Junit Annotations
+++++++++++++++++
1. @Test
2. @DisplayName
3. @Order
4. @Disabled
5. @Tag(Can be used to run testcases through Junit and also through Maven life cycle)
6. @TestMethodOrder(value=..../.../...)
7. @BeforeEach
8. @BeforeAll[setUpCode() :: public static]
9. @AfterEach
10. @AfterAll[cleanUpCode():: public static]
11. @RepeatedTest(value=int,name="")
12. @ParameterizedTest(...)
13. @EmptySource
14. @NullSource
15. @NullAndEmptySource

AssertClass static methods
++++++++++++++++++++++++++
1. assertEquals(expectedOutput, actualOutput) :: deals with content(value)
2. assertThrows(Exception.class,Executable(I))
3. assertTimeOut(Duration,Executable(I))
4. assertTrue(boolean)
5. asssertSame(Object expected,Object acutal) :: deals with reference(==)
6. assertNotNull(Object expected)

HttpUnit
++++++++
  WebConversation :: It is used to automate the request-response phase of Client-Server architecture in webapplications.

+++++++
Mockito
+++++++
=> It is built on top of Junit
=> It is given to perform unit testing by mocking the Local Dependent or external Dependent objects.

```
      Service class ---------------------------> DAO class ----------------------> DB s/w
         |->business methods                   |-> Persistence methods
          (having business logic)                  (persistence logic)
```

=> Let us assume DAO class coding is not completed, but Service coding is completed and we want finish unit testing of service class.
   Then we need to create Mock object/Fake object/Dummy object for DAO class and assign/inject to Service class, in order write
   test cases on service class methods.

```
    eg#2                                                  (paid)
      Sherkhan.com<----------Inject--------------------BSE Component
      (ServiceClass)                              (Distrubuted component)
```

=> When Sharekhan.com website is under development, we cannot take subscription of BSE component because they charge money for that.
   Generally, this subscription will take after hosting/ releasing the Sharekhan.com till that we need to take one mock BSE component
   and assign to Service class of Sharekhan.com to perform Unit Testing.

Note: Mock objects are for temporary needs, mostly they will be used in the Unit Testing. These mock objects created in test methods
or Test case class does not affect real code.

We can do this Mocking in 3 ways:
a. Using Mock object/ Fake object (Provides Temporary object)
b. Using Stub object (Providing some Functionality for the methods of mock object like specifying for certain inputs,certain output should come)
c. Using Spy object (It is called Partial Mock object/ Half mock object that means if you provide new functionality to method that will be
used otherwise real object functionality will be used).
Note: While working with Spy object we will be having real object also.

=> Instead of creating classes manually to prepare Mock, Stub and Spy objects, we can use mocking frameworks available in the market which
are capable generate such classes dynamically at runtime as InMemory classes (That classes that are generated in the JVM memory of RAM).

Note::
1. Normal Class

```
        Compilation
    .java(HDD)--javac--> .class(HDD)

         Execution
     .class(HDD) ---JVM--> Output
```

2. InMemory Class

```
                    RunTime                RunTime
      .java(JVM Memory)-------->bytecode(JVM Memory)---------> JVM ----> Output
```

List of Mocking Frameworks:
o Mockito (popular)
o JMockito
o EasyMock
o PowerMock
And etc.

Example Application setup:[Testing LoginMgmtService class without keeping LoginDAO class ready]

Step 1: Create maven standalone App
File -> Maven Project -> next -> select maven-archetype-quickstart ->
GroupId: ineuron
ArtifactId: MockitoUniTesting-LoginApp
Default package: in.ineuron.service

Step 2: Do following operations in pom.xml file,change java version to 1.8
Add the following dependencies (jars)
o junit-jupiter-api.5.7.0.jar
o junit-jupiter-engine.5.7.0.jar
o mockito-core.3.6.28.jar

