Software Quality Engineering

PRACTICAL SESSION NO. 1

UNIT TESTING - INTRODUCTION

Based on Introduction to Software Testing by University of Minnesota on Coursera
and Unit tests with Mockito – Tutorial on Vogella

Intro

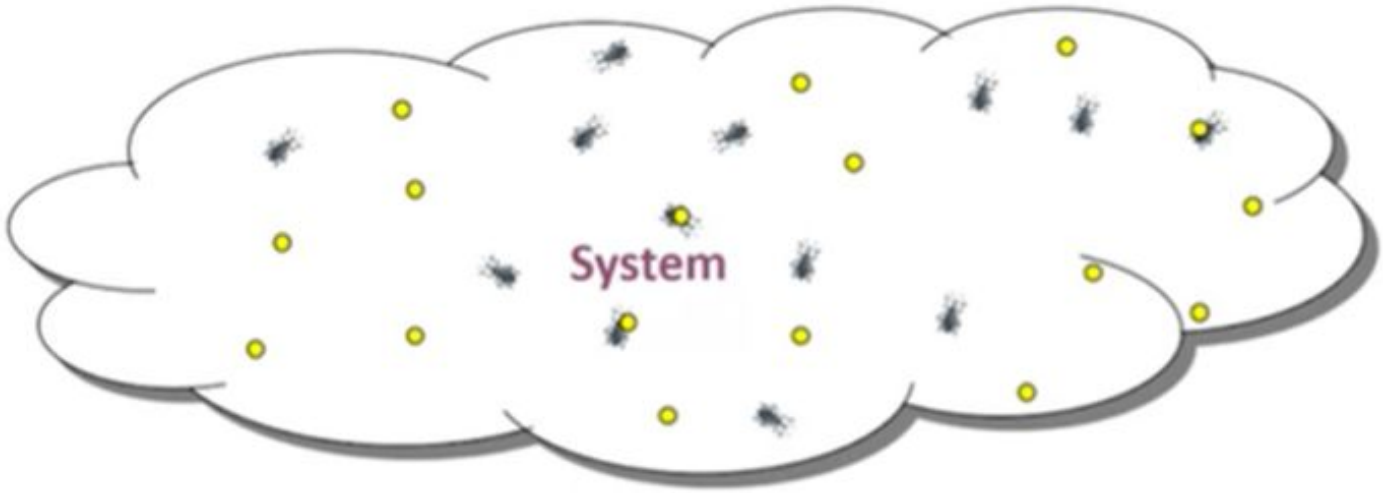
- Lecturer: Dr. Achiya Elyasaf
- Teaching Assistants:
  - Keren Gorelik
  - Maxim Bragilovski
- Assignments - 30%
- Exam - 70%
- Office hours - scheduled by email
  - Keren - gorelikk@post.bgu.ac.il
  - Maxim- maximbr@post.bgu.ac.il

Agenda

- Why is software testing challenging?
- Example: The Zune Killer Bug
- What is a Test?
- Introducing Unit Testing
- Unit Testing with Junit 5
- Example: Testing Calculator Class

Why is software testing challenging?

- Tests only sample a set of possible behaviors (limited resources)
- Most software systems are discontinuous (discrete states)



The Zune Killer

Code to switch from 'days since 1980' to 'years since 1980 + days in year'

```
year = ORIGINYEAR; /* = 1980 */

while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

The Zune Killer

Code to switch from 'days since 1980' to 'years since 1980 + days in year'

```
year = ORIGINYEAR; /* = 1980 */

while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

What happens on the last day of a leap year?

What is required to find this bug?

- Knowledge of how days/years work – what values might be problematic.
- Knowledge of where programmer mistakes are common (Edge cases) :
  - Boundary conditions
  - Complex Boolean Expressions
  - Unexpected values (negative year or a negative day)

