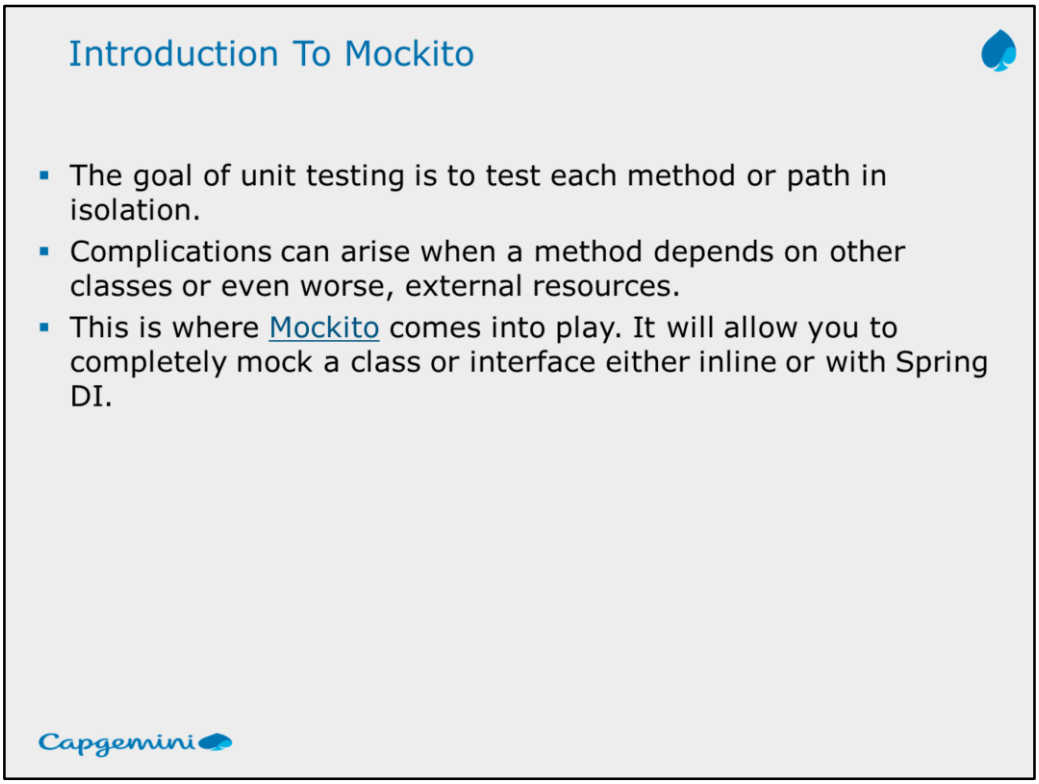
Instructor Notes:

Add instructor notes here.

Instructor Notes:



In software development there is an opportunity of ensuring that objects perform the behaviors that are expected of them.

