**Unit Testing:**

Unit testing in use to test the certain functionality of the code.

The Class used to test the functionality is called System/Class Under Test.

The Method written in the Class to test the functionality is called the Method Under Test.

Unit test: Initiate the test under the class, execute method under test, and verify the result is as expected or not.

Good Unit test:

Easy to write

Readability

Reliable: test doesn’t dependent on the each other then it call the reliable.

Fast

Isolated: test is the dependent on the third party api then it should be the isolated so that it is Reliable, Fast and Easy to write, Readability.

**Test Doubles:**

External Dependancy should be remove from the unit test by replacing the real object with their fack replacement is called test doubles.

Change the production object with the test object is called the test double.

**Why we need the Test Doubles:**

When we have the database operation.

When we have the Third party services.

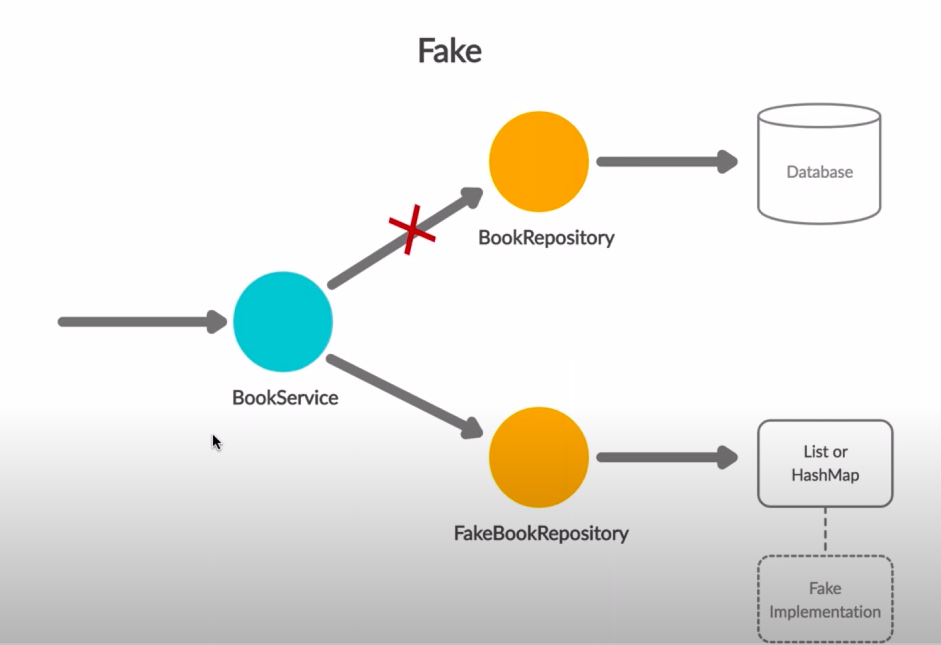
Difference types of the Test Doubles:

1. Dummy: not have any business logic. Dummy object is used to compile the project.

EX: We have pass the email service as the dummy object in the service.

1. Fake: Fake object is the light weight object that is only use for the testing purpose.(use In memory database, Hashmap or list is used)