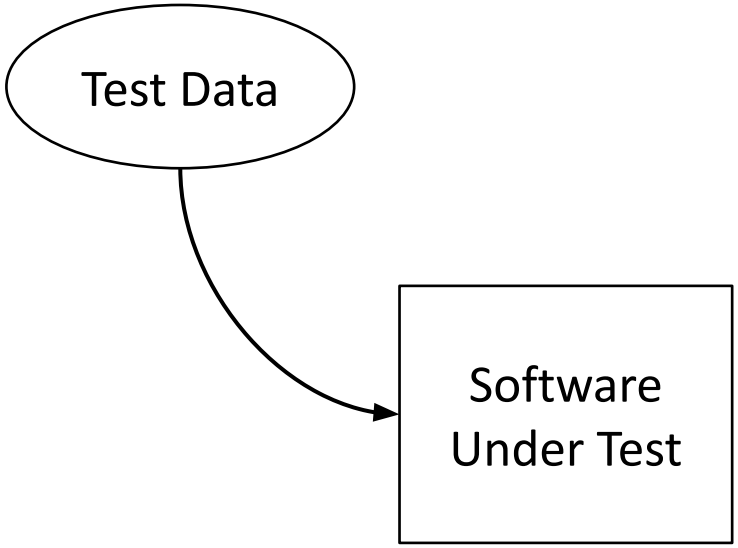
To test well we must do:

- Examine the requirements (What the system should do?)
- Examine the code (Is the code written well?)
- Think like a “helpful adversary” – how can I break this thing ?
- Test both **rainy day** scenarios and **sunny day** scenarios

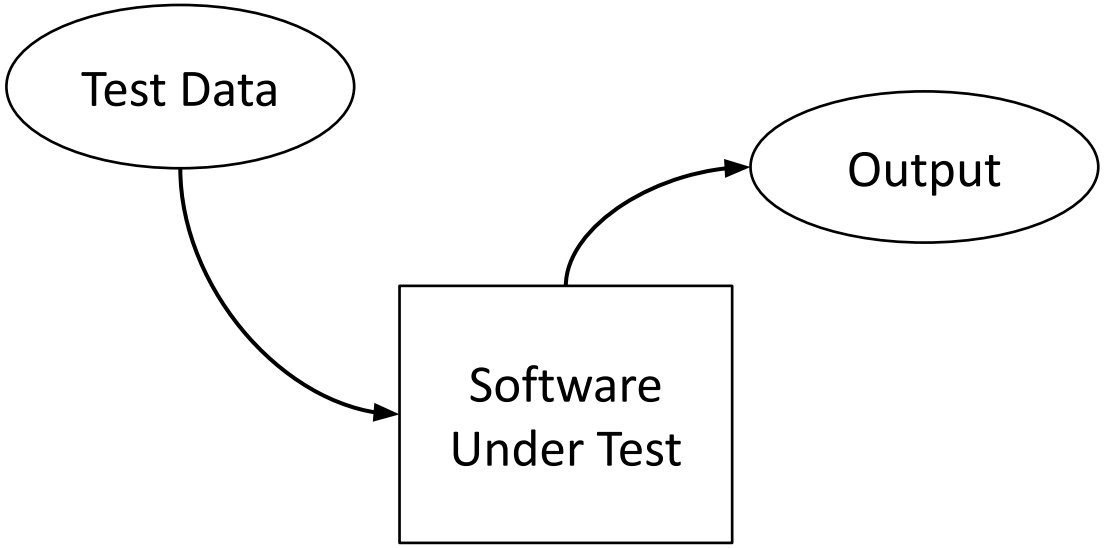
What is a test ?

