Annotations in Java & Unit Testing with JUnit

Advanced Programming Course – Spring 2021

CE@AUT

Annotations

- An annotation is a form of syntactic metadata that can be added to Java source code.
  - Annotations are meta-meta-objects which can be used to describe other meta-objects. Meta-objects are classes, fields and methods.
  - Annotations provide data about a program that is not part of the program itself.

- Annotations can be interpreted at development-time by the IDE or the compiler, or at run-time by a framework.

Annotations - cont.

- Annotations start with ‘@’.
- Annotations do not change action of a compiled program.
- Annotations help to associate metadata (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.
- Annotations are not pure comments as they can change the way a program is treated by compiler.

Annotations - cont.

- Some annotations applied to Java code:
  - **@Override** - Checks that the method is an override. Causes a compile error if the method is not found in one of the parent classes or implemented interfaces.
  - **@Deprecated** - Marks the method as obsolete. Causes a compile warning if the method is used.
  - **@SuppressWarnings** - Instructs the compiler to suppress the compile time warnings specified in the annotation parameters.

Annotations - cont.

- Annotation processing is a very powerful mechanism and can be used in a lot of different ways:
  - to describe constraints or usage of an element: e.g. @Deprecated, @Override, or @NotNull
  - to describe the "nature" of an element, e.g. @Entity, @TestCase, @WebService
  - to describe the behavior of an element: @Statefull, @Transaction
  - to describe how to process the element: @Column, @XmlElement
  - In all cases, an annotation is used to <u>describe the element and clarify its meaning</u>.
- Annotations are customizable.

Annotations - cont.

- Usage of annotations:
  - Documentation, e.g. XDoclet
  - Compilation
  - IDE
  - Testing framework, e.g. JUnit
  - IoC container e.g. as Spring
  - Serialization, e.g. XML
  - Aspect-oriented programming (AOP), e.g. Spring AOP
  - Application servers, e.g. EJB container, Web Service
  - Object-relational mapping (ORM), e.g. Hibernate, JPA
  - and many more...

Unit Testing

- Test of individual parts of an application
  - Opposed to *application testing*
- e.g. a single class, a single method
- Any single method, once written and compiled, can and <u>should</u> be tested.
- Manual testing: time consuming and boring! ☹
- Recall "Regression Testing"
  - We need Automated Testing
  - For Java: JUnit, TestNG, Mockito, Spock, Arquillian, …

