1.JUNIT Code

a) Create a sample class for any method(here it's sum of array)
b) Add dependencies of junit (included in spring-boot-starter-test)
c) Create a test class and inject sample class in it for testing.
d) Add all testing methods with @Test annotation on it.

```java
public class SomeBusinessImpl {

    public int calculateSum(int
[] data) {
        int sum = 0;
        for (int value : data) {
            sum += value;
        }
        return sum;
    }
}
```

```java
public class SomeBusinessTest {
    @Test
    public void calculateSunTest() {
        SomeBusinessImpl sbi = new SomeBusinessImpl();
        int actualResult = sbi.calculateSum(new int[]{1, 2, 3, 4});
        int expectedResult = 10;
        Assertions.assertEquals(expectedResult, actualResult);

    }

    @Test
    public void calculateSunTest_Empty() {
        SomeBusinessImpl sbi = new SomeBusinessImpl();
        int actualResult = sbi.calculateSum(new int[]{});
        int expectedResult = 0;
        Assertions.assertEquals(expectedResult, actualResult);

    }

    @Test
    public void calculateSunTest_One() {
        SomeBusinessImpl sbi = new SomeBusinessImpl();
        int actualResult = sbi.calculateSum(new int[]{1});
        int expectedResult = 1;
        Assertions.assertEquals(expectedResult, actualResult);

    }
}
```

2.JUNIT Code

a) What if we want to take the data from data Service ? we need to inject that dataService.
b) So, create an interface and implement stub classes for each instance.
c) Through setter set the data and mock it.

```java
public class SomeBusinessImpl {
    private SomeDataService someDataService;

    public void setSomeDataService(SomeDataService someDataService) {
        this.someDataService = someDataService;
    }

    public int calculateSumUsingDataService() {
        int sum = 0;
        int[] data = someDataService.retrieveAllData();
        for(int value: data) {
            sum += value;
        }
        return sum;
    }
}
```

```java
public interface SomeDataService {
    int[] retrieveAllData();
}
```

```java
class SomeDataServiceStub implements
SomeDataService {

    @Override
    public int[] retrieveAllData() {
        return new int[] {1,2,3};
    }
}
class EmptyDataServiceStub implements
SomeDataService {

    @Override
    public int[] retrieveAllData() {
        return new int[] {};
    }
}
class OneDataServiceStub implements
SomeDataService {

    @Override
    public int[] retrieveAllData() {
        return new int[] {1};
    }
}
```

```java
    @Test
    public void calculatesUsingDataSercviceTest() {
        SomeBusinessImpl sbi = new SomeBusinessImpl();
        sbi.setSomeDataService(new SomeDataServiceStub());
        int actualResult = sbi.calculateSumUsingDataService();
        int expectedResult = 6;
        Assertions.assertEquals(expectedResult, actualResult);
    }
    @Test
    public void calculatesUsingEmptyDataSercviceTest() {
        SomeBusinessImpl sbi = new SomeBusinessImpl();
        sbi.setSomeDataService(new EmptyDataServiceStub());
        int actualResult = sbi.calculateSumUsingDataService();
        int expectedResult = 0;
        Assertions.assertEquals(expectedResult, actualResult);
    }
    @Test
    public void calculatesUsingOneDataSercviceTest() {
        SomeBusinessImpl sbi = new SomeBusinessImpl();
        sbi.setSomeDataService(new OneDataServiceStub());
        int actualResult = sbi.calculateSumUsingDataService();
        int expectedResult = 1;
        Assertions.assertEquals(expectedResult, actualResult);
    }
}
```

It was making code more complex i.e., create separate class for each test to send the data hence the Mockito came into existence.

MOCKITO

Mockito is a **mocking framework** for Java that lets you write tests by creating **mock objects** of classes and interfaces, so you can evaluate the behaviour of your code in isolation.

Instead of creating a new class for each test case we create a mock for each test case.

a) Add Mockito dependency and create a mock for interface(mock is the static method in **Mockito** class).
b) Pass that mocking class to set the dataService.
c) Use when(method called).thenReturn(data) mock in test method.

```java
public class SomeBusinessImpl {
    private SomeDataService someDataService;

    public void setSomeDataService(SomeDataService someDataService) {
        this.someDataService = someDataService;
    }

    public int calculateSumUsingDataService() {
        int sum = 0;
        int[] data = someDataService.retrieveAllData();
        for(int value: data) {
            sum += value;
        }
        return sum;
    }
}
```

```java
public interface SomeDataService {
    int[] retrieveAllData();
}
```

```java
public class SomeBusinessMockTest {

    SomeBusinessImpl sbi = new SomeBusinessImpl();
    SomeDataService dataServiceMock = mock(SomeDataService.class);

    @BeforeEach
    public void before() {
        sbi.setSomeDataService(dataServiceMock);
    }

    @Test
    public void calculatesUsingDataSercviceTest() {

        when(dataServiceMock.retrieveAllData()).thenReturn(new int[] {1,2,3});
        Assertions.assertEquals(6, sbi.calculateSumUsingDataService());
    }

    @Test
    public void calculatesUsingEmptyDataSercviceTest() {
        when(dataServiceMock.retrieveAllData()).thenReturn(new int[] {});
        Assertions.assertEquals(0,sbi.calculateSumUsingDataService());
    }
    @Test
    public void calculatesUsingOneDataSercviceTest() {

        when(dataServiceMock.retrieveAllData()).thenReturn(new int[] {1});
        Assertions.assertEquals(1, sbi.calculateSumUsingDataService());
    }
}
```

Instead of creating a mock and calling a setter method each time we can use some annotations so that code looks easy.