Software
Under Test

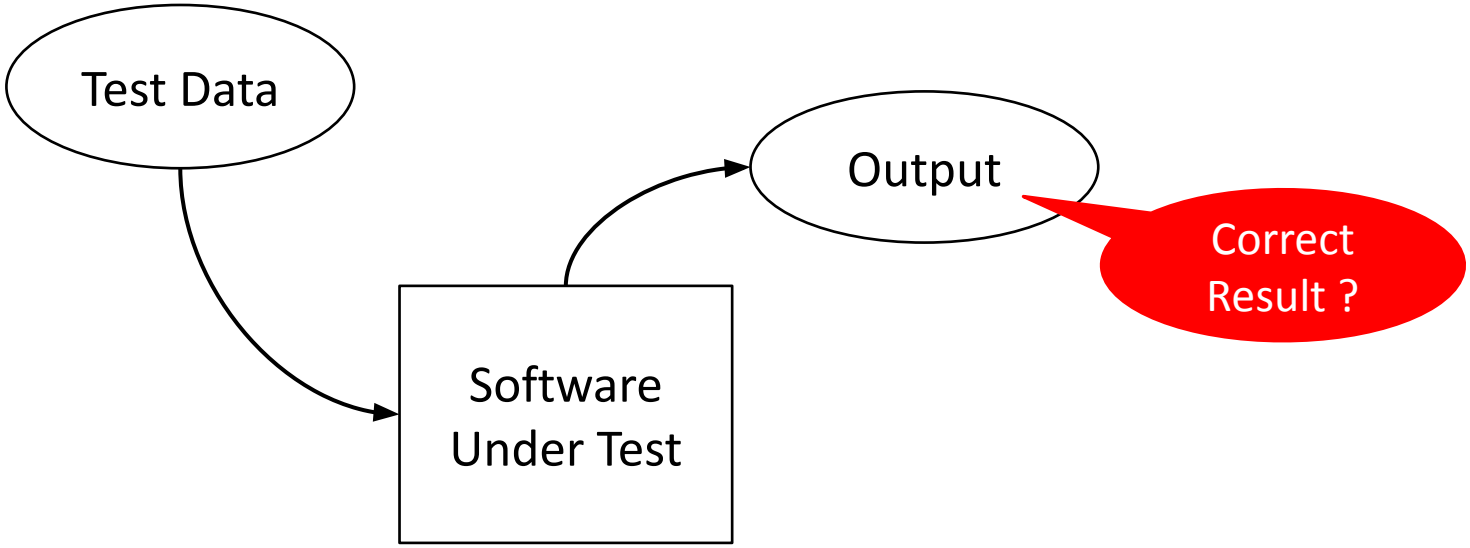
What is a test ?



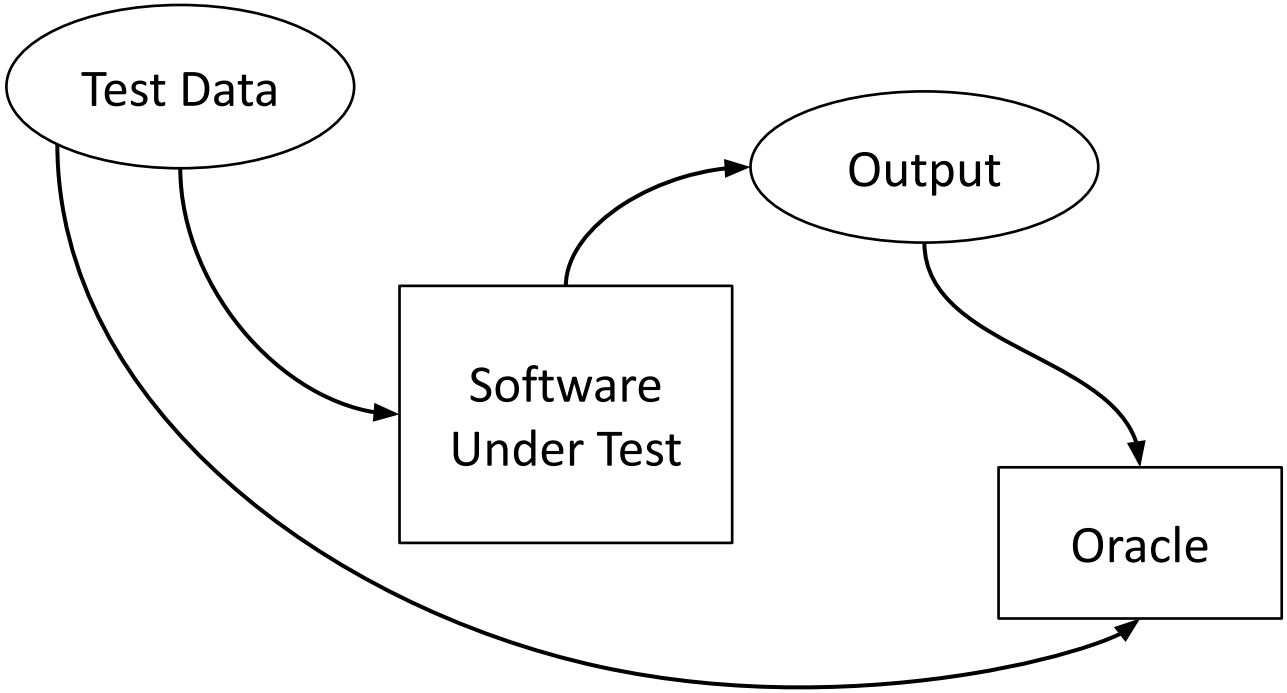
What is a test ?