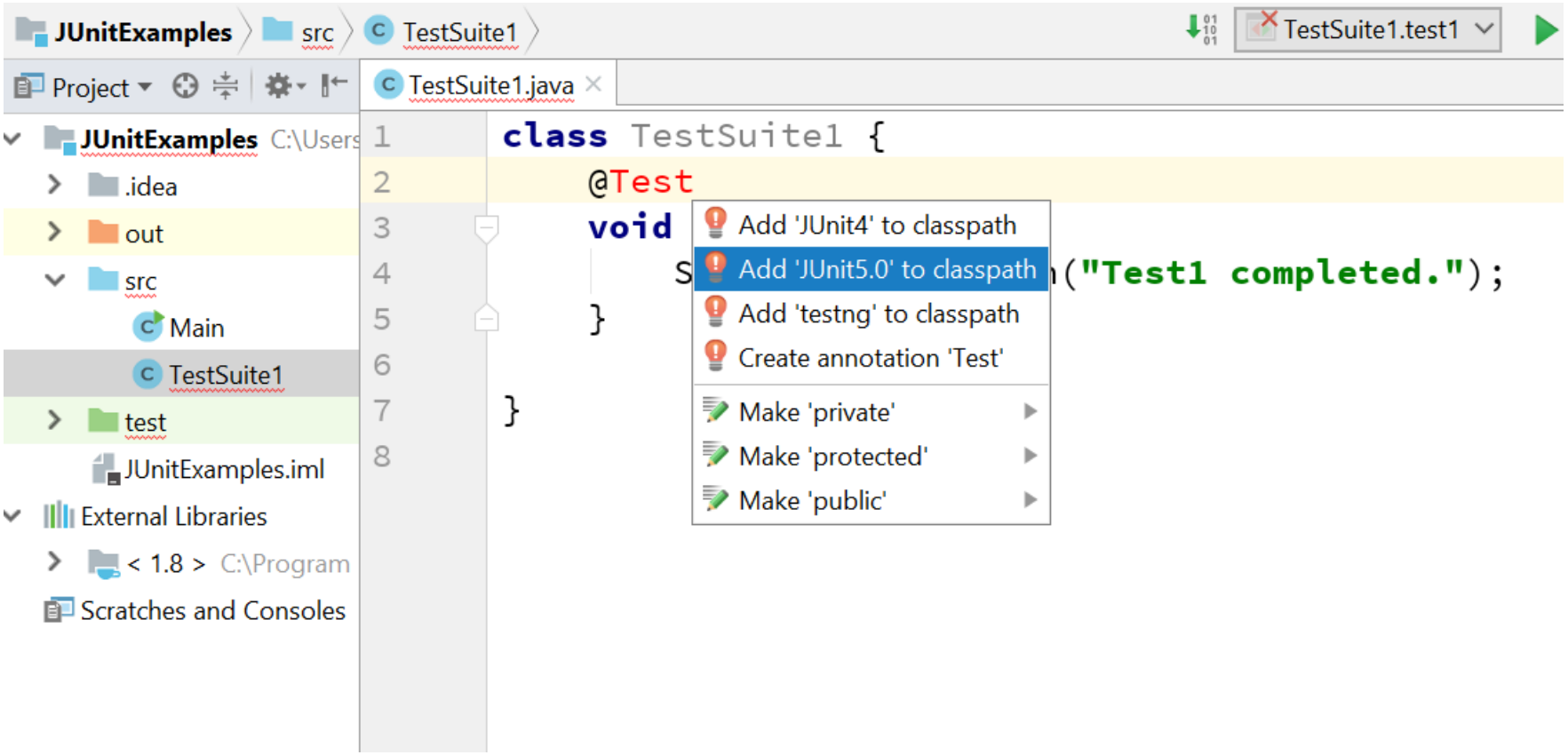
A Quote

Edsger Dijkstra

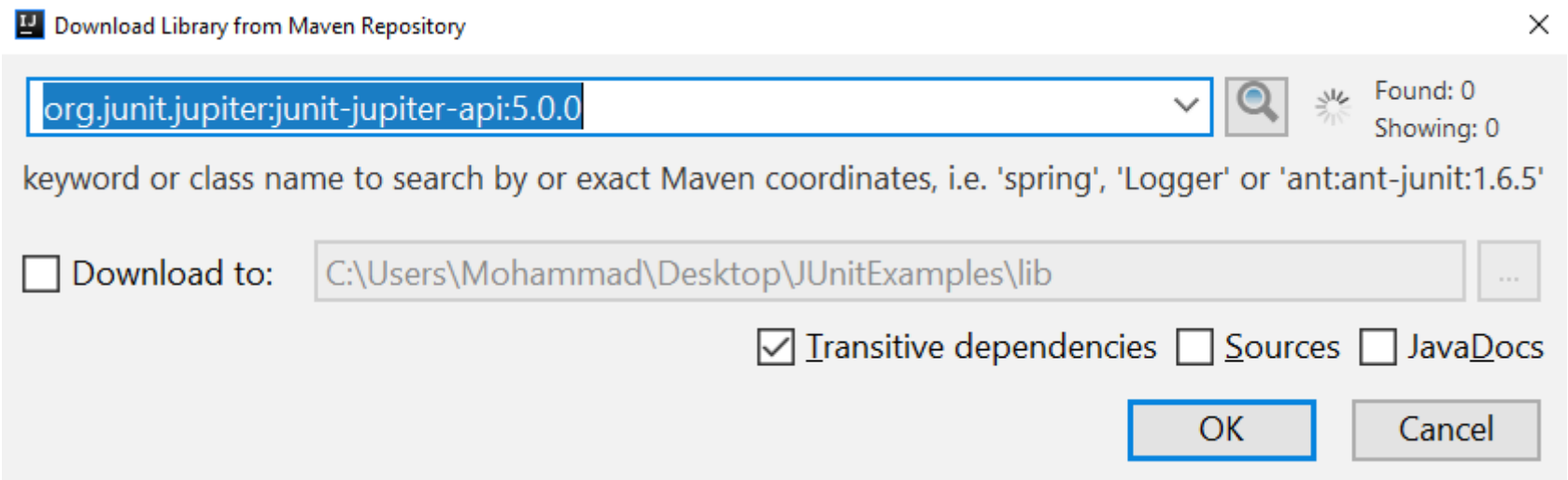
JUnit

- A unit testing framework for Java
- Important in Test-Driven Development (TDD)
- Stable release: 5.7.1 / February 4, 2021
  - For Java 8 and 9
- JUnit is linked as a JAR at compile-time
  - under package *org.junit* for JUnit 4 and later

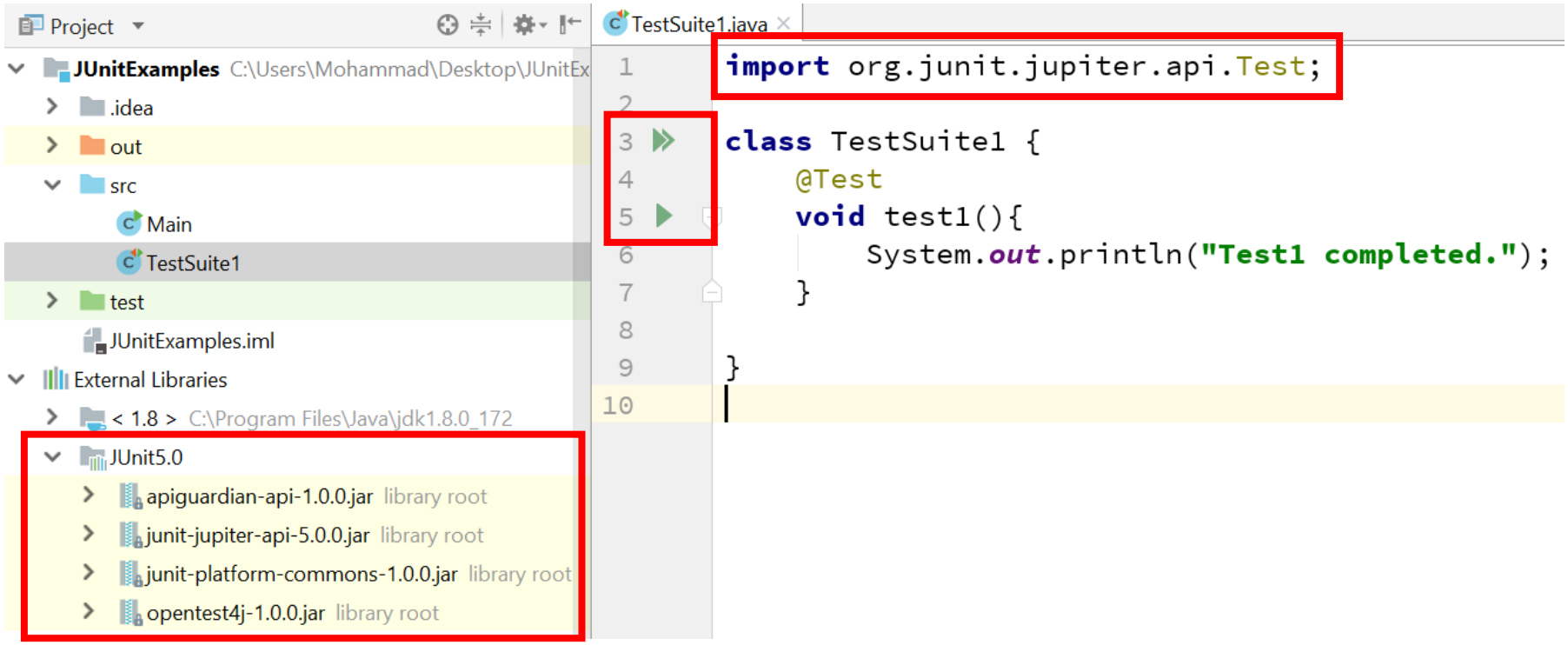
How to use JUnit in IntelliJ (The Simplest Way)



