**Mockito Framework**

**Understanding test doubles**

A stunt double or dummy is a trained replacement used for dangerous action sequences in movies, such as a fight sequence on the top of a burning train, jumping from an airplane, and so on, mainly fight scenes. Stunt doubles are used to protect the real actors, are used when the actor is not available, or when the actor has a contract to not get involved in stunts.

Similarly, sometimes it is not possible to unit test the code because of the unavailability of the collaborator objects, or the cost of interaction and instantiation of collaborators. For instance, when the code is dependent on database access, it is not possible to unit test the code unless the database is available, or when a piece of code needs to send information to a printer and the machine is not connected to a LAN. The primary reason for using doubles is to isolate the unit you are testing from the external dependencies.

**Different types of test doubles**

* **Dummy**: This is an object that is used only for the code to compile—it doesn't have any business logic (for example, an object passed as a parameter to a method)
* **Fake:** This is an object that has an implementation but it's not production ready (for example, using an in-memory database instead of communicating with a standalone one)
* **Stub:** This is an object that has predefined answers to method executions made during the test
* **Mock**: This is an object that has predefined answers to method executions made during the test and has recorded expectations of these executions
* **Spy**: These are objects that are similar to stubs, but they additionally record how they were executed (for example, a service that holds a record of the number of sent messages)

**Adding Mockito to a project's classpath**

Adding Mockito to a project's classpath is as simple as adding one of the two jars to your project's classpath:

**1. mockito-all:** This is a single jar with all dependencies (with the hamcrest and objenesis libraries—as of June 2011).

**2. mockito-core:** This is only Mockito core (without hamcrest or objenesis). Use this if you want to control which version of hamcrest or objenesis is used.

Mockito has its own test runner implementation that allows you to reduce boilerplate in order to create test doubles (mocks and spies) and to inject them (either via constructors, setters, or reflection) into the defined object.

**@Mock**: This is used for mock creation

**@Spy:** This is used to create a spy instance

**@InjectMocks:** This is used to instantiate the @InjectMock annotated field and inject all the @Mock or @Spy annotated fields into it (if applicable)

**@Captor:** This is used to create an argument captor

If your Junit test case already using **@Runwith** Annotation you need to call below code snippet in the @Before annotated method.

@Before

public void setup () {

MockitoAnnotations.initMocks (this);

}