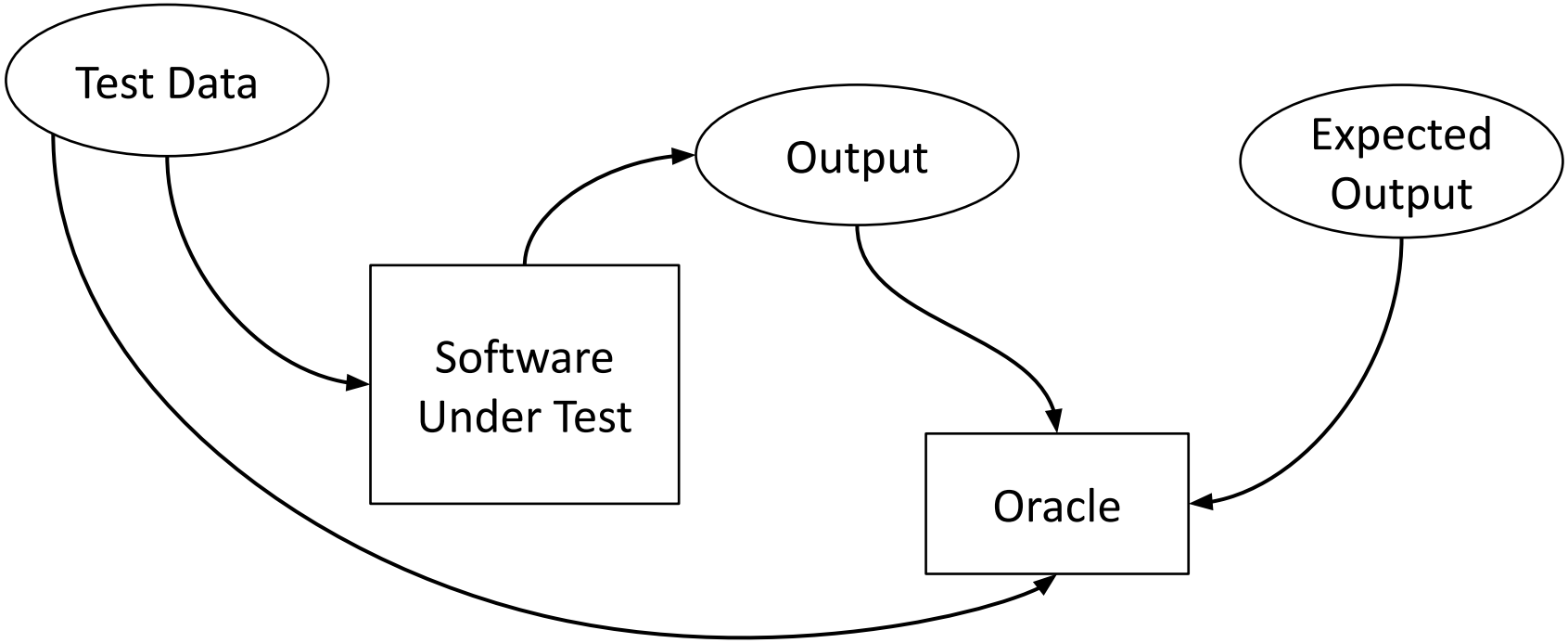
What is a test ?



