1. Testable App.
2. Mockito.
3. **Junit Test**: Noun
   1. Also called **Unit Test.**
   2. JUnit Test Cases.
4. **Unit Testing**: Action
   1. **Unit**: Methods.
   2. **Web App**: Contains all the methods so all the units.
5. **Archetype**:
   1. A way of creating **Sample Maven Project**.
   2. Archetype is a **Maven Project Templating toolkit.**
6. **Naming Convention**:
   1. Case01 🡺 The class to be tested + “Test”
   2. Case02🡺 If for one class, more than one test class, then different naming convention.
7. **Annotations:**
   1. Under **org.junit package**.
      1. **@Test**
         1. **@Test(expected=RuntimeException.class)**
         2. **@Test(timeout=<just\_number\_in\_milliseconds>)**
8. **Methods**:
   1. Under **package org.junit.Assert**
      1. **fail**(“msg”);
      2. **assertEquals**(expected, actual); 🡸 **Overloaded**
      3. **assertFalse(condition);**
      4. **assertTrue(condition);**
      5. **assertArrayEquals(expected, actual)**
   2. Under **package org.junit**
      1. @Before
      2. @After
      3. @BeforeClass
      4. @AfterClass
9. **Runner**
   1. **Parameterized Runner**

**@RunWith(Parameterized.class)**

public class **StringHelperParameterizedTest** {

StringHelper helper = new StringHelper();

private String input;

private String expected;

public **StringHelperParameterizedTest**(String input, String expected) {**//Constructor**

this.input = input;

this.expected = expected;

}

**@Parameters**

public static Collection<String[]> getParameters(){ **//Configuring Parameters**

String expectedOutputs[][]= { {"AACD", "AB"}, {"AEF", "CD"} };

return Arrays.asList(expectedOutputs);

}

**@Test**

public void truncateFirstAInFirstTwoPosition() { **//This test method will be called for each parameter.**

assertEquals(expected, helper.truncateAInFirst2Positions(input));

}

}

* 1. **Suite Runner**

**@RunWith(Suite.class)**

**@SuiteClasses**({ StringHelperParameterizedTest.class, StringHelperTest.class })

public class AllTests {

}



1. **JUnit Guidelines**
   1. Test methods should be **public void** otherwise **initialization error**.
   2. One test method should not have more than one assert statement or condition.
   3. **Method Naming Convention**:  
      **Template: “test”** + **“methodNameToBeTested”** + **“\_ConditionToBeTested”  
      Example:   
      “test”** + **“TruncateAInFirst2Positions()” + “\_AInFirstTwoPositions”  
      testTruncateAInFirst2Positions\_AInFirstTwoPositions()**
   4. If a class has dependency on other class object, create constructor and inject the dependency.   
      Doing so will make the dependent class to be testable as we can inject the mock dependency.  
      Dependency for SUT (**S**ystem **U**nder **T**est)

Mockito

1. **SUT**: **S**ystem **U**nder **T**est.
   1. Refers to **a system** which is being validated by the **testers**.
2. **Stub**: A sample Implementation.
3. **Mocking** is creating object that simulates the behavior of a real object.
4. **How to say:**
   1. Mocking a dependency.
5. **Mockito class methods:**
   1. **mock**(<class\_or\_interface>)
   2. **when**(mockObject.method(parameterList)).**return**(def\_return\_value); 🡺 To stub a method.
6. **EasyMock**: Doesn’t return default values if called method on mock is not stubbed.
7. **Mocking Examples**:
   1. Mockito.**mock**(<interface\_or\_class\_.class>)
   2. Mockito.**when**(mockObj.methodName(<param\_list>).**thenReturn**(<return\_value>);
   3. **Returning different values in subsequent call**.  
      Mockito.**when**(mockObj.methodName(<param\_list>)  
      .**thenReturn**(<return\_value\_1>)  
      .**thenReturn**(<return\_value\_2>)
   4. Mocking java.util.**List** interface.
      1. Mockito.mock(java.util.**List);**
      2. **when**(mockList.**get(0)**).thenReturn(<return\_value>);
      3. **when**(mockList.**get(0)**).thenReturn(<return\_value>);
      4. **when**(mockList.**get**(**Mockito.anyInt()**)).thenReturn(<return\_value>);
   5. Throwing **an exception**:
      1. when(mockObj.methodName(<param>).thenThrow(<Any\_Exception.class>);

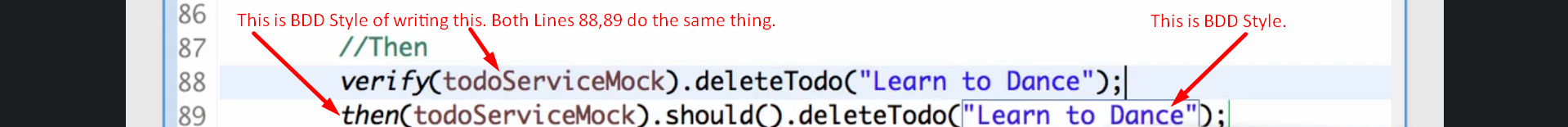
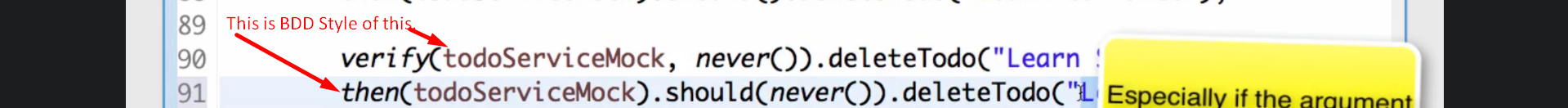
BDD Behavior Driven Development.

1. Agile projects use user-story. Create Scenario around the Story.
2. **G**iven-**W**hen-**T**hen.  
   **Given**: Set up, **When**: Actual Method Call, **Then**: Assertions.
3. **BDD Mockito**.
4. Graphical user interface, text, application, chat or text message

   Description automatically generatedGraphical user interface, text, application

   Description automatically generated

Checking if a method is called or not and how many times and with what arguments.

1. **verify**(todoServiceMock).deleteToDo(“Learn to Dance”);  
   Following Snapshot is BDD Style of the above.  
     
   **verify**(todoServiceMock, **never**()).deleteToDo(“Learn to Dance”);  
   The following snapshot is BDD Style of the above.  
     
   **verify**(todoServiceMock, **times**(2)).deleteToDo(“Learn to Dance”);  
   **verify**(todoServiceMock, **atLeastOnce**()).deleteToDo(“Learn to Dance”);  
   **verify**(todoServiceMock, **atLeast(5)**).deleteToDo(“Learn to Dance”);

Argument Captor

1. Text

   Description automatically generated
2. Capturing more than one value.  
   Graphical user interface, text, application

   Description automatically generated
3. **To capture multiple arguments.**  
   BDDMockito.then(todoService).should().deleteTodo(argumentCaptor.capture());