Everything on JUnit Testing

JUnit provides a framework for programming and managing unit tests in Java. It supports test driven development.

Re-running tests is easy (e.g. regression testing).

We make no changes to the Java class to be tested. The tests are run outside the class therefore we can only access public components of the class not the private ones.

For each class under test, we write a test class (driver) that contains the tests we want to run and checks the results using JUnit asserts.

JUnit provides Java annotations to tag parts e.g. @Test; indicates it is a test case to be called.

JUnit provides test class runners that execute the set of tests in a test class and generate a report. Asserts succeed silently, those that fail generate a report message and abandons the test method. Nothing happens after this. The focus is on highlighting failures.

Test methods in a test class may need a common set of objects initialized in the same way. This is known as a fixture. This is set up, before each test method is executed and torn down after each test method has executed.

Set up fixture	@Before
Execute test method	@Test
Tear down	@After

@After is used to get rid of the set up and prepare the method for next test.

In addition to Java annotation there are reflections, these are libraries that can ask an object what methods it contains or what parameters that method needs.

Test classes may contain:

1. Global variables
2. Private helper methods

@BeforeTest **@AfterTest**	Run code before/after the whole test class

All assert methods have two parameters and a third optional String parameter to display the error message:

assert(<type> expected, <type> actual)

It's good practice to name a method for testing (e.g. testMethodName)

Test Driven Development

1. Given a specification, write the tests (black box)
2. Then code and keep testing until all tests succeed
3. Then code is ready

Writing tests can reduce overall development time.