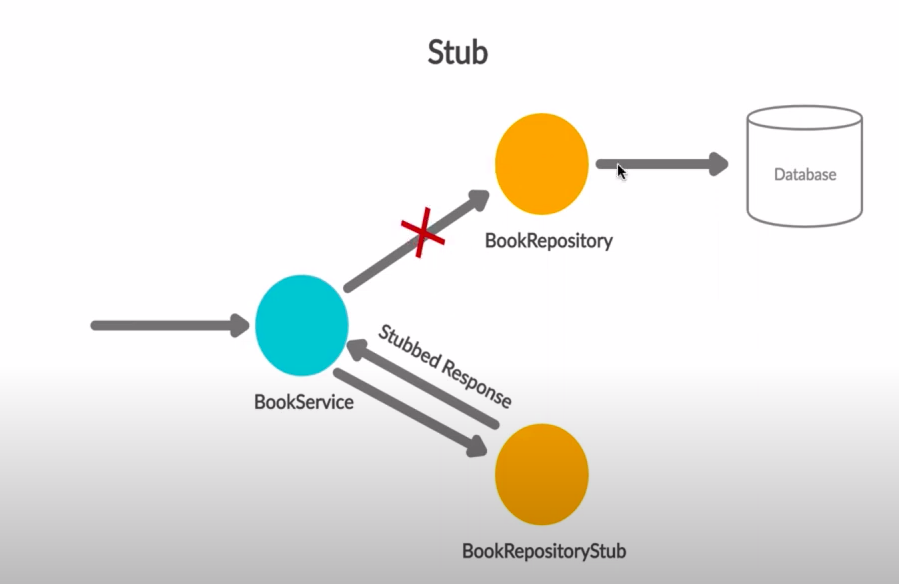
Ex: We create the Fake object using the Hashmap instead of getting that from the database.



1. Stub: Stub object provide predefine answer to method executions made during the test.

Behaviour is hardcoded programmatically for particular test.

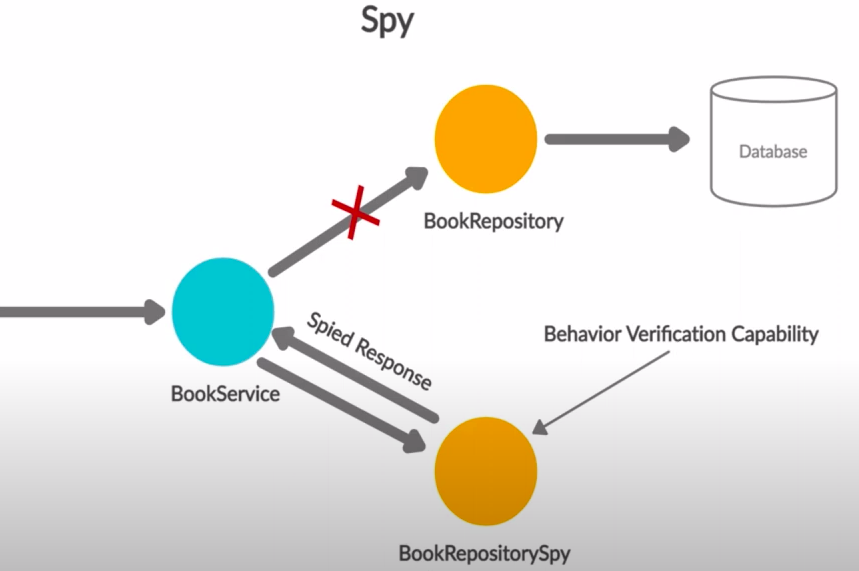
Instead of calling the external service call we call the stub and it return the object as same as third party service object.



1. Spy:

Spy is the same as the stub but apy can record the information about how they were executed.

