Unit Testing

Q)What is Unit ?
Ans) Unit is Method or group of method which needs to be tested

Q)WHat is Unit Testing ?
Ans) A functionality to be tested among its various true and false cases response to verify if its working appropriately

Note :1)If junit method to be tested is empty ..it will be shown as passed test case(green)
2)Method annotated with @Test has to be void and public ..Otherwise it will throw run time exception

Assert Syntax : assertEquals(expected, actual);
Note @BeforeClass and @AfterClass Methods has to be "static"

Points Covered Below
1.If In same Test Block we have more than one assert statment then if any assert fails then it will be considered as failed test case
2.Empty test will always be considered as Passed
3.If we want to test methods that returns true or false ..then we can use assertTrue and assertFalse
assertEquals(true,helper.isHappy("happy")); -->
assertTrue(helper.isHappy("happy"));
assertEquals(false,helper.isHappy("kappy"));-->
assertFalse(helper.isHappy("kappy"));
4.Comments will be shown in stack trace when this test fails ..If test pass it will not be shown
5.If we want to execute any block of statment just before and after each test we use @Before and @After
6.If we want to execute something Before All the Test and After all the Test ...But at once only we use @BeforeClass and @AfterClass

7.To test Arrays we use assertArrayEquals(expected_array,actual_array)
8.Testing Exceptions ..Must Read Examples ... Use @Test(expected = NullPointerException.class)
9.To test performance we use @Test(timeout = 1000) where Time is provided in millliseconds

Code Example 1:

```
public class StringHelper {

	public String getFirstChar(String str) {
		return str.substring(0,1);
	}

	public boolean isHappy(String str) {
		return str.equalsIgnoreCase("happy");
	}

}

public class StringHelperTest {

	StringHelper helper=new StringHelper();

	@Before
```

```
	public void randomNameBefore() {
		System.out.println("Before test"); ;
	}

	@After
	public void randomNameAfter() {
		System.out.println("After test"); ;
	}


	** In Below Test case 1 asserts is true and other is false
..finally it will be considered as failed test case
	@Test
	public void testingGetFirstChar() {
		assertEquals("Q",helper.getFirstChar("Pawan"));
		assertEquals("P",helper.getFirstChar("Pawan"));
	}

	** Empty test will always be considered as Passed
	@Test
	public void emptyTest() {
	}

	** If Output is True then use below code of assertTrue
	@Test
	public void assertTrueTesting() {
		///assertEquals(true,helper.isHappy("happy"));
		assertTrue(helper.isHappy("happy"));
	}

	** If Output is True then use below code of assertFalse
	@Test
	public void assertFalseTesting() {
		//assertEquals(false,helper.isHappy("kappy"));
		assertFalse(helper.isHappy("kappy"));
	}

	** Comments will be shown in stack trace when this test fails ..If
test pass it will not be shown
	@Test
	public void assertTrueTestingWithComments() {
		///assertEquals(true,helper.isHappy("happy"));
		assertTrue("passed hai !!",helper.isHappy("kappy"));
	}
}
```

Code Example 2:

```
	@BeforeClass
	public static void randomNameBeforeClass() {
		System.out.println("Before Class"); ;
	}

	@AfterClass
	public static void randomNameAfterClass() {
		System.out.println("After Class"); ;
	}

	@Before
```

```
	public void randomNameBefore() {
		System.out.println("Before test"); ;
	}


	@Test
	public void test1() {
		System.out.println("Test1");
	}
	@Test
	public void test2() {
		System.out.println("Test2");
	}

	@After
	public void randomNameAfter() {
		System.out.println("After test"); ;
```

Output =>

```
		Before Class
		Before test
		Test1
		After test
		Before test
		Test2
		After test
		After Class
```