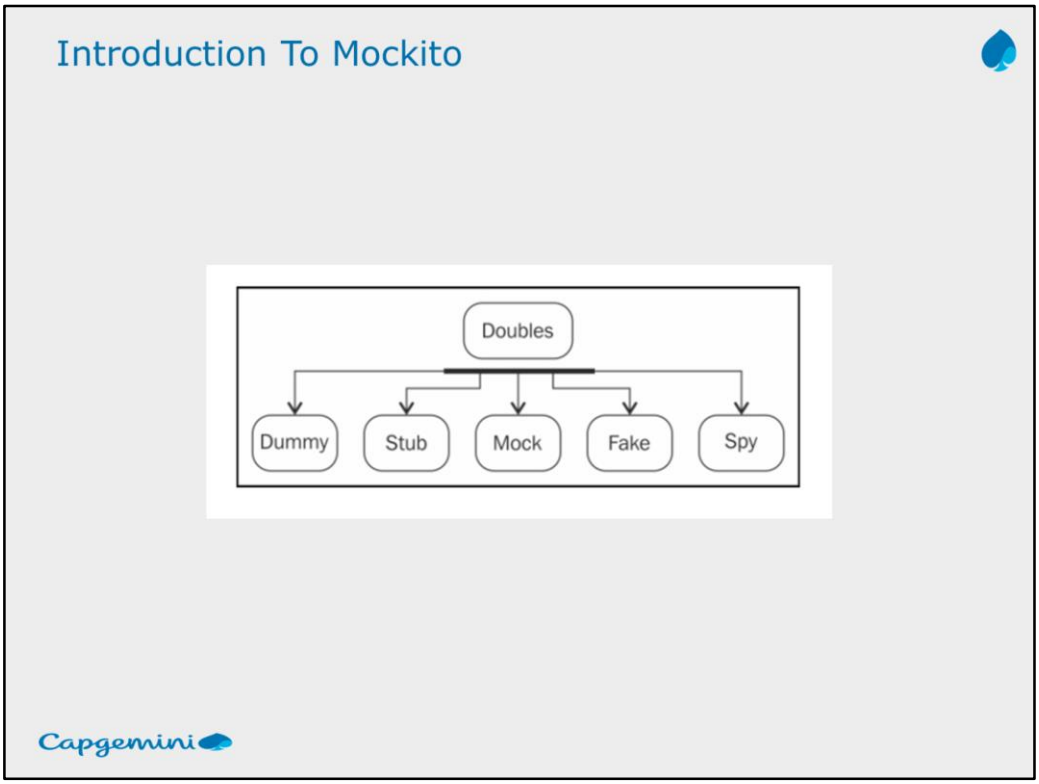
One approach is to create a test automation framework that actually exercises each of those behaviors and verifies that it performs as expected,
even after it is changed.

However, the requirement to create an entire testing framework is often an onerous task that requires as much effort as writing the original objects that were supposed to be tested.

For that reason, developers have created mock testing frameworks. These effectively fake some external dependencies so that the object being tested has a consistent interaction with its outside dependencies.

Mockito intends to streamline the delivery of these external dependencies that are not subjects of the test

Instructor Notes:



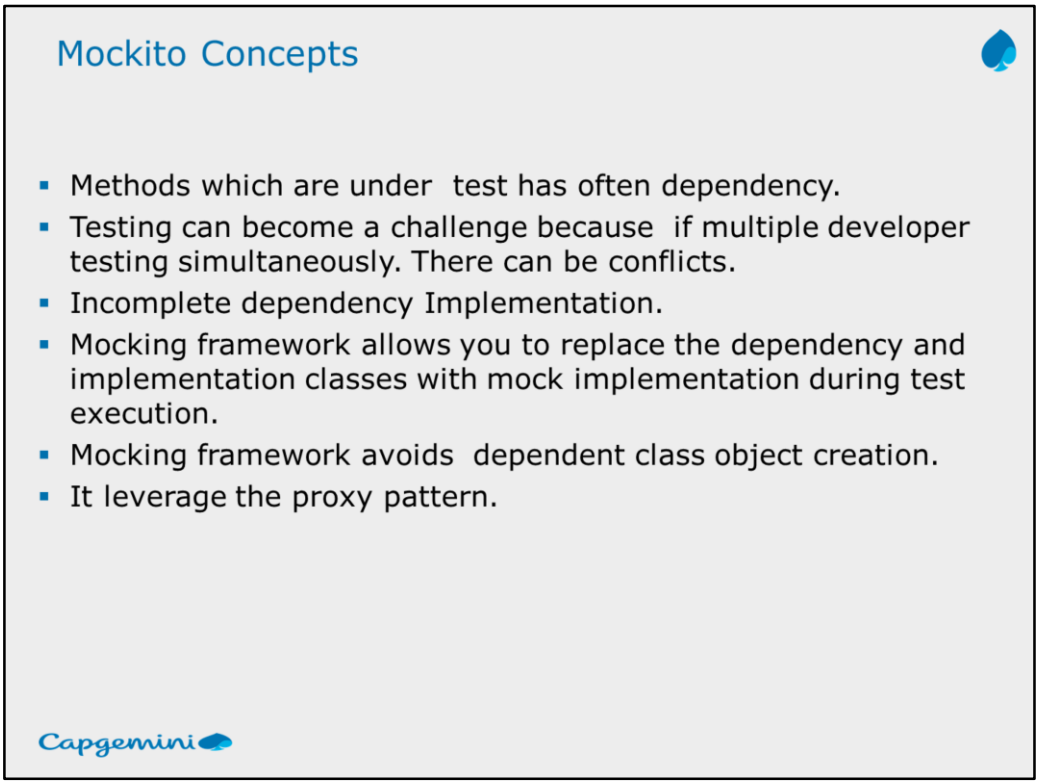
This lesson covers the concept of test doubles and explains various test double types, such as mock, fake, dummy, stub, and spy.
Sometimes, it is not possible to unit test a piece of code because of unavailability of collaborator objects or the cost of instantiation for the collaborator.
Test doubles alleviate the need for a collaborator.
We know about stunt doubles—a trained replacement used for dangerous action sequences in movies, such as jumping out of the
Empire State building, a fight sequence on top of a burning train, jumping from an airplane,
or similar actions. Stunt doubles are used to protect the real actors or chip in when the actor is not available.