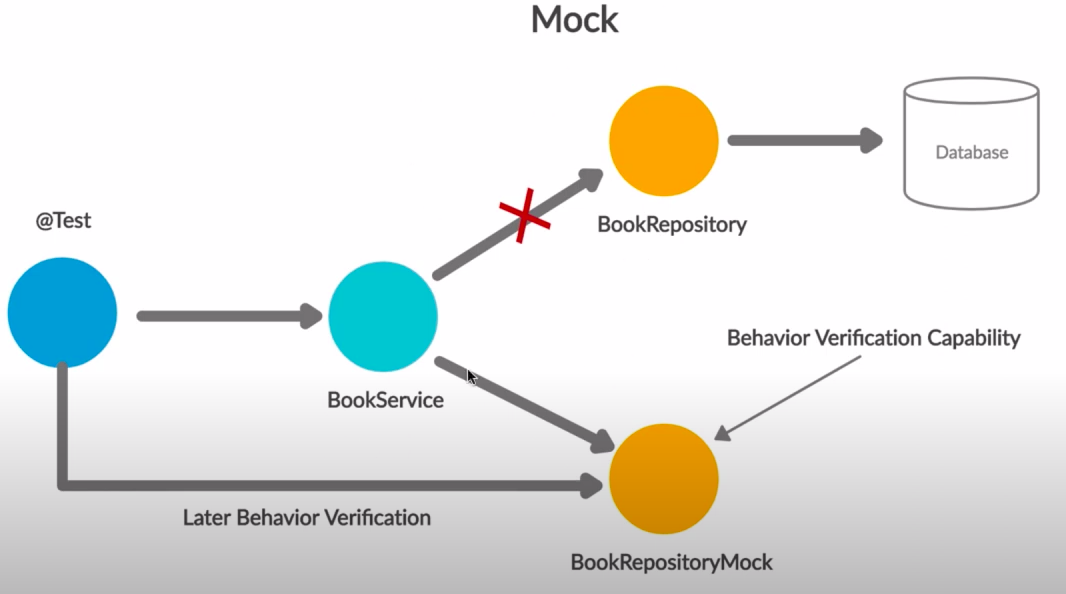
User to stubbed the External dependency and record every interaction with external dependencies.



1. Mock: It is similar to spy.

It record the methods calls on it and verify the record match the call or not.

Assert and behaviour verification are done on the mock object.



**Mokito:** Mokito is the Java Framework to test the feature.

Create test double for unit test. It **uses** **Java reflection** in order to test the mock object using the mokito.

Advantages:

Save developer to write the test double and the mock object

Provide the support for the Exceptional handling.

Support the stumbling method.help te return the configured object.

Check order of the methods being called and number of time the method is called.

Junit test Annotations

<https://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations>

AssertJ: to Validate the Result

<https://assertj.github.io/doc/>

Integration Testing:

Top Down Approch, Bottom Up Approch for the integration testing.

Stubing:

Process of writing hoe the mock method is behave is called mocking.

There is Two Ways if it return the value:

**When() + thenReturn()**

**doReturn() + when()**

Mockito uses the equals() method while matching the arguments while stubing.

Suppose we have the **void method that does not reatun the value** then we can do like below:

**doNoting().when(repo).save(user);**

**bookservice.addUser(user);**

**verify(): used to verify the method.**

Example we need to verify that save student is called when we called the addStudent method.

Example: verify(repository,**time**(1)).save(STUDENTOBJ);

verify(repository,**atLeast**(1)).save(STUDENTOBJ);

verify(repository,**atLeastOnce**()).save(STUDENTOBJ);

verify(repository,**atMost**(1)).save(STUDENTOBJ);

verify(repository,**atMostOnce**()).save(STUDENTOBJ);

verify(repository,**never**()).save(STUDENTOBJ); // not called the save method ever in the database

**If we have not perform any activity with the repository then**

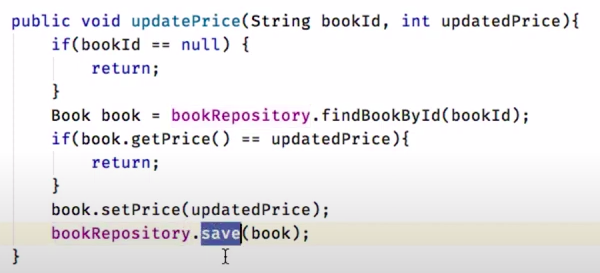
Ex: **Studentservice.update(null, 25);** with student id and age parameter.