How to use JUnit in IntelliJ (The Simplest Way) – cont.



How to use JUnit in IntelliJ (The Simplest Way) – cont.



How to use JUnit in IntelliJ (Using Maven)

- you have to start by adding the *junit-jupiter-engine* dependency to your project's classpath
- using *Maven*, you can simply add the following to your *pom.xml*:

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
     <artifactId>junit-jupiter-engine</artifactId>
     <version>5.0.0-M4</version>
</dependency>
```

How to use JUnit in IntelliJ (Using Maven) – cont.

- to run the tests is by using the Maven Surefire plugin:

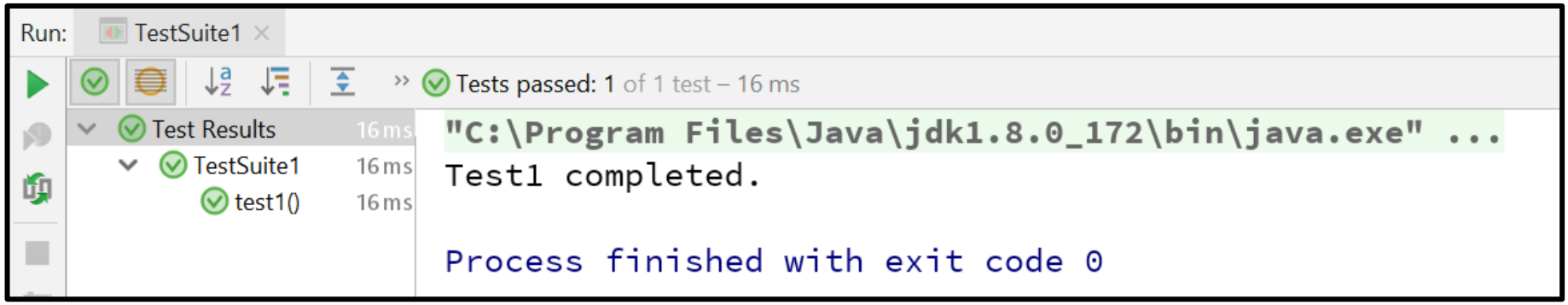
```
<plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.20</version>
    <configuration>
         <include>**/Test*.java</include>
    </configuration>
    <dependencies>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-surefire-provider</artifactId>
            <version>1.0.0-M4</version>
        </dependency>
    </dependencies>
</plugin>
```

- tests will run with the standard "mvn clean install" command

Example 1

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

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

class TestSuite1 {

    @Test
    void test1() {
        assertEquals(“hello", “hel"+“lo");
        System.out.println("Test1
completed.");

    }
}
```

Example 1

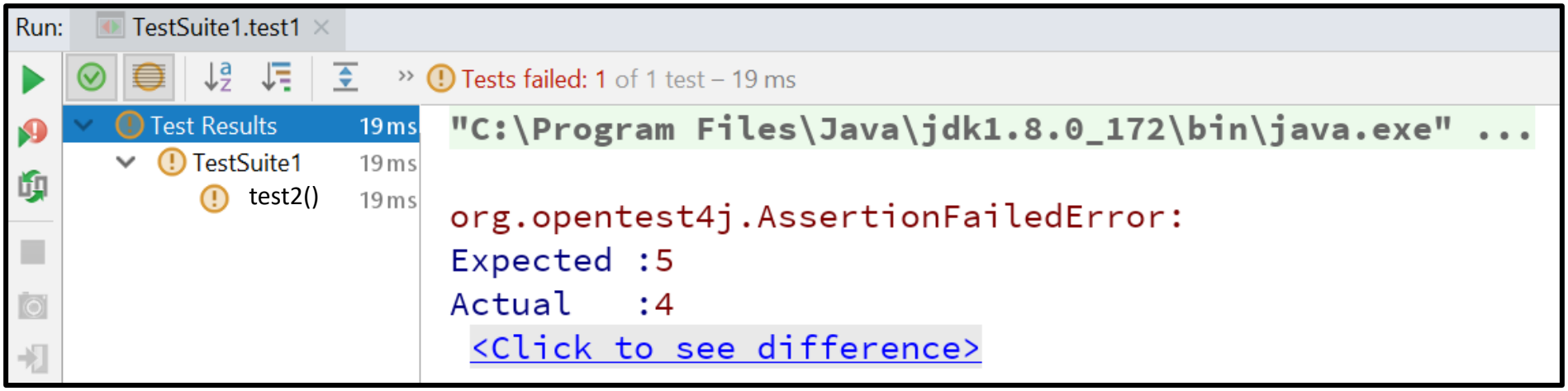



```
class TestSuite1 {

    @Test
    void test1() {
        assertEquals("hello", "hel"+"lo");
        System.out.println("Test1
completed.");

    }
}
```