While testing a class that communicates with an API, you don't want to hit the API for every single test;
for example, when a piece of code is dependent on database access, it is not possible to unit test the code unless the database is accessible.
Similarly, while testing a class that communicates with a payment gateway, you can't submit payments to a real payment gateway to run tests.

Test doubles act as stunt doubles. They are skilled replacements for collaborator objects.
Gerard Meszaros coined the term test doubles and explained test doubles in his book *xUnit Test Patterns*, *Pearson Education*.

Test doubles are categorized into five types. The above diagram shows these types:

Instructor Notes:



- Methods which are under test has often has dependency.
  -Ex Service layer depends of Dao using JPA API. So testing with dependencies is a big challenge because of live database we may require.

- Multiple developer Testing simultaneously
  Testing can become a challenge because multiple developer testing simultaneously. There can be conflicts
  -There might me a challenge to multiple developer testing data access object independently.
  -But we may not want conflicts to happen while testing service since service is dependent on DAO.

- Another problem is Incomplete dependency implementation.
  -Here dependent component is not yet developed-We may have a contract of interface defined but not the implantation developed.

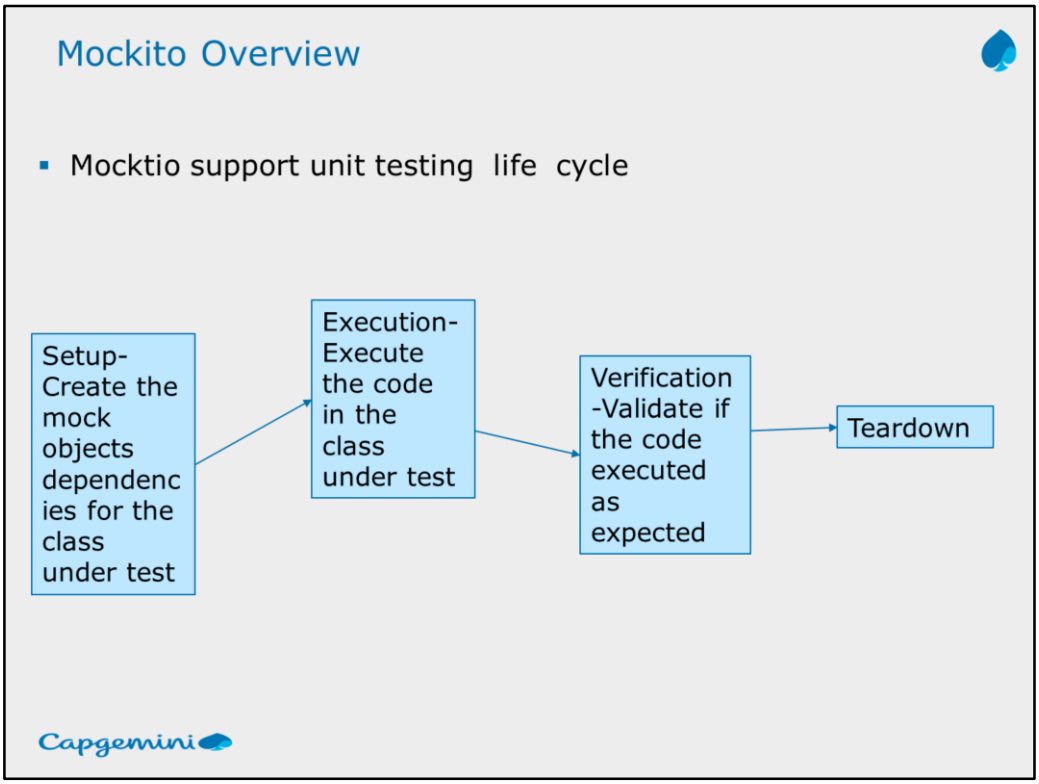
Instructor Notes:

Benefits Of Mockito

- **No Handwriting** – No need to write mock objects on your own.
- **Refactoring Safe** – Renaming interface method names or reordering parameters will not break the test code as Mocks are created at runtime.
- **Return value support** – Supports return values.
- **Exception support** – Supports exceptions.
- **Order check support** – Supports check on order of method calls.
- **Annotation support** – Supports creating mocks using annotation.

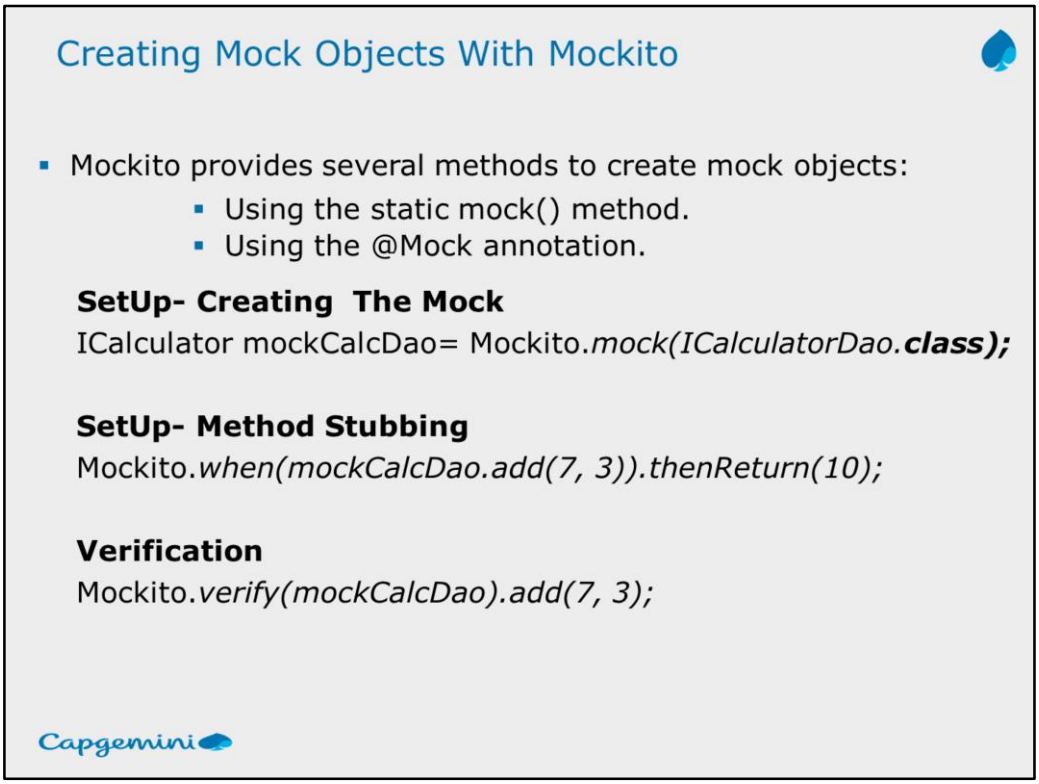
Capgemini

Instructor Notes:



- **Setup**- In this phase we ask the framework to create the dependency using mock objects.
- **Execution** – During execution mocks go in to response to a method under test.
- **Verification**-provide capability to ensure that mock behave in the manner you intended.