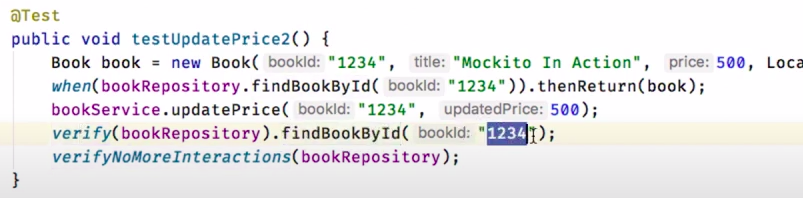
void update(int id, int age) {

if(id==null) return;

}

**verifyNoInteraction(repo);**





**verifyNoMoreInteraction(repo);**

**Check wheter the operation are performed in the sequence or not?**

InOrder inOrder = Mockito.inOrder(repository);

inOrder.verify(repository).findById(id);

inOrder.verify(repository, timeout(1)).deleteById(id);

**Throws the Exception:**

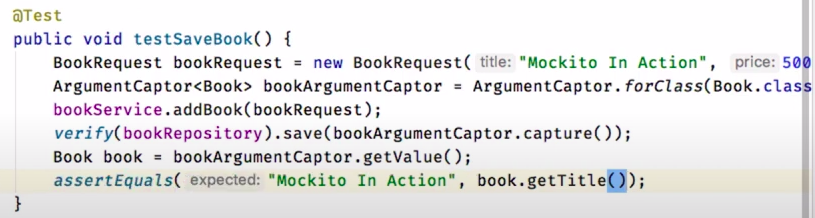
1. **For the Null return type**

doThrow(new CustomException("Some error message")).when(repository).getStudentAgeGreaterThan18AndNameIsPritesh();

1. **For Non Null Return type**

assertThrows(CustomException.class,() -> studentService.getStudentAndThrowException());

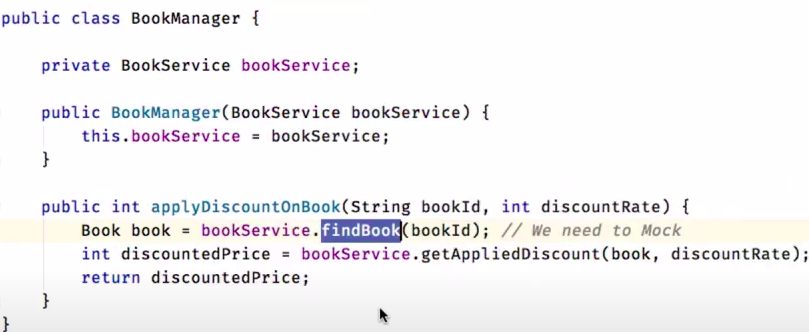
If we have the request and the repository DAO is different then we need argumentCaptor.

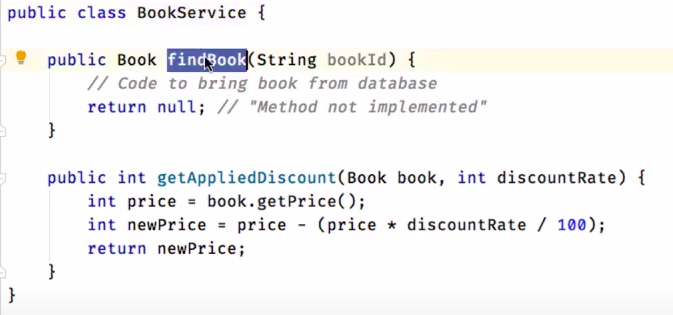


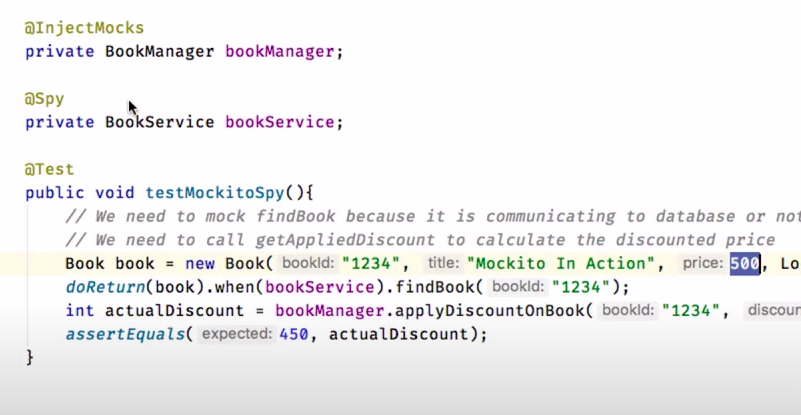
With spring boot annotation:



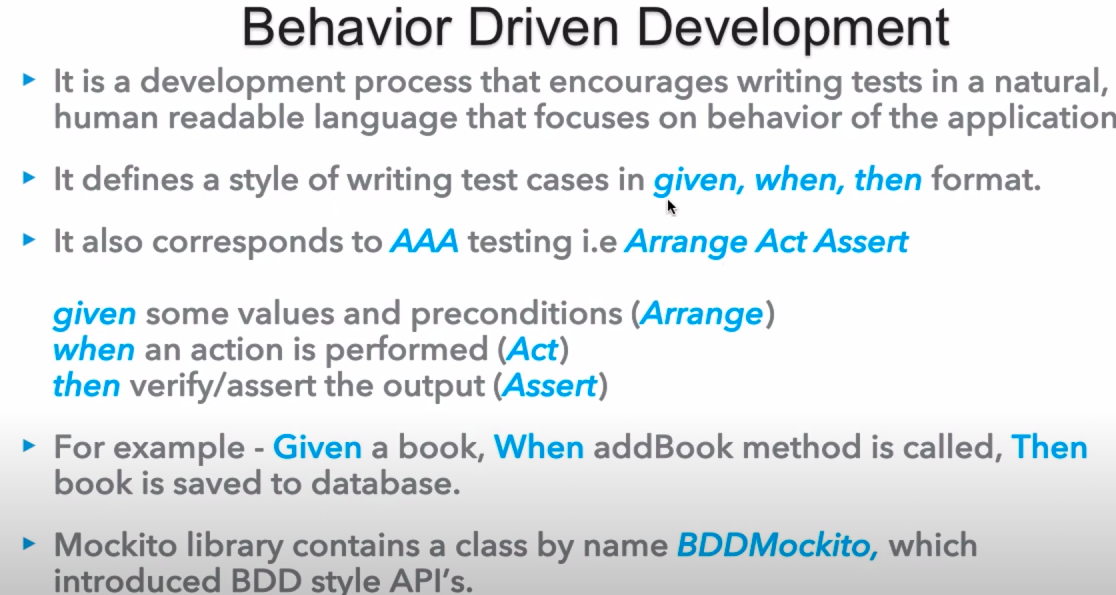
**Example of the Spy:**

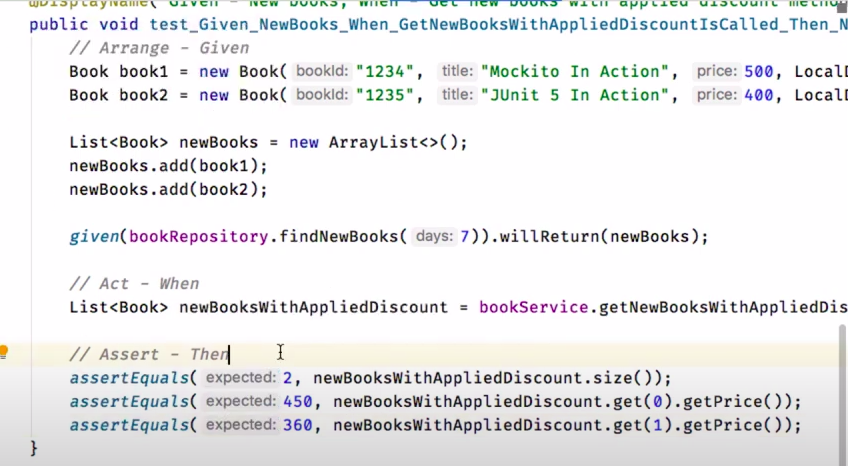
****

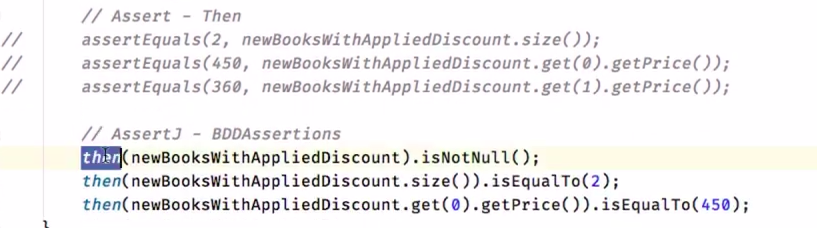
