S.No.	Annotations	Description
1.	@Test	This annotation is a replacement of org.junit.TestCase which indicates that public void method to which it is attached can be executed as a test Case.
2.	@Before	This annotation is used if you want to execute some statement such as preconditions before each test case.
3.	@BeforeClass	This annotation is used if you want to execute some statements before all the test cases for e.g. test connection must be executed before all the test cases.
4.	@After	This annotation can be used if you want to execute some statements after each Test Case for e.g resetting variables, deleting temporary files ,variables, etc.
5.	@AfterClass	This annotation can be used if you want to execute some statements after all test cases for e.g. Releasing resources after executing all test cases.
6.	@Ignores	This annotation can be used if you want to ignore some statements during test execution for e.g. disabling some test cases during test execution.
7.	@Test(timeout=500)	This annotation can be used if you want to set some timeout during test execution for e.g. if you are working under some SLA (Service level agreement), and tests need to be completed within some specified time.
8.	@Test(expected=IllegalArgumentException.class)	This annotation can be used if you want to handle some exception during test execution. For, e.g., if you want to check whether a particular method is throwing specified exception or not.

JUnit Assert Class

This class provides a bunch of assertion methods useful in writing a test case. If all assert statements are passed, test results are successful. If any assert statement fails, test results are failed.

As you seen earlier, below table describes important Assert methods and description:

S.No.	Method	Description
1.	void assertEquals(boolean expected, boolean actual)	It checks whether two values are equals similar to equals method of Object class
2.	void assertFalse(boolean condition)	functionality is to check that a condition is false.
3.	void assertNotNull(Object object)	"assertNotNull" functionality is to check that an object is not null.
4.	void assertNull(Object object)	"assertNull" functionality is to check that an object is null.
5.	void assertTrue(boolean condition)	"assertTrue" functionality is to check that a condition is true.
6.	void fail()	If you want to throw any assertion error, you have fail() that always results in a fail verdict.
7.	void assertSame([String message]	"assertSame" functionality is to check that the two objects refer to the same object.
8.	void assertNotSame([String message]	"assertNotSame" functionality is to check that the two objects do not refer to the same object.