Example 2

```java
import org.junit.jupiter.*;
import org.junit.jupiter.api.*;

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

class TestSuite2 {

    @Test
    void test2() {
        assertEquals(5, 2*2);
        System.out.println("Test2 completed.");

    }
}
```

Example 2




```
    @Test
    void test2() {
        assertEquals(5, 2*2);
        System.out.println("Test2 completed.");

    }
}
```

Annotations

Annotation	Description
@Test	Denotes that a method is a test method.
@RepeatedTest(<Number>)	Denotes that a method is a test template for a repeated test.
@BeforeEach	Denotes that the annotated method should be executed *before* **each** @Test and @RepeatedTest methods in the current class;
@AfterEach	Denotes that the annotated method should be executed *after* **each** @Test and @RepeatedTest methods in the current class;
@BeforeAll	Denotes that the annotated static method should be executed *before* **all** @Test and @RepeatedTest methods in the current class;
@AfterAll	Denotes that the annotated static method should be executed *after* **all** @Test and @RepeatedTest methods in the current class;
@Nested	Denotes that the annotated class is a nested, non-static test class.
@Disabled	Used to *disable* a test class or test method; (JUnit 4: @Ignore)
@DisplayName(“<Name>")	<Name> that will be displayed by the test runner. In contrast to method names the DisplayName can contain spaces.

Example 3

```
class TestSuite1 {
    @Test
    void test1(){
        assertEquals(2, 1+1);

System.out.println("Test1
completed.");
    }

    @Test
    @DisplayName(“Second One")
    void test2(){

System.out.println("Test2
completed.");
    }

    @BeforeEach
    void testBeforeEach(){

System.out.println("Before
Each!");
    }

    @AfterEach
        void testAfterEach(){

System.out.println("After
Each!");
        }
        @BeforeAll
        static void
testBeforeAll(){

System.out.println("Before
All!");
        }


        @AfterAll
        static void testAfterAll(){

System.out.println("After
All!");
        }
}
```

Example 3 - Output

Run: TestSuite1

Tests passed: 2 of 2 tests – 16 ms

Test Results	16 ms
TestSuite1	16 ms
test1()	15 ms
Second One	1 ms

```
"C:\Program Files\Java\jdk1.8.0_172\bin\java.exe"
Before All!
Before Each!
Test1 completed.
After Each!
Before Each!
Test2 completed.
After Each!
After All!
Process finished with exit code 0
```

Assertions

Assertion	Description
assertTrue [assertFalse]	to verify that a boolean value is true [false]
assertNull [assertNotNull]	to verify that an object is [not] null
assertEquals [assertNotEquals]	to verify that the expected value (or object) is [not] equal to the actual value (or object)
assertSame [assertNotSame]	to ensure that two objects [do not] refer to the same object
assertArrayEquals	to verify that two arrays are equal
assertThrows	to write assertions for the exceptions thrown by the system under test
assertTimeout/assertTimeoutPreemptively	to ensure that the execution of the system under test is completed before a specified timeout is exceeded
assertAll()	to write an assertion for a state that requires multiple assertions

Example 4

```
class UserTest {

    private static ArrayList<User> userList = new
ArrayList<>();

    private static User findOne(ArrayList<User> array,
String email) {
        Iterator<User> it = array.iterator();
        while (it.hasNext()) {
            User temp = it.next();
            if (temp.getEmail().equals(email)) {
                return temp;
            }
        }
        return null;
    }
```

Example 4 – cont.

```
@BeforeAll
static void addData() {
    User user1 = new User("john@gmail.com", "John");
    User user2 = new User("ana@gmail.com", "Ana");
    userList.add(user1);
    userList.add(user2);
    System.out.println("John and Anna Added.");
}
@AfterAll
static void removeData() {
    userList.removeAll(userList);
    System.out.println(userList.size());
    System.out.println("userList deleted.");
}
```

Example 4 – cont.

```
"C:\Program Files\Java\jdk1.8.0_172\bin\java.exe" ...
John and Anna Added.
```

```
@BeforeAll
static void addData() {
    User user1 = new User("john@gmail.com", "John");
    User user2 = new User("ana@gmail.com", "Ana");
    userList.add(user1);
    userList.add(user2);
    System.out.println("John and Anna Added.");
}
@AfterAll
static void removeData() {
    userList.removeAll(userList);
    System.out.println(userList.size());
    System.out.println("userList deleted.");
}
```

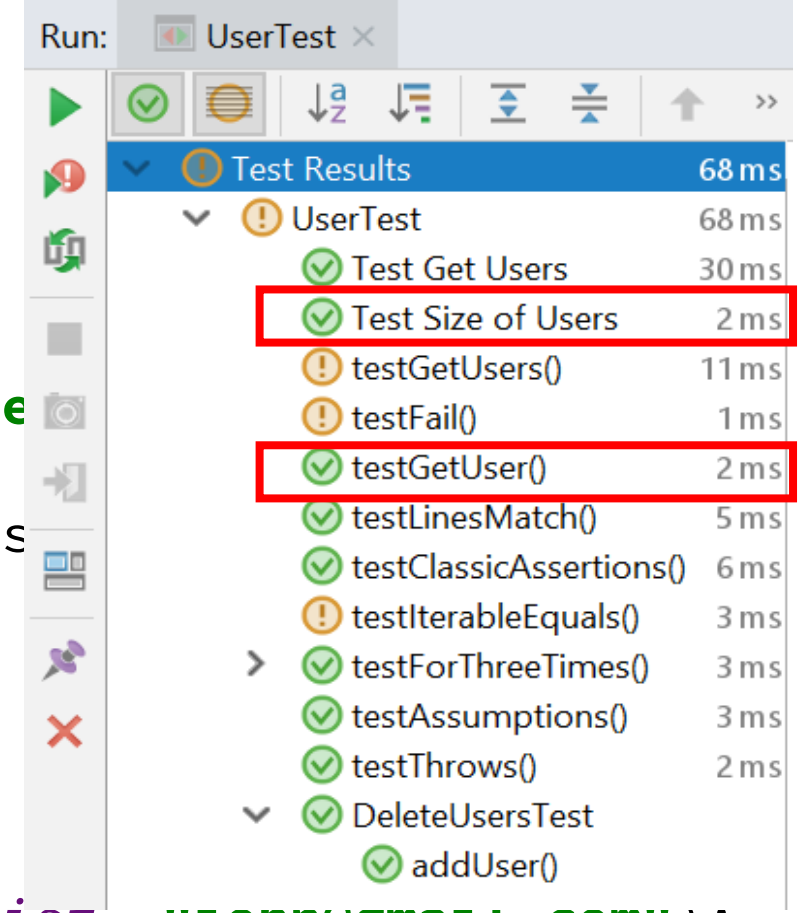
```
0
userList deleted.
Process finished with exit code -1
```