****

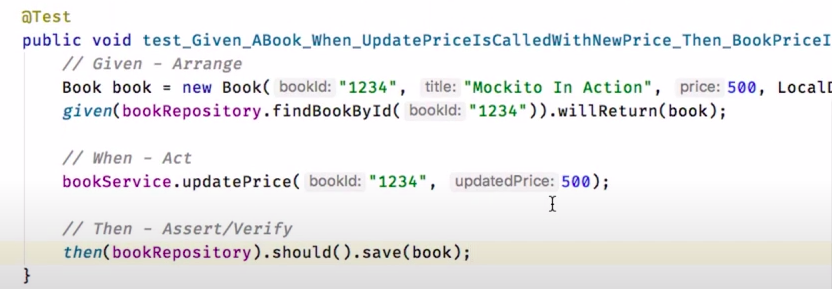
****

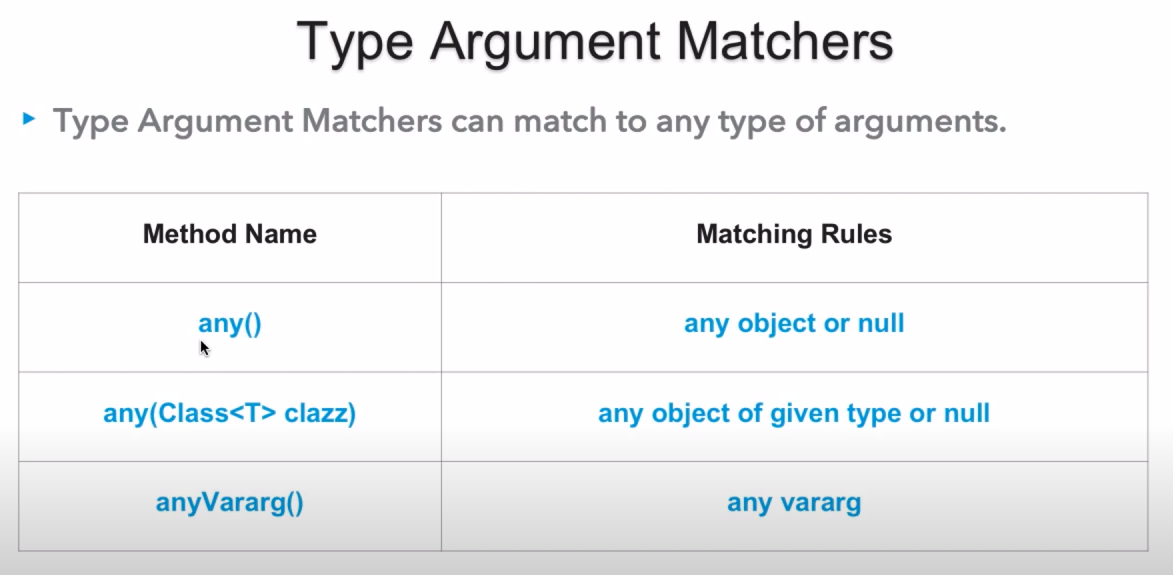
**Behaviour driven test. Treditional Mock Test:**

