UNIT TESTING IN JAVA: JUNIT

CSC 207 SOFTWARE DESIGN

Computer Science
UNIVERSITY OF TORONTO

LEARNING OUTCOMES

Be able to write unit tests in Java using Junit

FIRST YEAR TESTING

Mostly focused on unit testing: when you call a function, does it behave correctly?

Test case design:

How do we pick test cases?

SELECTING TEST CASES

- Test for success
    - General cases, well-formatted input, boundary cases
    - Classics:
        - 0, 1, more
        - odd, even
        - beginning, middle, end
- Check for data structure consistency (representation invariants)
- Test for atypical behaviour
    - Does it handle invalid input (if required)?
    - Does it throw the exceptions it is supposed to?

UNIT TESTING

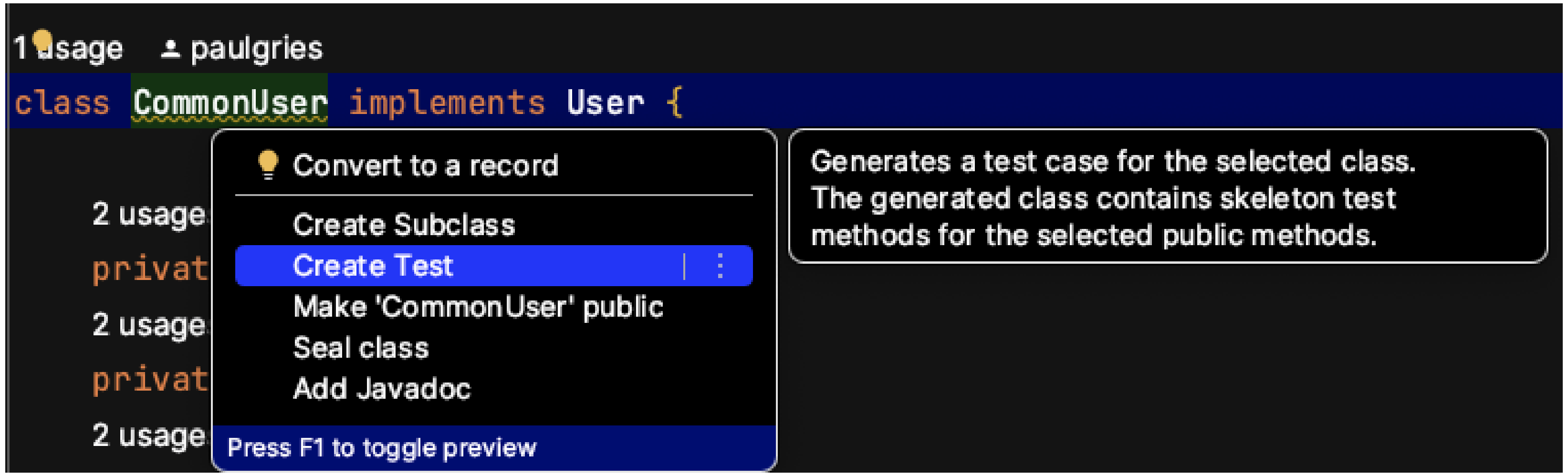
- Unit testing follows a pattern
    - Lots of small, **independent** tests
    - Reports passes, failures, and errors
    - Some optional setup and teardown shared across tests
    - Aggregation (combine tests into test suites)
- We could accomplish all of this “by hand”, but this common structure inspired the development of JUnit:
    - When you see a pattern, build a framework
    - Write shared code once
    - Make it easy for people to do things the right way

JUNIT

- Like pytest or unittest from Python
- You may encounter JUnit4 or JUnit5 when programming; they have very slight differences in syntax.
- We encourage you to explore https://github.com/junit-team/junit5 and the documentation on Junit5 linked from there to learn more as you start writing tests.

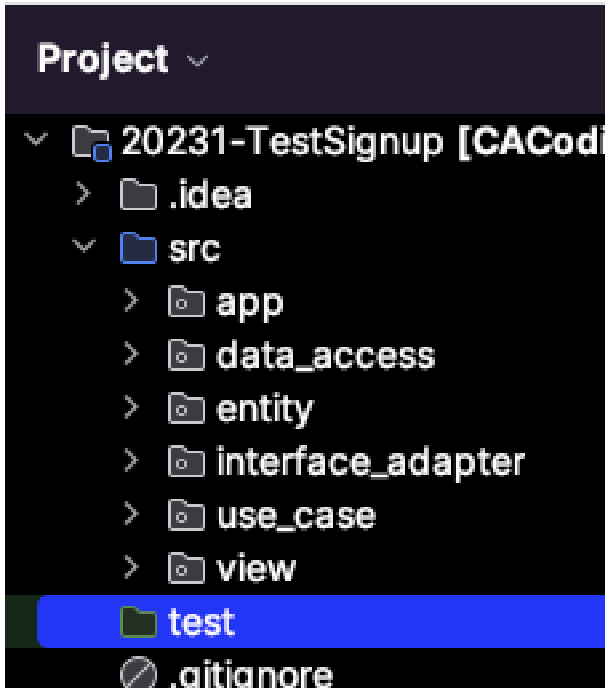
GENERATE JUNIT TESTS IN INTELLIJ

1. Create a test directory at the root of your project
   - Mark it as Test Sources Root
   - It will soon have the same package hierarchy as src
2. Select a class to test and choose Create Test