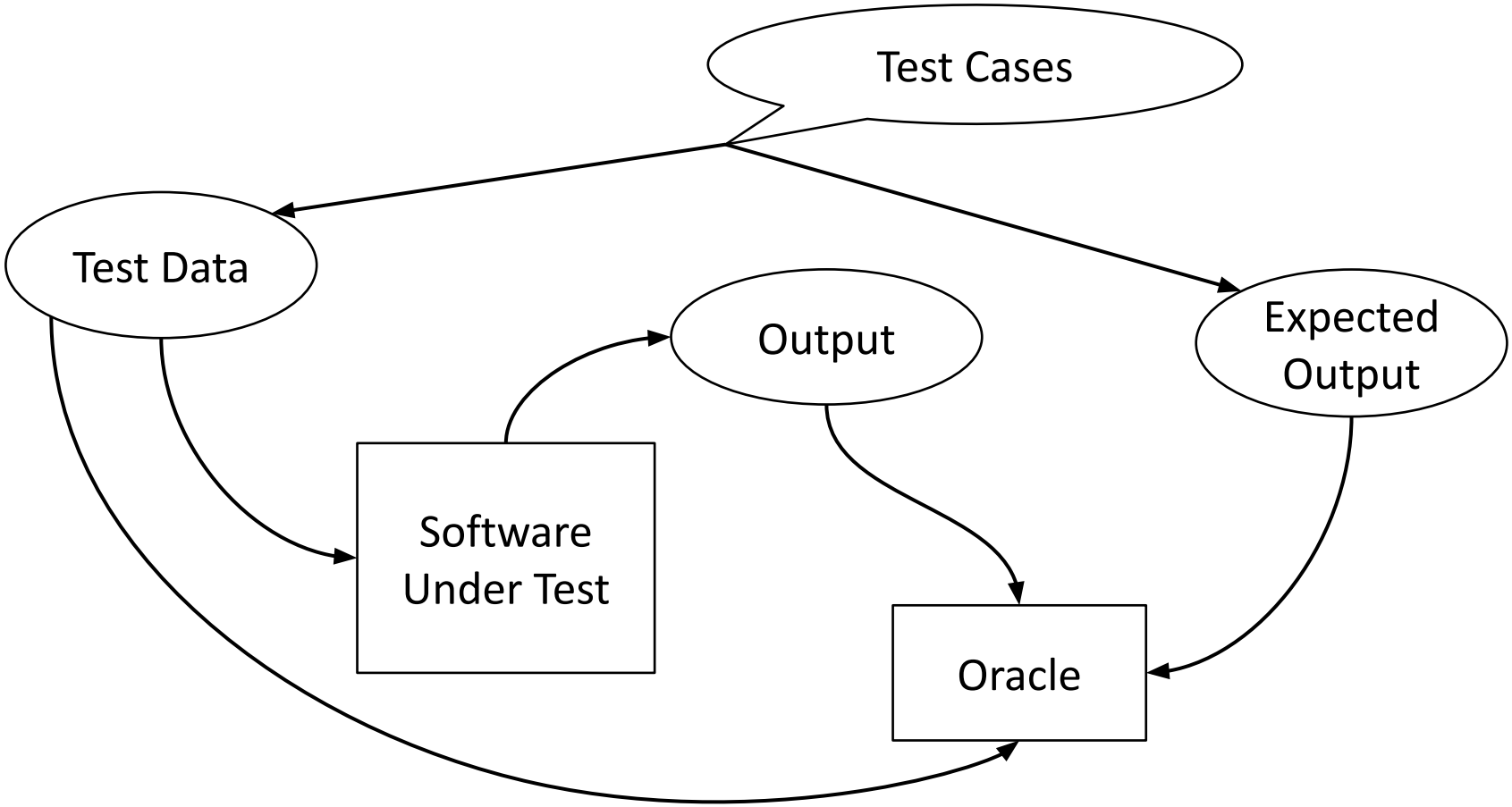
What is a test ?



What is a test ?



What is a test ?



Four parts to a test (AAA Pattern)

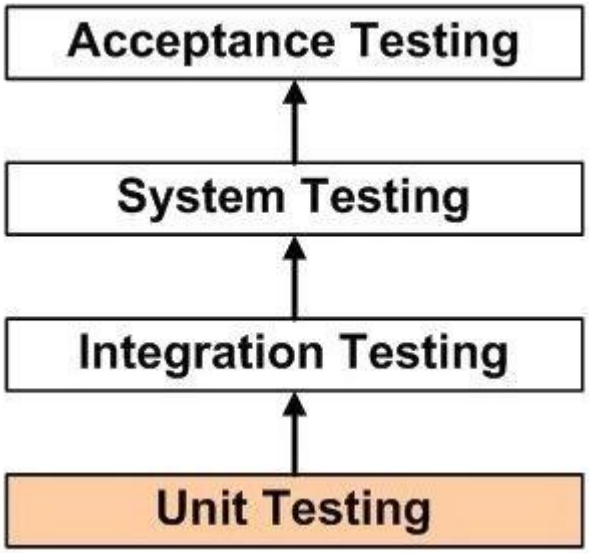
1. Setup (Arrange)
2. Invocation (Action)
3. Assessment (Assertion)
4. Teardown

What is Unit Testing?