Step 3: Develop service interface, service class
src/main/java
in.ineuron.service
|-> ILoginMgmtService.java (I)
|-> LoginMgmtServiceImpl.java (c)

```
      in.ineuron.dao
            |-> LoginDAO.java

src/test/java
      in.ineuron.test
            |-> TestLoginMgmtService.java(C)

Step 4: Develop Mockito based Test classes and DAO interface
Step 5: Run Tests.

ILoginDao.java
+++++++++++++++
public interface ILoginDao {
      public int authenticate(String username, String password);
}


ILoginMgmtService.java
++++++++++++++++++++
public interface ILoginMgmtService {
      public boolean login(String username, String password);
}
LoginMgmtServiceImpl.java
+++++++++++++++++++++++
public class LoginMgmtServiceImpl implements ILoginMgmtService {

      private ILoginDao dao;

      public LoginMgmtServiceImpl(ILoginDao dao) {
            this.dao = dao;
      }

      @Override
      public boolean login(String username, String password) {
            if (username.equals("") || password.equals("")) {
                  throw new IllegalArgumentException("Empty Credentials...");
            }

            // use DAO
            int count = dao.authenticate(username, password);
            if (count == 0) {
                  return false;
            } else {
                  return true;
            }
      }
}


Performing UnitTesting on Service class method called "login(username,password)"

TestLoginMgmtService.java
++++++++++++++++++++++++
public class TestLoginMgmtService {

      private static ILoginMgmtService service;
      private static ILoginDao loginDaoMock;
```

```
	@BeforeAll
	public static void setUpOnce() {
		loginDaoMock = Mockito.mock(ILoginDao.class);
		System.out.println(loginDaoMock.getClass().getName()
+""+Arrays.toString(loginDaoMock.getClass().getInterfaces()) );
		service = new LoginMgmtServiceImpl(loginDaoMock);
	}

	@AfterAll
	public static void cleanUpOnce() {
		loginDaoMock = null;
		service = null;
	}

	@Test
	public void testLoginWithValidCredentials() {
		Mockito.when(loginDaoMock.authenticate("root",
"root123")).thenReturn(1);
		Assertions.assertTrue(service.login("root", "root123"));
	}

	@Test
	public void testLoginWithInValidCredentials() {
		Mockito.when(loginDaoMock.authenticate("root",
"root123@")).thenReturn(0);
		Assertions.assertFalse(service.login("root", "root123@"));
	}

	@Test
	public void testLoginWithNoCredentials() {
		Mockito.when(loginDaoMock.authenticate("", "")).thenReturn(0);
		Assertions.assertThrows(IllegalArgumentException.class, () ->
service.login("",""));
	}

}
```

Difference b/w Mock and Spy object

```
++++++++++++++++++++++++++++++++
public class MockSpyTest {

	@Test
	public void testList() {

		List<String> listMock = Mockito.mock(ArrayList.class);//mock object
		List<String> listSpy  = Mockito.spy(new
ArrayList<String>());//spyobject
		listMock.add("table");
		listSpy.add("sachin");
		System.out.println(listMock.size()+" "+listSpy.size());
	}
}
```

Output

```
 0 1
```

Stubbing on Spy and Mock Object

```
++++++++++++++++++++++++++++++
public class MockSpyTest {
```

```
	@Test
	public void testList() {

		List<String> listMock = Mockito.mock(ArrayList.class);//mock object
		List<String> listSpy  = Mockito.spy(new ArrayList<String>());//spy
object

		listMock.add("table");
		Mockito.when(listMock.size()).thenReturn(10);//stubbing the mocking
object

		listSpy.add("sachin");
		Mockito.when(listSpy.size()).thenReturn(10);//stubbing on spy object


		System.out.println(listMock.size()+" "+listSpy.size());
	}
}
```

Output
10 10

Note: Spy objects are useful to check how many time methods are called whether they are called or not. Because Spy object is always linked with
real object (for this use Mockito.verify(-,-) method).

ILoginDao.java

```
++++++++++++++
public interface ILoginDao {
	public int addUser(String user, String role);
}
```

ILoginMgmtService.java

```
++++++++++++++++++++++
public interface ILoginMgmtService {
	public String registerUser(String username,String roles);
}
```

LoginMgmtServiceImpl.java

```
+++++++++++++++++++++++++
public String registerUser(String username, String roles) {
	if (!roles.equalsIgnoreCase("") && !roles.equalsIgnoreCase("visitor")) {
		dao.addUser(username, roles);
		return "User added";
	} else
		return "User not added";
	}
}
```

TestLoginMgmtService.java

```
+++++++++++++++++++++++++
@Test
public void testRegisterUser() {
	ILoginDao loginDaoSpy=Mockito.spy(ILoginDao.class);
	ILoginMgmtService service = new LoginMgmtServiceImpl(loginDaoSpy);

	service.registerUser("sachin", "admin");
	service.registerUser("dhoni", "visitor");
```

```
	service.registerUser("kohli","");

	Mockito.verify(loginDaoSpy,Mockito.times(1)).addUser("sachin", "admin");

	Mockito.verify(loginDaoSpy,Mockito.times(0)).addUser("dhoni", "visitor");

	Mockito.verify(loginDaoSpy,Mockito.never()).addUser("kohli", "");

}
```

Mockito Annotations:
- @Mock: To generate mock object
- @Spy: To generate spy object
- @lnjectMocks: To Inject Mock or Spy Objects Service class.

MockitoAnnotations.openMocks(this); - call this method in @Before or constructor Testcase class in order to activate Mockito Annotations.