Example 4 – cont

```
@Test
@DisplayName("Test Size of Users")
void testSizeOfUsers() {
    assertEquals(2, userList.size());
}


@Test
void testGetUser() {
    User user = findOne(userList, "john@gmail.com");

    assertNotNull(user);
    assertEquals("John", user.getName(),
            "User name:" + user.getName() + "
incorrect");
}
```

Example 4 – cont

```
@Test
@DisplayName("Test Size of Use
void testSizeOfUsers() {
    assertEquals(2, userList.s
}


@Test
void testGetUser() {
    User user = findOne(userList, "jonn@gmail.com");

    assertNotNull(user);
    assertEquals("John", user.getName(),
            "User name:" + user.getName() + "
incorrect");
}
```

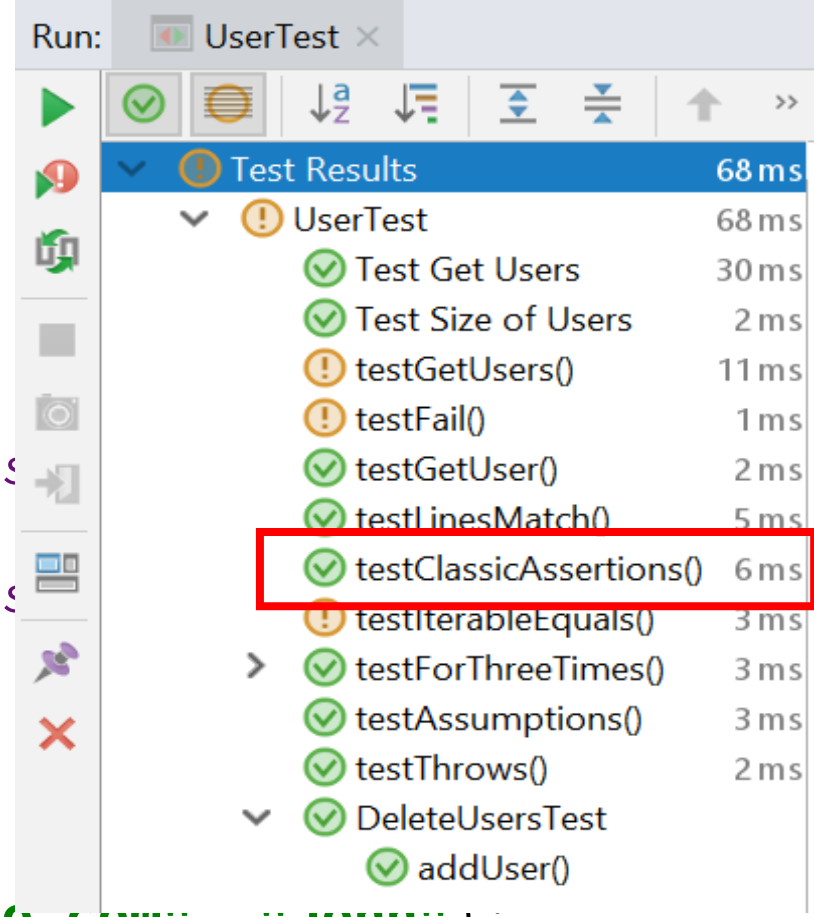


Example 4 – cont.

```
@Test
void testClassicAssertions() {
    User user1 = findOne(userList,
           "john@gmail.com");
    User user2 = findOne(userList,
            "john@yahoo.com");
    assertNotNull(user1);
    assertNull(user2);

    user2 = new User("john@yahoo.com", "John");
    assertEquals(user1.getName(), user2.getName(),
"Names are not equal");

assertFalse(user1.getEmail().equals(user2.getEmail()),
"Emails are equal");
    assertNotSame(user1, user2);
}
```

Example 4 – cont.

```
@Test
void testClassicAssertions() {
    User user1 = findOne(userLis
            "john@gmail.com");
    User user2 = findOne(userLis
            "john@yahoo.com");
    assertNotNull(user1);
    assertNull(user2);

    user2 = new User("john@yahoo.com", "John");
    assertEquals(user1.getName(), user2.getName(),
"Names are not equal");

assertFalse(user1.getEmail().equals(user2.getEmail()),
"Emails are equal");
    assertNotSame(user1, user2);
}
```