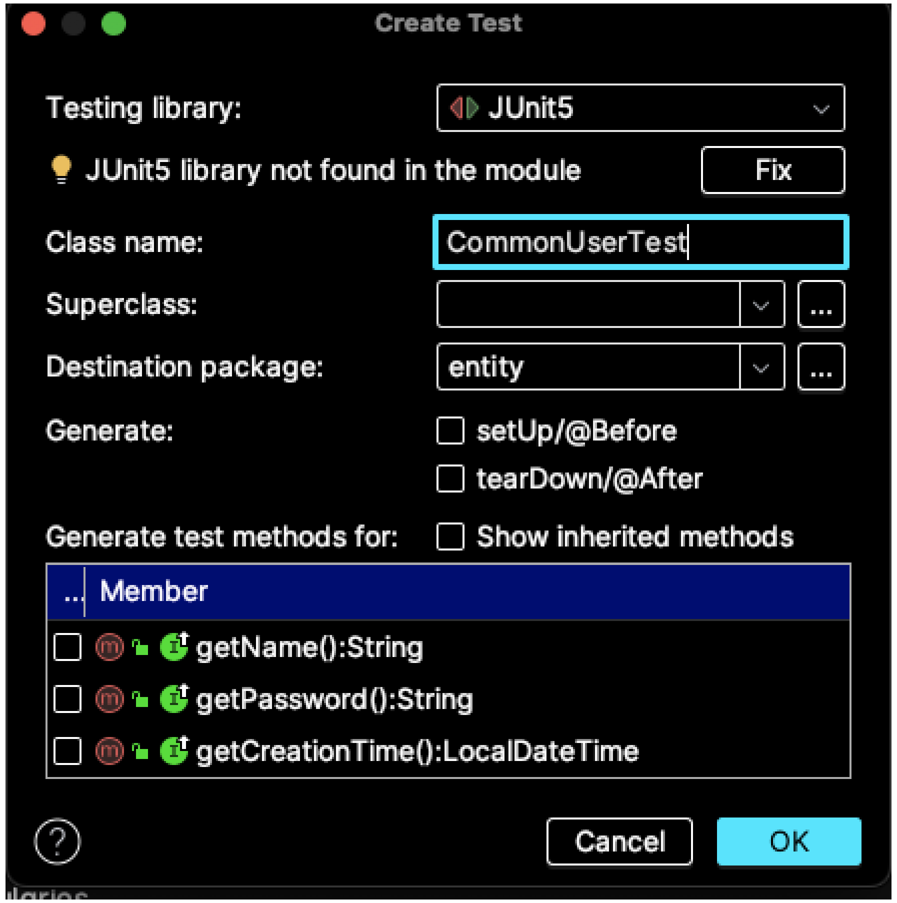


```
1 usage   paulgries
class CommonUser implements User {
    2 usage
    privat
    2 usage
    privat
    2 usage
```

Generates a test case for the selected class. The generated class contains skeleton test methods for the selected public methods.



USING JUNIT IN INTELLIJ



Testing library: JUnit5

JUnit5 library not found in the module [Fix]

Class name: CommonUserTest

Superclass:

Destination package: entity

Generate: ☐ setUp/@Before ☐ tearDown/@After

Generate test methods for: ☐ Show inherited methods

	Member
☐	getName():String
☐	getPassword():String
☐	getCreationTime():LocalDateTime

- We'll use JUnit5 here, but 4 is fine too
- Click "Fix" if you need to
- Select the methods to generate tests for

USING JUNIT IN INTELLIJ

- This will be placed in the test/entity directory (but with empty bodies for you to fill in)
- @BeforeEach methods are called before every @Test method
- There is an @AfterEach as well

```
package entity;

class CommonUserTest {

    private CommonUser user;

    @BeforeEach
    void init() {
        user = new CommonUser(
                "Paul", "password", LocalDateTime.now());
    }

    @Test
    void getName() {
        assertEquals("Paul", user.getName());
    }

    @Test
    void getPassword() {
        assertEquals("password", user.getPassword());
    }
}
```

Computer Science
UNIVERSITY OF TORONTO

TESTING CODE WITH EXCEPTIONS

```
class ExceptionDemoTest {
    @Test
    void exceptionTest() {
        Calculator calculator = new Calculator();


        // Assert that the calculator.divide call throws an exception
        Exception exception = assertThrows(
                ArithmeticException.class,
                // This creates an anonymous method that gets called by the assertThrows method
                () -> calculator.divide(1, 0)
        );
        assertEquals("/ by zero", exception.getMessage());
    }
}
```

```
class Calculator {
    public void divide(int i, int j) {
        int result = i / j;
    }
}
```

Computer Science
UNIVERSITY OF TORONTO

SETUP AND TEARDOWN

- There are three steps in running a test: **setup**, **run**, and **teardown**
- The **setup** phase is in a single method annotated with `@BeforeEach`
- The **teardown** phase is in a single method annotated with `@AfterEach`
- These are called before and after every test method
- The methods annotated with `@BeforeAll` run once before all test methods in that test class are executed, and those methods annotated with `@AfterAll` run once after.
- The `@Before*` and `@After*` methods are used to avoid repetition. For example, to create/destroy data structures required for more than one test method.

ASSERTION METHODS

- Single-Outcome Assertions
  - `fail();` OR `fail(msg);`
- Stated Outcome Assertions
  - `assertNotNull(object);` OR `assertNotNull(msg, object);`
  - `assertTrue(booleanEx);` OR `assertTrue(msg, booleanEx);`
- Equality Assertions
  - `assertEquals(exp, act);` OR `assertEquals(msg, exp, act);`
- Fuzzy Equality Assertions (for floating-point numbers)
  - `assertEquals(msg, expected, actual, tolerance);`
- https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html

TEST-DRIVEN DEVELOPMENT

- Try writing your tests first!
- Then your tests:
    - are based on requirements rather than code
    - determine the code you need to write
- Later, if you think of a situation that your code doesn't handle, add a test for it
- This approach aids in the definition of requirements
- It provides tangible evidence of progress