D0%BF%D0%BE-%D1%83%D0%BC%D0%BE%D0%BB%D1%87%D0%B0%D0%BD%D0%B8%D1%8E)

Давайте начнем с рассмотрения жизненного цикла тестового класса по умолчанию, общего для JUnit 4 и 5:

class AdditionTest {  
  
 private int sum = 1;  
  
 @Test  
 void addingTwoReturnsThree() {  
 sum += 2;  
 assertEquals(3, sum);  
 }  
  
 @Test  
 void addingThreeReturnsFour() {  
 sum += 3;  
 assertEquals(4, sum);  
 }  
}

Этот код вполне может быть тестовым кодом JUnit 4 или 5, за исключением отсутствующего ключевого слова public , которое не требуется для JUnit 5.

Эти тесты проходят успешно, потому что перед вызовом каждого тестового метода создается новый экземпляр AdditionTest . Это означает, что значение переменной sum всегда устанавливается в 1 перед выполнением каждого теста.

Если бы был только один общий экземпляр тестового объекта, переменная sum сохраняла бы свое состояние после каждого теста. В результате второй тест провалился.

**3. Аннотации @BeforeClass и @BeforeAll[​](https://for-each.dev/lessons/b/-junit-testinstance-annotation" \l "3--%D0%B0%D0%BD%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D0%B8--beforeclass--%D0%B8--beforeall" \o "Прямая ссылка на 3--аннотации--beforeclass--и--beforeall)**

Бывают случаи, когда нам нужно, чтобы объект существовал в нескольких тестах. Давайте представим, что мы хотели бы прочитать большой файл, чтобы использовать его в качестве тестовых данных. Поскольку повторение этого перед каждым тестом может занять много времени, мы можем предпочесть прочитать его один раз и сохранить для всего тестового набора.

JUnit 4 решает эту проблему с помощью аннотации @BeforeClass :

private static String largeContent;  
  
@BeforeClass  
public static void setUpFixture() {  
 *// read the file and store in 'largeContent'*  
}

Следует отметить, что мы должны сделать переменные и методы, аннотированные **@BeforeClass JUnit 4, статическими.**

**JUnit 5** предлагает другой подход. Он предоставляет аннотацию **@BeforeAll** , которая используется в статической функции для работы со статическими членами класса.

Однако **@BeforeAll** также можно использовать с функцией экземпляра и членами экземпляра, если жизненный цикл тестового экземпляра изменен на per-class .

**4. Аннотация @TestInstance[​](https://for-each.dev/lessons/b/-junit-testinstance-annotation" \l "4--%D0%B0%D0%BD%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D1%8F--testinstance" \o "Прямая ссылка на 4--аннотация--testinstance)**

Аннотация @TestInstance позволяет настроить жизненный цикл тестов JUnit 5.

**@TestInstance имеет два режима:**

-одним из них является LifeCycle.PER\_METHOD (по умолчанию);

- другой — LifeCycle.PER\_CLASS . Последнее позволяет нам попросить JUnit создать только один экземпляр тестового класса и повторно использовать его между тестами.

Давайте аннотируем наш тестовый класс аннотацией @TestInstance и используем режим LifeCycle.PER\_CLASS :

@TestInstance(LifeCycle.PER\_CLASS)  
class TweetSerializerUnitTest {  
  
 private String largeContent;  
  
 @BeforeAll  
 void setUpFixture() {  
 *// read the file*  
 }  
  
}

Как мы видим, ни одна из переменных или функций не является статической. Нам разрешено использовать **метод экземпляра для @BeforeAll , когда мы используем жизненный цикл PER\_CLASS** .

Также следует отметить, что изменения, внесенные в состояние переменных экземпляра одним тестом, теперь будут видны другим.

**4.1 Использование @TestInstance(PER\_CLASS)**[**​**](https://for-each.dev/lessons/b/-junit-testinstance-annotation#5-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5--testinstanceper_class)

**4.1.1 Дорогие ресурсы**[**​**](https://for-each.dev/lessons/b/-junit-testinstance-annotation#51--%D0%B4%D0%BE%D1%80%D0%BE%D0%B3%D0%B8%D0%B5-%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D1%8B)

Эта аннотация полезна, когда создание экземпляра класса перед каждым тестом довольно дорого. Примером может быть установление соединения с базой данных или загрузка большого файла.

Решение этой проблемы ранее приводило к сложному сочетанию статических переменных и переменных экземпляра, что теперь проще с общим экземпляром тестового класса.

**4.1.2. Преднамеренное совместное использование состояния**[**​**](https://for-each.dev/lessons/b/-junit-testinstance-annotation#52--%D0%BF%D1%80%D0%B5%D0%B4%D0%BD%D0%B0%D0%BC%D0%B5%D1%80%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5-%D1%81%D0%BE%D0%B2%D0%BC%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D0%B5-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F)

**Совместное использование состояния обычно является антишаблоном в модульных тестах, но может быть полезным в интеграционных тестах.**Жизненный цикл для каждого класса поддерживает последовательные тесты, которые преднамеренно совместно используют состояние. Это может быть необходимо, чтобы в последующих тестах не приходилось повторять шаги предыдущих тестов, особенно если приведение тестируемой системы в нужное состояние происходит медленно.

При совместном использовании состояния для последовательного выполнения всех тестов JUnit 5 предоставляет нам аннотацию @TestMethodOrder на уровне типа. Затем мы можем использовать аннотацию @Order для тестовых методов, чтобы выполнять их в выбранном нами порядке.

@TestMethodOrder(OrderAnnotation.class)  
class OrderUnitTest {  
  
 @Test  
 @Order(1)  
 void firstTest() {  
 *// ...*  
 }  
  
 @Test  
 @Order(2)  
 void secondTest() {  
 *// ...*  
 }  
  
}

**4.1.3. Совместное использование некоторого состояния**[**​**](https://for-each.dev/lessons/b/-junit-testinstance-annotation#53--%D1%81%D0%BE%D0%B2%D0%BC%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D0%B5-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BD%D0%B5%D0%BA%D0%BE%D1%82%D0%BE%D1%80%D0%BE%D0%B3%D0%BE-%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F)

Проблема с совместным использованием одного и того же экземпляра тестового класса заключается в том, что некоторые члены могут нуждаться в очистке между тестами, а некоторые — поддерживаться на протяжении всего теста.

**Мы можем сбросить переменные, которые необходимо очищать между тестами, с помощью методов, аннотированных с помощью @BeforeEach или @AfterEach .**