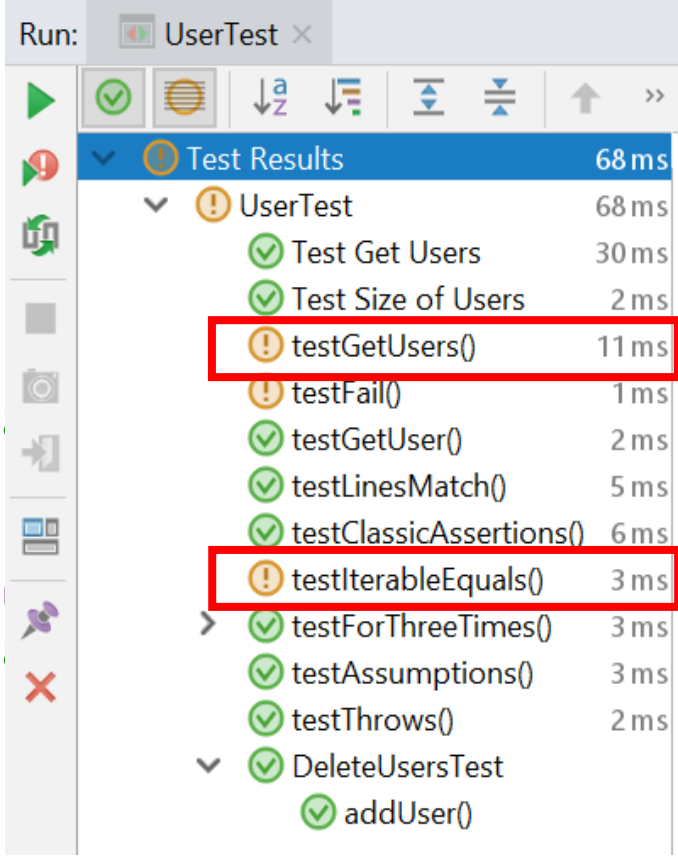


Example 4 – cont.

```
@Test
void testGetUsers() {
    User user = findOne(userList, "john@gmail.com");

    assertAll("user",
            () -> assertEquals("Johnson", user.getName()),
            () -> assertEquals("johnson@gmail.com",
                        user.getEmail()));
}

@Test
void testIterableEquals() {
    User user1 = new User("john@gmail.com", "John");
    User user2 = new User("ana@gmail.com", "Ana");

    List<User> users = new ArrayList<>();
    users.add(user1);
    users.add(user2);
    assertIterableEquals(users, userList);
}
```

Example 4 – cont.

```
@Test
void testGetUsers() {
    User user = findOne(userList, "john@gm

    assertAll("user",
            () -> assertEquals("Johnson",
            () -> assertEquals("johnson@gm
                        user.getEmail())
}
```




```
@Test
void testIterableEquals() {
    User user1 = new User("john@gmail.com", "John");
    Us
    Li
    us
    us
    as
}
```

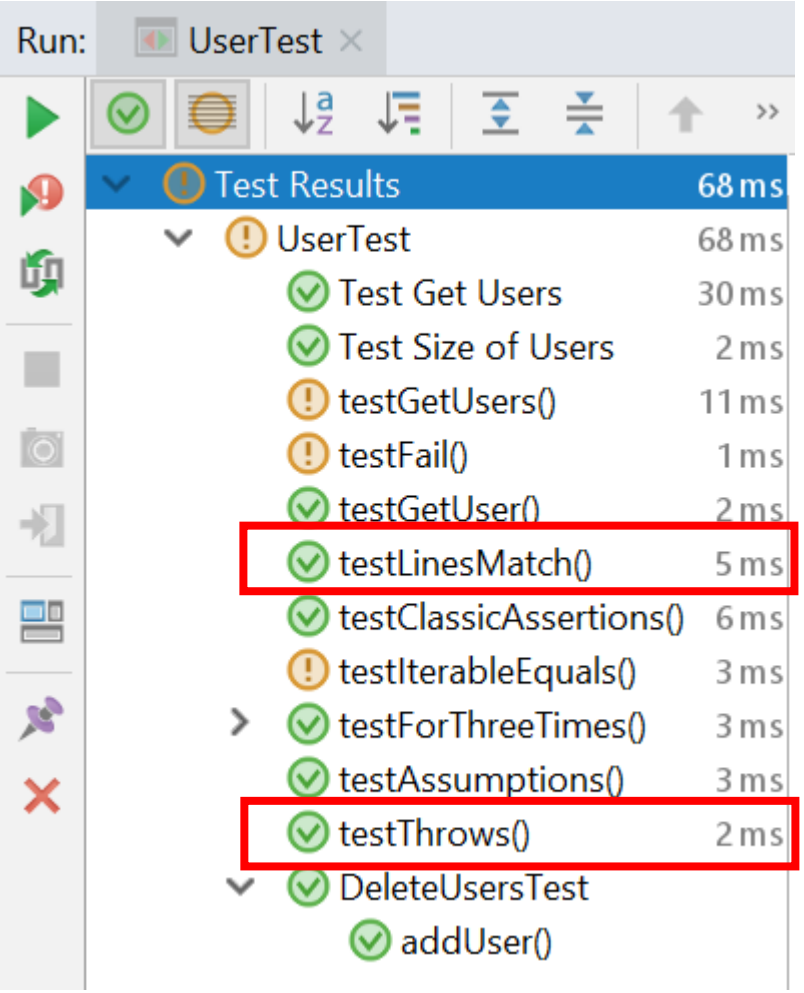
```
org.opentest4j.AssertionFailedError: iterable contents differ at index [0],
Expected :<User@8bd1b6a>
Actual   :<User@18be83e4>
 <Click to see difference>

 <8 internal calls>
    at UserTest.testIterableEquals(UserTest.java:89) <15 internal calls>
    at java.util.stream.ForEachOps$ForEachOp$OfRef.accept(ForEachOps.java:184)
    at java.util.Iterator.forEachRemaining(Iterator.java:116) <3 internal calls
```

Example 4 – cont.

```
@Test
void testLinesMatch() {
    List<String> expectedLines =
    Collections.singletonList("(.*)@(.*)");
    List<String> emails =
          Arrays.asList("john@gmail.com");
    assertLinesMatch(expectedLines,
          emails);
}

@Test
void testThrows() {
    User user = null;
    Exception exception =
assertThrows(NullPointerException.class, () ->
user.getName());
    System.out.println(exception.getMessage());
}
```

Example 4 – cont.

```
@Test
void testLinesMatch() {
    List<String> expectedLines =
    Collections.singletonList("(
    List<String> emails =
            Arrays.asList("john@gm
    assertLinesMatch(expectedLir
            emails);
}

@Test
void testThrows() {
    User user = null;
    Exception exception =
assertThrows(NullPointerException.class, () ->
user.getName());
    System.out.println(exception.getMessage());
}
```



Example 4 – cont.