Code Example 3: Testing Arrays and Exception

```
	** To Test Arrays Comparision
	@Test
	public void testArraySorted() {
		int[] numbers= {12,3,4,1};
		int[] expected= {1,3,4,12};
		Arrays.sort(numbers);
		assertArrayEquals(expected, numbers);
	}


	** Arrays Exception Testing
	@Test
	public void testArraySorted1() {
		int[] numbers=null;
		try {
		Arrays.sort(numbers);
		}catch (NullPointerException e) {
			// Success
		}
	}

	** Arrays Exception Testing
	@Test(expected = NullPointerException.class)
	public void testArraySorted2() {
		int[] numbers=null;
		Arrays.sort(numbers);

	}
```

```
	@Test(expected = NullPointerException.class)
	public void testArraySorted3() {
		int[] numbers= {};
		Arrays.sort(numbers);

	}
```

Code Example 4 :Performance Test

```
 ** Time is provided in millliseconds
	@Test(timeout = 1000)
	public void testPerformance() {
		int[] arr= {12,2,16};
		for(int i=0;i<=1000000;i++) {
			arr[0]=i;
			Arrays.sort(arr);
		}
	}
```

Code Example 5: Instead Of Writing Test Positive and Negative Test cases for same @Test Method and checking them for various input and expected outputs we can use below Steps

1.Use Class Level Annotation @RunWith(Parameterized.class)

2.Create method annotated with @Parameters that returns input and output as Collections

3. Create istance variables input and expectedOutput + Constructor

```
@RunWith(Parameterized.class)
public class StringHelperParameterizedTest {

	// AACD => CD ACD => CD CDEF=>CDEF CDAA => CDAA

	StringHelper helper = new StringHelper();

	private String input;
	private String expectedOutput;

	public StringHelperParameterizedTest(String input, String
expectedOutput) {
		this.input = input;
		this.expectedOutput = expectedOutput;
	}

	@Parameters
	public static Collection<String[]> testConditions() {
		String expectedOutputs[][] = {
				{ "AACD", "CD" },
				{ "ACD", "CD" } };
		return Arrays.asList(expectedOutputs);
	}

	@Test
	public void testTruncateAInFirst2Positions() {

	assertEquals(expectedOutput,helper.truncateAInFirst2Positions(input
));
	}
}
```

Q)What is Suite and Why it is Required ?
Ans) Suppose We have 4 test classes and we want to run any 2 classes containing test cases then we use suite
SUite is used to run group of test classes

```
@RunWith(Suite.class)
@SuiteClasses({ArraysTest.class,StringHelperTest.class})
public class DummyTestSuite {

}
```

==========Scenario ==========

```
public interface TodoService {

	public List<String> retrieveTodos(String user);

	void deleteTodo(String todo);

}

public class TodoBusinessImpl {
	private TodoService todoService;

	TodoBusinessImpl(TodoService todoService) {
		this.todoService = todoService;
	}

	public List<String> retrieveTodosRelatedToSpring(String user) {
		List<String> filteredTodos = new ArrayList<String>();
		List<String> allTodos = todoService.retrieveTodos(user);
		for (String todo : allTodos) {
			if (todo.contains("Spring")) {
				filteredTodos.add(todo);
			}
		}
		return filteredTodos;
	}

	public void deleteTodosNotRelatedToSpring(String user) {
		List<String> allTodos = todoService.retrieveTodos(user);
		for (String todo : allTodos) {
			if (!todo.contains("Spring")) {
				todoService.deleteTodo(todo);
			}
		}
	}
}
```

==========Testing with Stub ==========
QWhat is Stub ?
Ans)stub is an object that resembles a real object with the minimum number of methods needed for a test.
-It always returns the predefined output regardless of the input.