Level of software testing that tests each individual unit of the software.

Goal: Validate whether each unit of the software performs as designed.

Unit: Smallest testable part of any software.



Unit Testing with JUnit 5

Junit is a unit testing framework for Java

Allows you to write unit tests in Java using a simple interface

Automated testing enables running and rerunning tests very easily and quickly

Junit 5 - Asserts

During a test use **Asserts** to specify if the test passed or failed.

Assert Statement:

- Used to check an expected result versus the actual result

- Allows to define messages for when the test fails

```
1    assertEquals(int expected, int actual, String message)
```

https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html

Common Asserts:

- **assertEquals / assertNotEquals:**

  Used to compare expected and actual values.

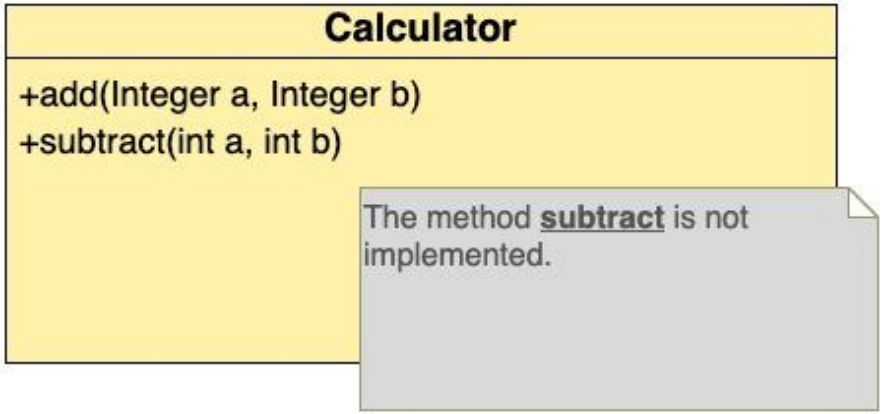
- **assertTrue / assertFalse:**

  Used to check if a given condition is true or false.

- **assertNull / assertNotNull:**

  Used to verify if a reference is null or not null.

Demo - Junit5


Calculator
+add(Integer a, Integer b)
+subtract(int a, int b)
The method subtract is not implemented.

JUnit5 - Annotations

@Test	Denotes that a method is a test method.
@BeforeEach	Denotes that the annotated method should be executed before each test method in the current class
@AfterEach	Denotes that the annotated method should be executed after each test method in the current class.
@BeforeAll	Executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database.
@AfterAll	Executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database.
@Timeout(value = n, unit = SECONDS)	Fails if the method takes longer than n seconds.
@Disabled(“optional”)	Marks a test to be disabled

JUnit 5 – Parameterized

→ **The test is marked with the *@ParameterizedTest* annotation (Instead of @Test)**

→ **The test includes a source annotation (@<Type>Source).**

- **It specifies the source of the parameter values.**

@ValueSource(ints = { 1, …, n })	Lets you define an array of test values. Permissible types are String, int, long, or double.
@EnumSource(value = class, names = {"JANUARY", "FEBRUARY"})	Lets you pass Enum constants as test class. With the optional attribute names you can choose which constants should be used. Otherwise all attributes are used.
@MethodSource(names = "genTestData")	The result of the named method is passed as argument to the test.
@ArgumentsSource(MyArgumentsProvider.class)	Specifies a class that provides the test data. The referenced class has to implement the ArgumentsProvider interface.

Testing Convention

Format: *Given<Condition>_When<Method>_Then<Result>*

Example*:* *GivenAgeLessThan18_WhenIsAdult_ThenReturnFalse*

> ***Given***: The specific condition or scenario under which the method is tested.
>
> - Example: *Given**AgeLessThan18***
>
> ***When***: The name of the method being tested.
>
> - Example: *When**IsAdult***
>
> ***Then***: The expected result or outcome of the test.
>
> - Example: *Then**ReturnFalse** / Then**Fail***

https://www.baeldung.com/java-unit-testing-best-practices

Do & Do-not in Unit Testing

- Keep conventions
  - Simple tests, only single aspect in each test.
- No random parameters
- Test edge cases and regular cases