Instructor Notes:

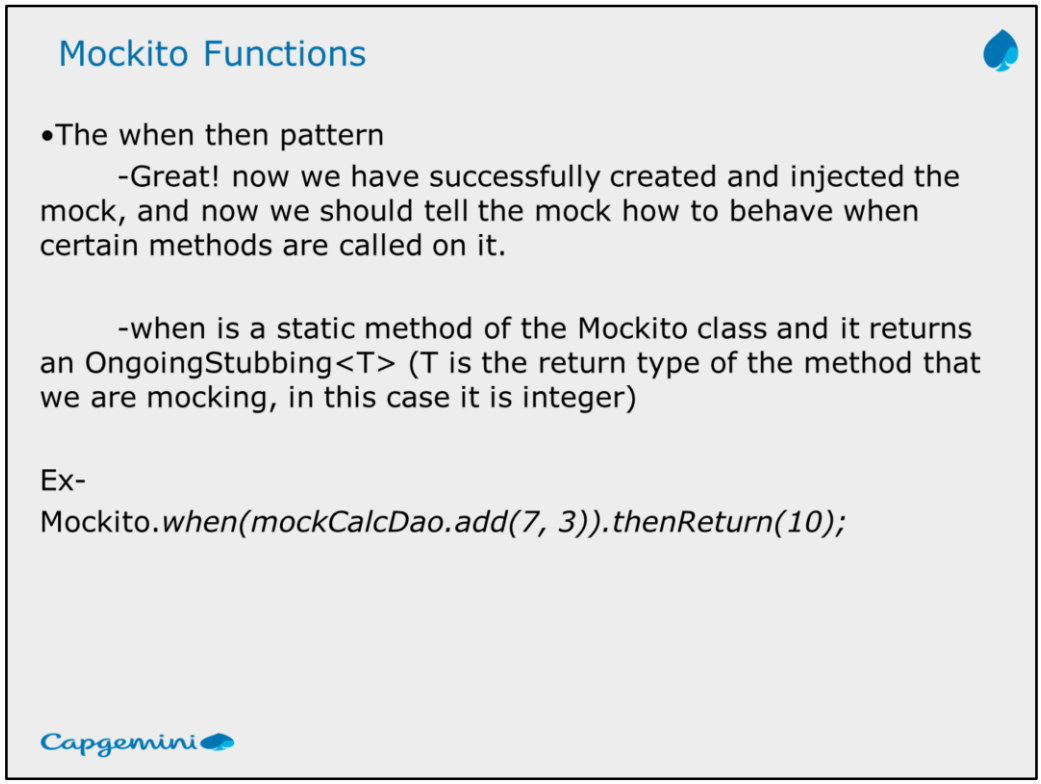


SetUp- Creating The Mock -

SetUp- Method Stubbing- It follows when then pattern . It specify how the operation behave when it is called with specific set of values.
We don't do anything special in the execution phase.

Verification –You user Mockito verify method to assert that particular method was called with a matched set of inputs.