Disadvantage Of Stub :

1.As we have to test various test cases so we have to create multiple Stubs of same Interface...So it will become tedious process
2.As Interface to be stubbed has various methods... we have to handle various methods while creating stub class to keep compiler happy

Code Example :
1. Add Dependency : junit and mockito-all

2.Create Interface of which we have to make Stub: ****Focus****
package : src/main/java

```
	public interface TodoService {
		public List<String> retrieveTodos(String user);
	}
```

3.Create Class with is called System Under Test ******Focus****
package : src/main/java

```
public class TodoBusinessImpl {
	private TodoService todoService;

	public TodoBusinessImpl(TodoService todoService) {
		this.todoService = todoService;
	}

	public List<String> retrieveTodosRelatedToSpring(String user) {
		List<String> filteredTodos = new ArrayList<String>();
		List<String> allTodos = todoService.retrieveTodos(user);
		for (String todo : allTodos) {
			if (todo.contains("Spring")) {
				filteredTodos.add(todo);
			}
		}
		return filteredTodos;
	}
}
```

4. Create Stub Now
package : src/main/test

```
public class TodoServiceStub implements TodoService {

	@Override
	public List<String> retrieveTodos(String user) {
		return Arrays.asList("Learn Spring MVC", "Learn
Spring","Learn to Dance");
	}

}
```

5.Create System Under Test

```
public class TodoBusinessImplStubTest  {

	*********Understand Each line of Code here ********
	@Test
	public void usingAStub() {
		TodoService todoService = new TodoServiceStub();
```

```
            TodoBusinessImpl todoBusinessImpl = new
TodoBusinessImpl(todoService);
            List<String> todos =
todoBusinessImpl.retrieveTodosRelatedToSpring("Ranga");
            assertEquals(2, todos.size());
      }
}
```

==========Testing with Mockito ==========

Q) What is Mockito ?
Ans)Mocking is used to replicate behaviour of real objects
Unlike Stubs ,Mocks can be created dynamically at run time
it can verify method calls

Stub is real Object created | Mock is used to replicate behaviour of real object ..it is not real object

Q)explain mock () method ?
-mock() is method in Mockito Class
-We can use mock() with Interface or class

```
      @Test
      public void usingMockito() {
            TodoService todoService = mock(TodoService.class);
            List<String> allTodos = Arrays.asList("Learn Spring
MVC","Learn Spring", "Learn to Dance");

      when(todoService.retrieveTodos("Ranga")).thenReturn(allTodos);
            TodoBusinessImpl todoBusinessImpl = new
TodoBusinessImpl(todoService);
            List<String> todos =
todoBusinessImpl.retrieveTodosRelatedToSpring("Ranga");
            assertEquals(2, todos.size());
      }
```