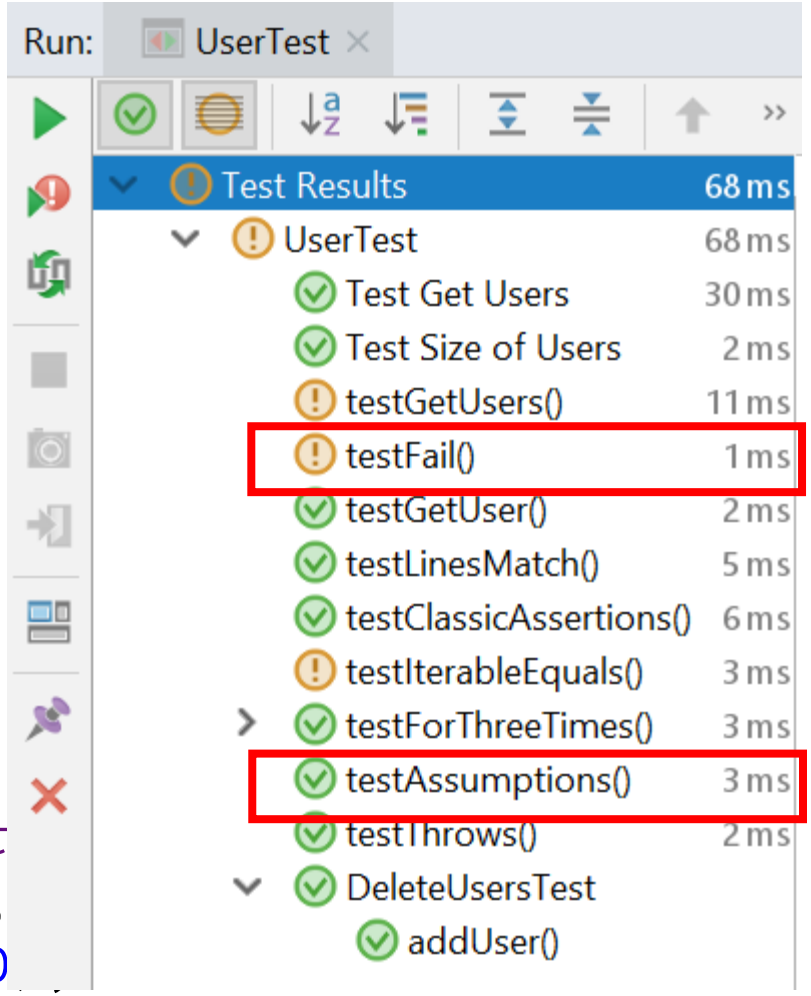
```
@Test
void testFail() {
    fail("this test fails");
}

@Test
void testAssumptions() {
    List<User> users = userList;
    assumeFalse(users == null);
    assumeTrue(users.size() > 0);

    User user1 = new User("john@gmail.com", "John");
    assumingThat(users.contains(user1), () ->
assertTrue(users.size() > 1));
}
```

Example 4 – cont.

```
@Test
void testFail() {
    fail("this test fails");
}

@Test
void testAssumptions() {
    List<User> users = userList;
    assumeFalse(users == null);
    assumeTrue(users.size() > 0);

    User user1 = new User("john@gmail.com", "John");
    assumingThat(users.contains(user1), () ->
assertTrue(users.size() > 1));
}
```

Example 4 – cont.

```
@Test
void testFail() {
    fail("this test fails");
}

@Test
void testAssumptions() {
    List<User> users = userList
    assumeFalse(users == null);
    assumeTrue(users.size() > 0

    User user1 = new User("john@gmail.com", "John");
```



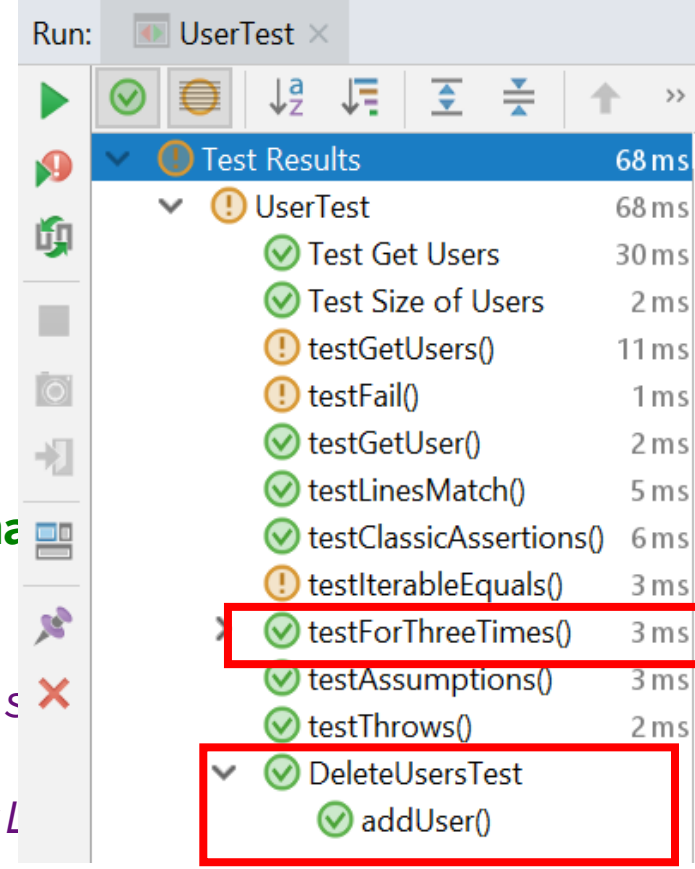

```
org.opentest4j.AssertionFailedError: this test fails
 <3 internal calls>
    at UserTest.testFail(UserTest.java:108) <15 internal calls>
    at java.util.stream.ForEachOps$ForEachOp$OfRef.accept(ForEachOps.java:184)
    at java.util.Iterator.forEachRemaining(Iterator.java:116) <3 internal call
```

