JUnit 5

0

What's AUTOMATION TESTING?

JUnit 5 tutorial

#0: About the tutorial

/RaminZare

About the tutorial

- What is JUnit framework about?
- Who needs this?

What do we mean, when we say software testing?

History

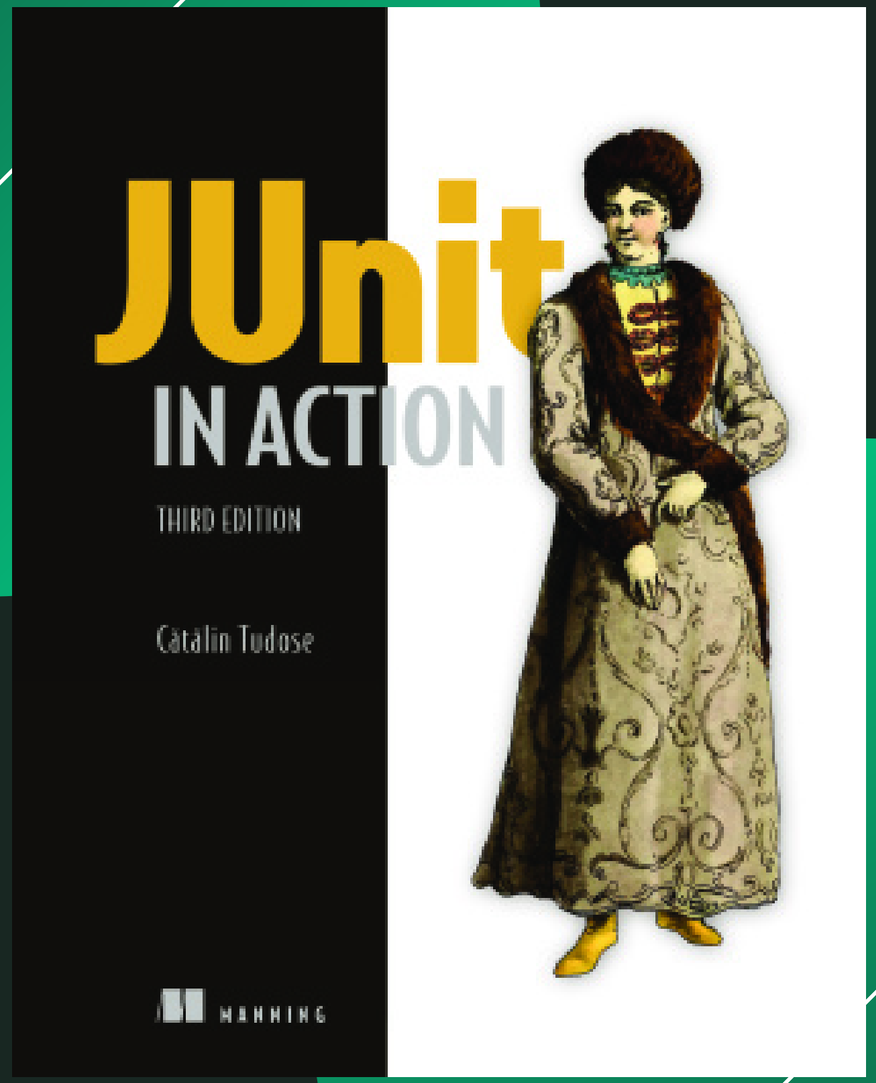
JUnit (1997)

authors

Erich Gamma

Kent Beck

Hello JUnit world



REFERENCE

JUnit in action

Third edition

Publication date: 2020

https://www.manning.com/books/junit-in-action-third-edition

JUnit 5

1

Setting up JUnit

JUnit 5 tutorial

#1: Setting up JUnit

/RaminZare

IDEs

Intellij IDEA

Eclipse

Build tools

Maven

Gradle

JUnit 5

2

Writing the first unit test

JUnit 5 tutorial

#2: Writing the first unit test

Core testing elements

/RaminZare

We decided that we need a reliable, repeatable way to test programs

Core testing elements

Test classes and lifecycle methods

Test Methods

Assertions

Test classes

- **Classifies test cases**
- Minimum visibility is package-private
- Cannot be abstract
- Must have a single constructor
- Can be nested

Lifecycle methods

Methods with annotations

- @BeforeAll & @AfterAll
  - run once before and after each instantiation
  - must be static
- @BeforeEach & @AfterEach
  - run before each test case

Test methods

- Must have an annotation
  - **@Test**
  - or other types of tests...
- Must be void
- Minimum visibility is package-private
- Cannot be abstract

Assertions

To perform test validation, you use the assert methods provided by the JUnit Assertions class

- `assertTrue`
- `assertEquals`
- `assertArrayEquals`
- `assertTimeout`
- `assertThrows`
- `assertAll`