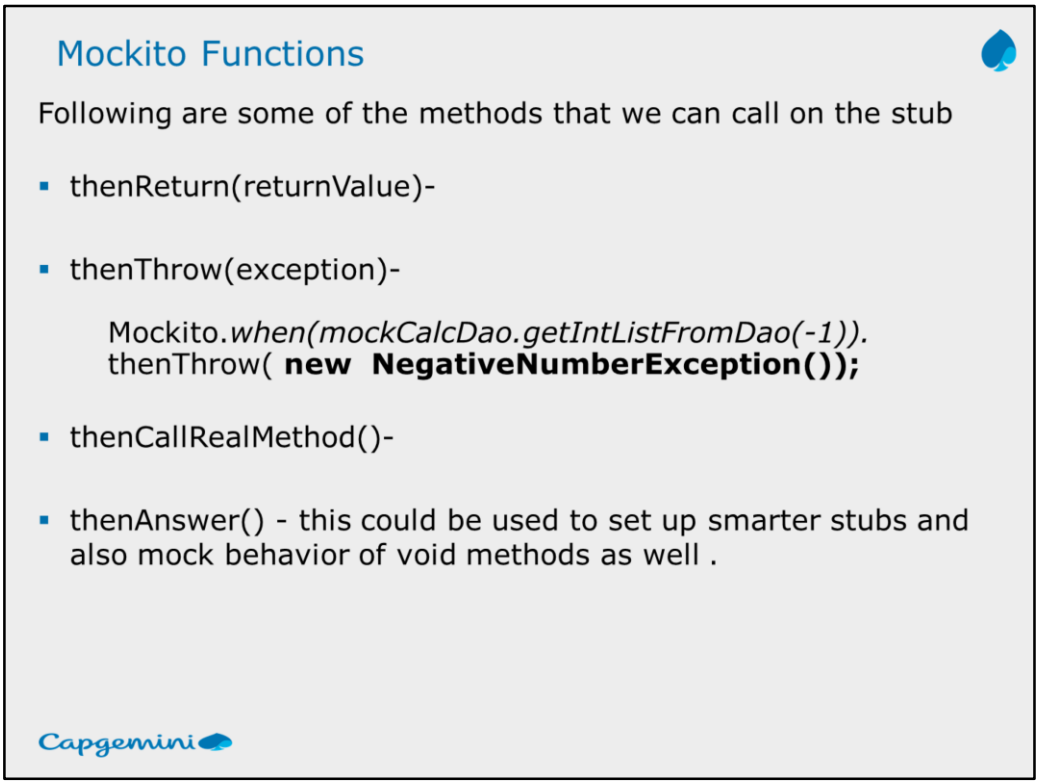
Instructor Notes:



Mockito.*when(mockCalcDao.add(7, 3)).thenReturn(10);*

-the above line of code tells the Mockito framework that we want the add() method of the mock dao instance to return 10 when 7 and 3 is passed as parameter

Instructor Notes:



- **thenReturn(returnValue)-** Specify object or value to return when method is called
- **thenThrow(exception)-** Specify mock invocation should result in exception thrown.
- **doThrow(..)** -If we need to throws exception when a method whose return type is void is called, then we can use the alternate way of throwing exception , i.e. doThrow(..) of class org.mockito.Mockito

- **thenCallRealMethod()-** When we are mocking a class then delegate call to underlying instance with thenCallRealMethod()
- **thenAnswer() –** answering allows you to provide means to conditionally respond based on mock operation parameter.

- Methods with return values can be tested by asserting the returned value, but how to test void methods? The void method that you want to test could either be calling other methods to get things done or processing the input parameters or maybe generating some values or all of it.
- With Mockito, you can test all of the above scenarios.

Instructor Notes:

Mockito Functions-Verification

- Mockito.verify(..) is used to verify an intended mock operation was called
- VerificationMode allows extra verification of the operation
  - times(*n*)
  - atLeastOnce()
  - atLeast(*n*)
  - atMost(*n*)
  - never()
- Verifying no interactions globally
  - Mockito.verify(..).zeroInteractions()

Capgemini

Instructor Notes:

Question 1: Option 2
Question 2: True

Review Questions

- Question 1: Spring IO _____ layer provides API to connect to cloud services
  - Option 1: Foundation
  - Option 2: Coordination
  - Option 3: Execution
- Question 2: Spring Boot reduces the effort needed to create production-ready, DevOps-friendly, XML-free Spring applications.
  - Option 1: True
  - Option 2: false

Capgemini