++++++++++++++++++++++++++++++++++
AnnotationTestLoginServiceImpl.java
++++++++++++++++++++++++++++++++++

```
public class AnnotationTestLoginServiceImpl {

	@Mock
	private static ILoginDao loginDaoMock;

	@Spy
	private static ILoginDao loginDaoSpy;

	@InjectMocks
	private static LoginServiceImpl loginService;


	public AnnotationTestLoginServiceImpl() {
		MockitoAnnotations.openMocks(this);
	}

	@Test
	public void testLoginWithNoCredentials() {
		System.out.println(loginService);
		assertThrows(IllegalArgumentException.class, () ->
loginService.login("", ""));
	}

	@Test
	public void testRegisterWithSpy() {
		ILoginDao loginDaoSpy = Mockito.spy(ILoginDao.class);
		System.out.println("Spy object is :: " +
loginDaoSpy.getClass().getName());
		LoginServiceImpl loginService = new LoginServiceImpl(loginDaoSpy);

	System.out.println("AnnotationTestLoginServiceImpl.testRegisterWithSpy() ::
"+loginService);

		loginService.registerUser("sachin", "admin");
		loginService.registerUser("dhoni", "visitor");
		loginService.registerUser("kohli", "");

		Mockito.verify(loginDaoSpy, Mockito.times(1)).addUser("sachin",
```

```
"admin");
            Mockito.verify(loginDaoSpy, Mockito.times(0)).addUser("dhoni",
"visitor");
            Mockito.verify(loginDaoSpy, Mockito.never()).addUser("kohli", "");
      }

      @Test
      public void testLoginWithInValidCredentials() {
            System.out.println(loginService);

            // Provide stub(providing functionality) for DAO authenticate method
            Mockito.when(loginDaoMock.authenticate("sachin",
"sachin@123")).thenReturn(0);

            // call login method to get the result
            boolean acutalOutput = loginService.login("sachin", "sachin@123");

            // compare the boolean result using assert
            assertFalse(acutalOutput);
      }

}

Note: To write stub functionality according agile user stories/ JIRA user stories
(given, when, then)
      BDDMockito.given(loginDAOMock.authenticate("sachin",
"tendulkar")).willReturn(1);
                  |
      Mockito.when(loginDAOMock.authenticate("sachin", "tendulkar")).thenReturn(1);


What is CSRF problem?
   CSRF :: It stands for CrossSiteRequestForgery Problem.

It stands for fishing or hacking technique of hacker or attacker, who makes the
innocent enduser sendhing his data to usersites and accounts.
            eg: sending spammails,trapping mails
      for different websites the attacker makes the victim to send following
details by showing lottery ticket details.

<form action ="http://icicibank.com/transferfunds" method="POST">
      <input type="hidden" name="amount" value="10000"/>
      <input type="hidden" name="srcAccount" value="444555666"/>
        <input type="hidden" name="desAccount" value="656778"/>
        <input type='submit' value='clickheretowinlottery'/>
</form>
                        (or)
  <a href='http://icicibank.com/transferfunds?
amount=10000&srcAccount=444555666&desAccount=656778">Click here to win Lottery </a>

++++++++++++++++++++++++++++++
Feeling CSRF problem practially
++++++++++++++++++++++++++++++
  1. By default CSRF will be enabled in SpringBoot or Spring applications.
         http.csrf().disable();

  2. run springboot security application with authentication and to use one or two
services
         http://localhost:99999/bank/
```

```
            |
      click on check balance
            |
      username : sachin
      password : tendulkar
        click on login ==========> ShowBalance Page
                                   balance :: 40734312
                                   Home
                                   logout
```

3. Create another dynamic webproject with an index.html page

```
      index.html
      ++++++++++
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>

<head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title>Login Page</title>
</head>
<body>
      <form action="http://localhost:9999/bank/balance" method="post"">
            <input type="submit" value="click here to win lottery" />
      </form>
</body>
</html>
```

4. Run the application by keeping the oldbrowser ready, without logout
   we can feel the fishing or hacking problem by clicking the button.

```
++++++++++++++++
Solution to CSRF
++++++++++++++++
1. make sure CSRF is disabled in "SecurityConfig" class.[default it will be enabled
in SpringBoot]
2. add CSRF token in custom-login.html which is a form page
      <input type="hidden"  th:name="${_csrf.parameterName}"  th:value="$
{_csrf.token}"/>

3. In another webapplication index.html try to send the request
      <form action="http://localhost:9999/bank/balance" method="post"">
            <input type="submit" value="click here to win lottery" />
      </form>

How does CSRF internally works?
 when csrf protection is enabled,one session token will be generated as a session
attribute having
    _csrf as token name and "32" digits hexadecimal value.

 using the following hidden box
      <input type="hidden"  th:name="${_csrf.parameterName}"  th:value="$
{_csrf.token}"/>
 we try to get the csrf token name and token value to the form page along with
submission.
The security environment at the server side takes the token name and value coming
from browser and validates with already available session
token value. if it is matching then the activities will be continued or allowed, if
```

request comes with invalid token or no token then
error will be used.

Client-Server Architecture in realtime
++++++++++++++++++++++++++++++++++++

React   application
Angular application   (reactlibrary)   @CrossOrigin("*")
Mobile application -----axios(http)------------> Restful Controller ------------
ServiceLayer--------DAOLayer----> Database
Postman application
chrome   browser

==>Wednesday{React with SpringBoot Integration}