Example 4 – cont.

```
@Nested
class DeleteUsersTest {
    @Test
    void addUser() {
        User user = new User("bob@gmail.com",
              "Bob");
        userList.add(user);
        assertNotNull(findOne(userList,
               "bob@gmail.com"));
        userList.remove(findOne(userList,
               "bob@gmail.com"));
        assertNull(findOne(userList,"bob@gmail.com"));
    }
}
@RepeatedTest(3)
void testForThreeTimes() {
    assertTrue(1 == 1);
    System.out.println("Repeated Test");
}
```

Example 4 – cont.

```
@Nested
class DeleteUsersTest {
    @Test
    void addUser() {
        User user = new User("bob@gma
                "Bob");
        userList.add(user);
        assertNotNull(findOne(userLis
                "bob@gmail.com"));
        userList.remove(findOne(userL
                "bob@gmail.com"));
        assertNull(findOne(userList,"bob@gmail.com"));
    }
}
@RepeatedTest(3)
void testForThreeTimes() {
    assertTrue(1 == 1);
    System.out.println("Repeated Test");
}
```



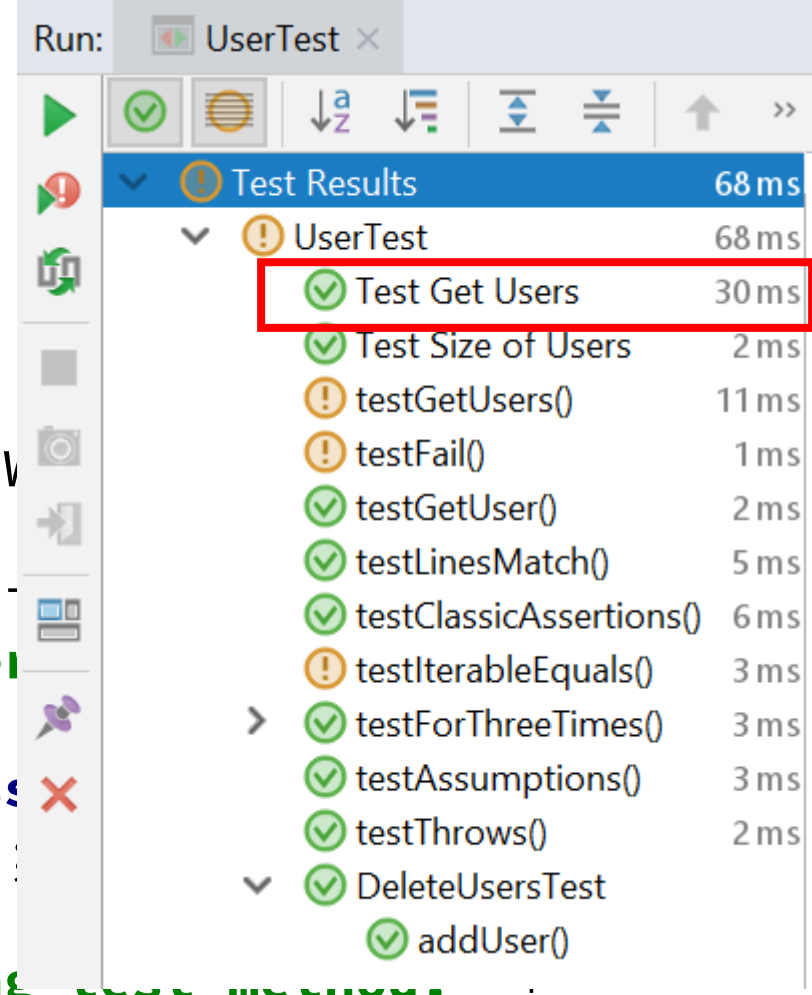

```
Repeated Test
Repeated Test
Repeated Test
```

Example 4 – cont.

```
@Test
@DisplayName("Test Get Users")
public void testGetUsersNumberWithInfo(TestInfo
testInfo) {
    assertEquals(2, userList.size());
    assertEquals("Test Get Users",
testInfo.getDisplayName());
    assertEquals(UserTest.class,
testInfo.getTestClass().get());

    System.out.println("Running test method:" +
testInfo.getTestMethod().get().getName());
}
```

Example 4 – cont.

```
@Test
@DisplayName("Test Get Users")
public void testGetUsersNumberW
testInfo) {
    assertEquals(2, userList.s
    assertEquals("Test Get Use
testInfo.getDisplayName());
    assertEquals(UserTest.class
testInfo.getTestClass().get());

    System.out.println("Running test method:
testInfo.getTestMethod().get().getName());
}
```



Stopped. Tests failed: 3, passed: 13 of 14 tests – 68 ms

```
Running test method:testGetUsersNumberWithInfo
```