JUnit 5

3

Nested test classes

JUnit 5 tutorial

#3: Nested Test classes

/RaminZare

Test classes

- **UPPER CLASS**
- **STATIC MEMBER CLASS**
- **INNER CLASS**

  Adding @Nested annotation on top of an inner class

JUnit 5

Lifecycle of test classes

4

JUnit 5 tutorial

#4: The lifecycle of test classes

/RaminZare

Construction

PER METHOD

The default mode

PER CLASS

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
```

Life cycle

- `@BeforeEach`
  `@AfterEach`
  Before and After each method

- `@BeforeAll`
  `@AfterAll`
  Before and after all methods

JUnit 5

5

Annotation:

@DisplayName

JUnit 5 tutorial

#5: Better test reporting with `@DisplayName` & `@DisplayNameGeneration`

/RaminZare

JUnit 5

Annotation: `@Disabled`

6

JUnit 5 tutorial

#6: Disable a test class or method

/RaminZare

JUnit 5

More about Assertions

7

JUnit 5 tutorial

#7: More about Assertions

/RaminZare

Assertion methods

assertThrows()

assertTrue()

assertNull()

assertArrayEquals()

assertAll()

JUnit 5

Dependency Injection in JUnit

8

JUnit 5 tutorial

#8: Dependency injection in JUnit 5

/RaminZare

JUnit 5 allows test constructors and methods to have parameters, but they need to be resolved through dependency injection.

ParameterResolver

ParameterResolver defines the API for Extensions that wish to dynamically resolve arguments for parameters at runtime.

@ExtendWith is a repeatable annotation that is used to register extensions

JUnit 5

Reusable parameter resolvers

9

JUnit 5 tutorial

#9: Reusable parameter resolvers in JUnit

/RaminZare

ParameterResolver

- TestInfoParameterResolver
- TempDirectory
- RepetitionInfoParameterResolver
- TestReporterParameterResolver

JUnit 5

Repeated tests

10

JUnit 5 tutorial

#10: Repeated test

/RaminZare

Repeated tests

- Repeating a test, a specific number
- Useful when conditions may change

```
@RepeatedTest(5)
void testMethod(){
    //...
}
```

Repeated tests

```
@RepeatedTest(value = 5 , name = "{displayName}-
repetition{currentRepetition}/{totalRepetitions}")
```

Placeholders:

- {displayName}
- {currentRepetition}
- {totalRepetitions}

RepetitionInfoParameterResolver

```
@RepeatedTest(5)
void testMethod(RepetitionInfo info){
    info.getCurrentRepetition();
    info.getTotalRepetitions();
}
```

JUnit 5

11

Parameterized tests

JUnit 5 tutorial

#11: Parameterized tests

/RaminZare

Allows a test to run multiple times with different arguments

@ParameterizedTest

```
@ParameterizedTest
@ValueSource(strings = {"/url1","/url2"})
void testMethod(String url){
    //...
}
```

Sources

@ValueSource

Single array of literal values

@EnumSource

Using an enum instances as parameters

@CsvSource

String arguments as comma-separated values (CSV)

@CsvFileSource

Using a CSV file from the classpath as parameters

JUnit 5

12

Tagged tests

JUnit 5 tutorial

#12: Tagged tests

/RaminZare

You can use the @Tag annotation over classes and test methods. Later, you can use tags to filter test discovery and execution

@Tag

```
@Tag("integration")
class TestTwoComponents{

   @Tag("performanceTest")
  void checkPerformance(){
   //...
  }

}
```

JUnit 5

13

Assumptions

JUnit 5 tutorial

#13: Assumptions

/RaminZare

We can prevent our tests from being executed under inappropriate conditions

Assumptions

- Assumptions
  - **`assumeTrue`**`()`/**`assumeFalse`**`()`
    - The test will not run unless the assumption is true/false
- When it does not make sense to continue the execution of a given test method

Assumptions

- **`assumingThat()`**
  - executes an assertion only if the assumption is fulfilled
  - works only for one expression

JUnit 5

14

Dynamic tests with @TestFactory

JUnit 5 tutorial

#14: Dynamic tests with @TestFactory

/RaminZare

Test factory is a dynamic new programming model that can generate tests at runtime

@TestFactory

FACTORY METHOD

A @TestFactory method is a factory that generates tests that supports parameter resolver

RETURN OBJECT

One or an array or a collection/Stream/Iterable of DynamicNode

VISIBILITY

at least package-private

DIFFERENT LIFECYCLE

@BeforeEach and @AfterEach only calls once before and after the whole test factory

```java
@TestFactory
Iterator<DynamicTest> positiveNumberPredicateTestCases() {
    return asList(
            dynamicTest("negative number",
                    () -> assertFalse(predicate.check(-1))),
            dynamicTest("zero",
                    () -> assertFalse(predicate.check(0))),
            dynamicTest("positive number",
                    () -> assertTrue(predicate.check(1)))
    ).iterator();
}
```

JUnit 5

Replacing Assertions with Hamcrest

15

JUnit 5 tutorial

#15: Replacing Assertions with Hamcrest

/RaminZare

Some assertions are big and hard to read. Hamcrest is a library to write more readable assertions

```java
@Test
@DisplayName("List without Hamcrest")
public void testWithoutHamcrest() {
    assertEquals(3, values.size());
    assertTrue(values.contains("Oliver")
            || values.contains("Jack")
            || values.contains("Harry"));
}
@Test
@DisplayName("List with Hamcrest")
public void testListWithHamcrest() {
    assertThat(values, hasSize(3));
    assertThat(values, hasItem(anyOf(equalTo("Oliver"),
            equalTo("Jack"), equalTo("Harry"))));
}
```

JUNIT5

```
assertEquals(expected, actual);
assertNotEquals(expected, actual)
```

HAMCREST

```
assertThat(actual, is(equalTo(expected)));
assertThat(actual, is(not(equalTo(expected))));
```