****

****

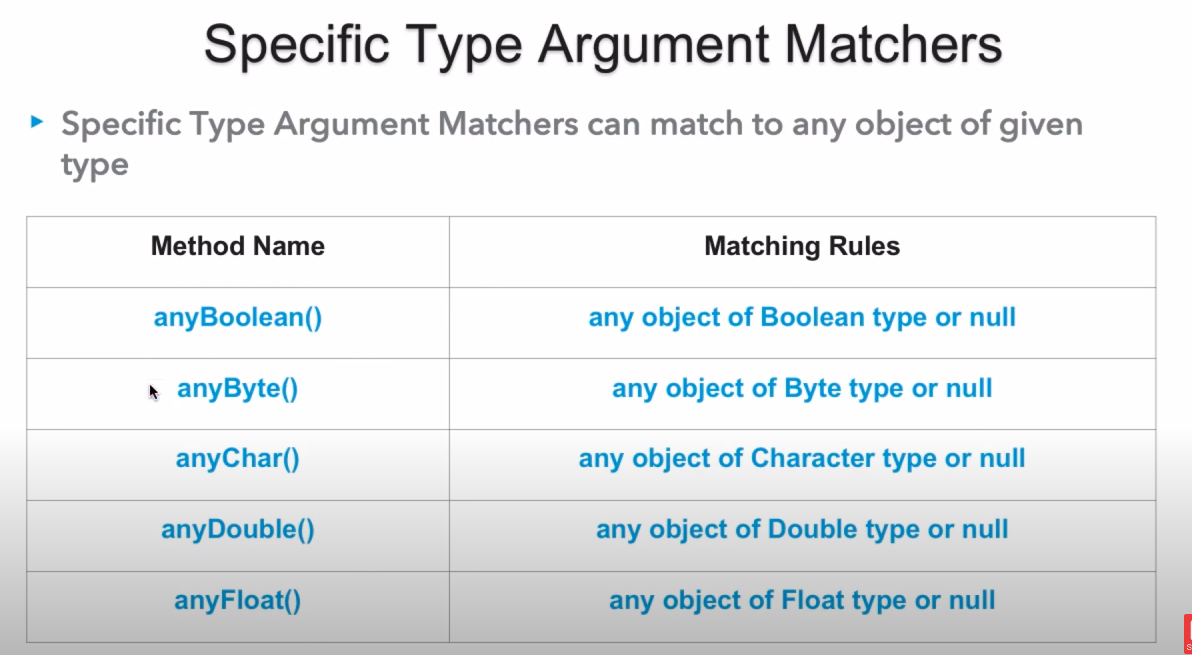
****

****

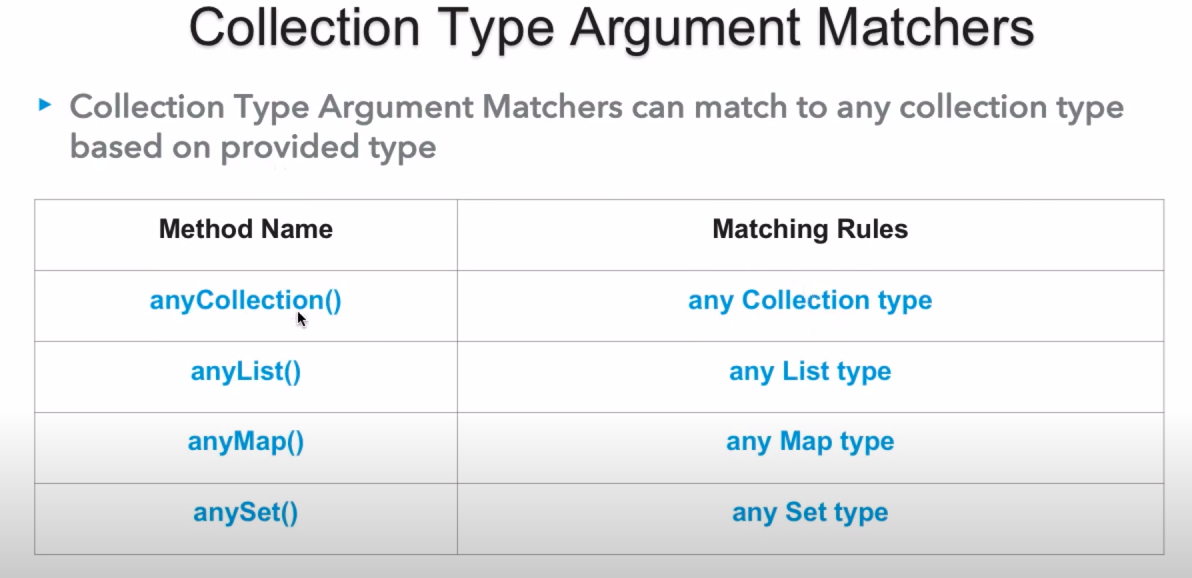
****

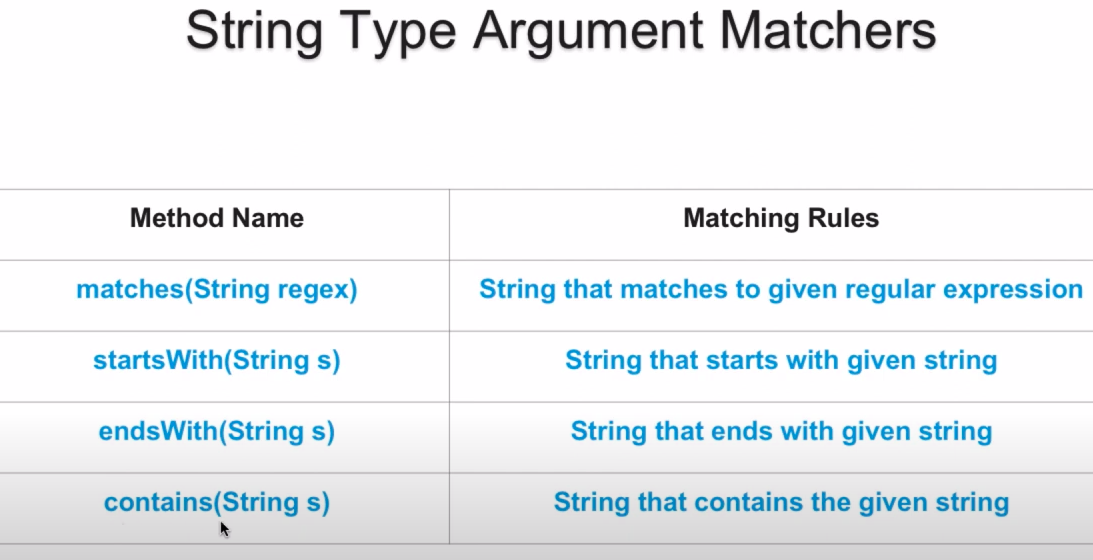
****

****

****

****

****

****

**Youtube:** [**https://www.youtube.com/watch?v=RfErIPo94bc**](https://www.youtube.com/watch?v=RfErIPo94bc)

**Github:** [**https://github.com/dinesh-varyani/mockito**](https://github.com/dinesh-varyani/mockito)

**DisplayName: @**DisplayName is use for give the name of the test.

**Tagging and Filtering:** Those tags can later be used to filter test discovery and execution.

**Order: @TestMethodOrder, @TestClassOrder**

[**TestInfo**](https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/TestInfo.html) **,** [**RepetitionInfo**](https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/RepetitionInfo.html)**,** [**TestReporter**](https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/TestReporter.html)