@Mock – to create mock (upon reference)

@InjectMock – to inject Mock (on object)

@ExtendMock (on class) – this passes the mock through setter before each test.

```java
@ExtendWith(MockitoExtension.class)
public class SomeBusinessMockTest {
    @InjectMocks
    SomeBusinessImpl sbi = new SomeBusinessImpl();

    @Mock
    SomeDataService dataServiceMock;

    @Test
    public void calculatesUsingDataSercviceTest() {

        when(dataServiceMock.retrieveAllData()).thenReturn(new int[] {1,2,3});
        Assertions.assertEquals(6, sbi.calculateSumUsingDataService());
    }

    @Test
    public void calculatesUsingEmptyDataSercviceTest() {
        when(dataServiceMock.retrieveAllData()).thenReturn(new int[] {});
        Assertions.assertEquals(0,sbi.calculateSumUsingDataService());
    }
    @Test
    public void calculatesUsingOneDataSercviceTest() {

        when(dataServiceMock.retrieveAllData()).thenReturn(new int[] {1});
        Assertions.assertEquals(1, sbi.calculateSumUsingDataService());
    }
}
```

Multiple return Value and with paramater

For same mock we can return multiple value 1st return 1st value and 2nd return 2nd value.

List is the Collection interface we used here to Mock.

```java
public class ListMockTest {
    @Test
    public void size_basic() {
        List mock = mock(List.class);
        when(mock.size()).thenReturn(5);
        assertEquals(5, mock.size());
    }
    @Test
    public void size_multipleReturns() {
        List mock = mock(List.class);
        when(mock.size()).thenReturn(5).thenReturn(10); //multiple
        assertEquals(5, mock.size());
        assertEquals(10, mock.size());
    }
    @Test
    public void retrunWithParameters() {
        List mock = mock(List.class);
        when(mock.get(0)).thenReturn("in28Minutes"); // with parameter
        assertEquals("in28Minutes", mock.get(0));
        assertEquals(null, mock.get(1));
    }
}
```

Generic parameter

There are arguments matcher in Mockito that matches generic parameter like.

- anyInt()
- anyChar()
- anyLong()

```java
@Test
public void returnWithGenericParameters() {
    List mock = mock(List.class);
    // anyInt() is a generic matcher that matches any integer value - Argument Matcher
    when(mock.get(anyInt())).thenReturn("in28Minutes");
    assertEquals("in28Minutes", mock.get(0));
    assertEquals("in28Minutes", mock.get(1));
}
```

Verification

Till now all method the methods are returning a value how to test if the method does not return any value and calls another method inside it. Here comes verification,

Check whether the mock calls the method we use verify method of Mockito.

```java
@Test
public  void verificationBasics() {
    List<String> mock = mock(List.class);
    String value1 = mock.get(0);
    String value2 = mock.get(1);
    verify(mock).get(0);
    verify(mock).get(1);
    verify(mock, times(2)).get(anyInt());
    verify(mock, atLeast(2)).get(anyInt());
    verify(mock, never()).get(2);
}
```

Argument Capture

Check what is the argument passed into the Method.

```java
@Test
public void captureArguments() {
    List<String> mock = mock(List.class);
    mock.add("SomeString");
    ArgumentCaptor<String> captor = ArgumentCaptor.forClass(String.class);
    verify(mock).add(captor.capture());
    assertEquals("SomeString", captor.getValue());
}
```

For multiple argument checking.

```java
@Test
public void MultipleCaptureArguments() {
    List<String> mock = mock(List.class);
    mock.add("SomeString1");
    mock.add("SomeString2");
    ArgumentCaptor<String> captor = ArgumentCaptor.forClass(String.class);
    verify(mock, times(2)).add(captor.capture());
    List<String> allValues = captor.getAllValues();
    assertEquals("SomeString1", allValues.get(0));
    assertEquals("SomeString2", allValues.get(1));
}
```

Spy

Spying is just like real world spying is you let the action to happen, but you keep a watch on it. Mocking is instead of the real world action.

```java
@Test
public void spyBasics() {
    // Spy - real object but you can stub specific methods
    List<String> listSpy = spy(List.class);
    listSpy.add("Test0");
    // Default behaviour of spy calls the real methods
    System.out.println(listSpy.get(0));
    System.out.println(listSpy.size());
    listSpy.add("Test1");
    listSpy.add("Test2");
    System.out.println(listSpy.size());
    when(listSpy.size()).thenReturn(5);
    System.out.println(listSpy.size());
    listSpy.add("Test3");
    // Final size method will still return 5
    System.out.println(listSpy.size());
    verify(listSpy).add("Test3");
}
```