Getting Started with Junit 5

@ParameterizedTest -> marks the method as junit 5 method that takes in some parameter.

@Tag -> declare tag and allow us to set filters on that.. We can specify tag of test we want to execute on.

@ExtendWith -> register third party extensions/annotations specially with mockito...
Or we can use to get spring extension to get all spring goodness inside our tests.

For java 11 we required passing arg-line to surefire and safe fail plugin in configuration:

```
<configuration>
	<argLine>
		—illegal-access=permit
	</argLine>
</configuration>
```

Make sure failsafe and surefire plugin has version 2.22.0 or higher to be compatible with junit 5.

Clicking on start directly for test methods, the IntelliJ runs the test-engine bypassing maven.
To run test through maven we can go to lifecycle and click on test phase...

.mvn has the maven jar file in it.

The script from maven wrapper executes the required maven...

Mvn -> runs the maven in environment of operating system..
./mvnw -> runs the bash script (containing the flavour of maven included with project) included with project.

Assertions:

```
assertEquals(2,2)
assertEquals(2,5)
assertEquals(2,5,”they are not equal”)
```

Junit 5 incorporates support of lambdas in assertions.
Grouped assertions -> all assertions run in block, all failures reported
Dependent assertions -> allows for block of grouped assertions
Expected assertions are tested with assertThrows lambda expressions

Timeouts are tested with assertTimeout lambda expressions

In junit 4 they were controlled with annotations.

Junit 5 also works with popular assertion framworks:

The java community has a variety of assertion frameworks
- Some generalised
- Other specialised i.e, JSON assertions

Popular assertions:
- AssertJ
- Hamcrest
- Truth

Grouped assertions:

```
@Test
void groupedAssertions() {
    Person person = new Person(1l,"Joe","Buck");
    assertAll("Test Props Set",
            ()->assertEquals("Joe",person.getFirstName()));
    assertAll("Test Props Set",
            ()->assertEquals("Buck",person.getLastName()));
}
@Test
void groupedAssertionsMsg() {
    Person person = new Person(1l,"Joe","Buck");
    assertAll("Test Props Set",
            ()->assertEquals("Joe",person.getFirstName(),"First Name failed"));
    assertAll("Test Props Set",
            ()->assertEquals("Buck",person.getLastName(),"Second name failed"));
}
```

First value in assertion should be expected and second actual…..

assertAll takes first value as name for group….

Grouped Assertion:
- In a grouped assertion, multiple assertions are combined together in a single test case to verify the behaviour or state of the system under test.
- All assertions are executed within the same test case, and the test case fails if any one of the assertions within the group fails.
- Grouped assertions are useful when you want to check multiple related conditions or expectations within the same test case.

Dependent Assertion:

- Dependent assertion refers to a situation where the outcome of one assertion is dependent on the success or failure of a previous assertion.
- In other words, the result of one assertion influences the condition or behaviour that is being asserted in another assertion.
- Dependent assertions are commonly encountered when testing sequential or dependent behaviour in the system under test.
-

```
assertAll("Properties Test",
      () -> assertAll("Person Properties",
            () -> assertEquals("Joes", owner.getFirstName(), "First Name Did not
Match"),
            () -> assertEquals("Bucks", owner.getLastName())),
      () -> assertAll("Owner Properties",
            () -> assertEquals("Key Wests", owner.getCity(), "City Did Not
Match"),
            () -> assertEquals("1231231234", owner.getTelephone())
      ));
```

In the provided code, each assertion within a group is evaluated independently. The failure of one assertion within the group does not affect the evaluation of other assertions within the same group.

Disable test

```
@Disabled(value ="Disabled until we learn mockito")
class OwnerSDJpaServiceTest {
   OwnerSDJpaService service;
   @BeforeEach
   void setUp() {
      service = new OwnerSDJpaService(null,null,null);
   }

   @Disabled
   @Test
   void findByLastName() {
      Owner owner = service.findByLastName("Buck");
   }

   @Test
   void findAllByLastNameLike() {
   }

   @Test
   void findAll() {
```

```
    }

    @Test
    void findById() {
    }

    @Test
    void save() {
    }

    @Test
    void delete() {
    }

    @Test
    void deleteById() {
    }
}
```

We can also give useful names to test:

```
@Test
@DisplayName("Pratik testing grouped assertions")
void groupedAssertions() {
    Person person = new Person(1l,"Joe","Buck");
    assertAll("Test Props Set",
            ()->assertEquals("Joe",person.getFirstName()));
    assertAll("Test Props Set",
            ()->assertEquals("Buck",person.getLastName()));
}
```

We can also add emojis or special characters..

Exception assertions:

```
void oupsHandler() {
        assertThrows(ValueNotFoundException.class,()->{
            controller.oupsHandler();
        });
//        assertTrue("notimplemented".equals(controller.oupsHandler()),()->"This
is some expensive " +
//                "Message to build " +
//                "for my test");
    }
```

It will run only if exception is thrown else it will fail.

Testing timeouts-> tests run within that time period...

assertionTimeout -> run as long as it takes and then when process completes it's gonna do an assertion of the timeout to check if timeout is exceeded...the test method runs in same method, so it will run for the same duration as it takes and it will show the output of the result.. So in both cases it will show the output (either it is failing or passing).

assertionTimeoutPreemptively -> once duration gets exceeded it will preempt that test early... the test method runs in different thread. Junit spawns a thread and if its gets too long it will kill it. Here it gets killed if it exceeds the expected duration and it will not show the output in that case...

Assertion vs Assumption:

Assertion is going to fail the test if assertion fails...
Assertion:

- An assertion is a statement that checks whether a condition holds true during the execution of a test case.
- Assertions are used to validate expected outcomes or behavior in the system under test.
- If an assertion fails (i.e., the condition does not hold true), the test case is marked as failed, and the execution of the test case stops.
- Assertions are typically used to check expected results or behavior that are essential for the correctness of the test case.

Assumption checks for some kind of optional condition [like if you running on ci environment or not, so we may run or not run in that condition.]. It gonna treat the test as being ignored.. It throws an aborted exception (org.opentest4j.TestAbortedException: Assumption failed: assumption is not true).
It's just like a flag.. (e.g. some data is not there just mark that but not fail it).
It does not actually fail the test but mark it ignored.

Assumption:

- An assumption is a condition that must be true for the test case to be executed or to continue its execution.
- Assumptions are used to declare prerequisites or preconditions that must be satisfied for the test case to proceed.
- If an assumption fails (i.e., the condition does not hold true), the test case is marked as "ignored" rather than failed, and the execution of the test case is halted.
- Assumptions are typically used to skip test cases or test paths that are not applicable under certain conditions, such as specific environment configurations or system states.

```
@Test
void testAssumptionTrue() {


assumeTrue("GURU".equalsIgnoreCase(System.getenv("GURU_RUNTIME")));
    System.out.println("hello");
}
```

Hello will not be printed as test case is ignored from there... because of assumption

Conditional Junit Test Execution:

```
@EnabledOnOs(OS.MAC)
    @Test
    void testMeOnMacOS() {
    }

    @EnabledOnOs(OS.WINDOWS)
    @Test
    void testMeOnWindows() {
    }

    @EnabledOnJre(JRE.JAVA_8)
    @Test
    void testMeOnJava8() {
    }

    @EnabledOnJre(JRE.JAVA_11)
    @Test
    void testMeOnJava11() {
    }

    @EnabledIfEnvironmentVariable(named = "USER", matches = "jt")
    @Test
    void testIfUserJT() {
    }

    @EnabledIfEnvironmentVariable(named = "USER", matches = "fred")
    @Test
    void testIfUserFred() {
    }
```

If it fails then that test case is ignored...