Code Example : Mocking List Class Methods
1.Understand assertNull
2.Understand what happens for unmocked values..it sets with default values
3.Understand
anyInt(),anyBoolean(),any(),anyString(),anyCollection(),anySet(),anyMap()
...and many more

```
      @Test
      public void letsMockListSize() {
            List list = mock(List.class);
            Mockito.when(list.size()).thenReturn(10);
            assertEquals(10, list.size());
      }

      @Test
      public void letsMockListSizeWithMultipleReturnValues() {
            List list = mock(List.class);
            Mockito.when(list.size()).thenReturn(10).thenReturn(20);
            assertEquals(10, list.size()); // First Call
            assertEquals(20, list.size()); // Second Call
      }
```

```
	@Test
	public void letsMockListGet() {
		List<String> list = mock(List.class);
		Mockito.when(list.get(0)).thenReturn("in28Minutes");
		assertEquals("in28Minutes", list.get(0));
		assertNull(list.get(1));
	}

	@Test
	public void letsMockListGet() {
		List<String> list = mock(List.class);

		---Before Mocking--->
		assertEquals(null, list.get(0)); ---> By Default null is
present for non micket values ****
		aassertEquals(null, list.get(1));

		---After Mocking---->
		Mockito.when(list.get(0)).thenReturn("in28Minutes");
		assertEquals("in28Minutes", list.get(0));
		assertEquals(null, list.get(1)); ---> By Default null is
present for non micket values ****
		assertNull(list.get(1));          --> *****Another way Of
Checking Null
	}

	@Test(expected=RunTimeException.class)
	public void letsMockListGetWithAny() {
		List<String> list = mock(List.class);

	Mockito.when(list.get(Mockito.anyInt())).thenReturn("in28Minutes");
		// If you are using argument matchers, all arguments
		// have to be provided by matchers.
		assertEquals("in28Minutes", list.get(0));
		assertEquals("in28Minutes", list.get(1));
	}

	@Test
	public void mockException (){
	List list = mock(List.class);
	Mockito.when(list.get(Mockito.anyInt())).thenThrow(new
RuntimeException("Something"));

	list.get(0);
	}

------------------BDD - Behaviour Driven Development -----------
It has Given ,When and Then
Given --> do basic setup
When -> actual methpd call happens
Then --> what has to be expected

1.Understand assertThat(actual,matchers) syntax


	@Test
	public void usingMockitoBDD() {
```

```
            //Given
            TodoService todoService = mock(TodoService.class);
            List<String> allTodos = Arrays.asList("Learn Spring
MVC","Learn Spring", "Learn to Dance");

      given(todoService.retrieveTodos("Ranga")).willReturn(allTodos);

            //When
            TodoBusinessImpl todoBusinessImpl = new
TodoBusinessImpl(todoService);
            List<String> todos =
todoBusinessImpl.retrieveTodosRelatedToSpring("Ranga");

            //Then
            assertThat(todos.size(), is(2));
      }

      @Test
      public void bddAliases_UsingGivenWillReturn() {
            //given
            List<String> list = mock(List.class);
            given(list.get(Mockito.anyInt())).willReturn("in28Minutes");

            //When
            String firstElement=list.get(0);

            //then
            assertThat(firstElement, is("in28Minutes"));

      }

----verify()-----------
verify() is used to check if method is called or not

Code :
public interface TodoService {

      void deleteTodo(String todo);
}


public class TodoBusinessImpl {

      private TodoService todoService;

      public TodoBusinessImpl(TodoService todoService) {
            this.todoService = todoService;
      }

      public void deleteTodosNotRelatedToSpring(String user) {
            List<String> allTodos = todoService.retrieveTodos(user);
            for (String todo : allTodos) {
                  if (!todo.contains("Spring")) {
                        todoService.deleteTodo(todo);
                  }
            }
      }
}
```

```
	@Test
	public void letsTestDeleteNow() {

		TodoService todoService = mock(TodoService.class);
		List<String> allTodos = Arrays.asList("Learn Spring
MVC","Learn Spring", "Learn to Dance");

	when(todoService.retrieveTodos("Ranga")).thenReturn(allTodos);

		TodoBusinessImpl todoBusinessImpl = new
TodoBusinessImpl(todoService);
		todoBusinessImpl.deleteTodosNotRelatedToSpring("Ranga");

		verify(todoService).deleteTodo("Learn to Dance");
		verify(todoService, Mockito.never()).deleteTodo("Learn Spring
MVC");
		verify(todoService, Mockito.never()).deleteTodo("Learn
Spring");
		verify(todoService, Mockito.times(1)).deleteTodo("Learn to
Dance");		// atLeastOnce, atLeast
		verify(todoService, Mockito.atLeastOnce()).deleteTodo("Learn
to Dance");
		verify(todoService, Mockito.atLeast(2)).deleteTodo("Learn to
Dance");
	}

Note : Below statments can be written in either way :
1.verify(todoService).deleteTodo("Learn to Dance");
  then(todoService).should().deleteTodo("Learn to Dance");

2.verify(todoService, Mockito.never()).deleteTodo("Learn Spring MVC");
then(todoService).should(never()).deleteTodo("Learn to Dance");

3.verify(todoService, Mockito.times(1)).deleteTodo("Learn to Dance");
then(todoService).should(times(1)).deleteTodo("Learn to Dance");

-----------Capturing Mock Arguments ---------------
	@Test
	public void captureArgument() {
		ArgumentCaptor<String> argumentCaptor =
ArgumentCaptor.forClass(String.class);

		TodoService todoService = mock(TodoService.class);

		List<String> allTodos = Arrays.asList("Learn Spring
MVC","Learn Spring", "Learn to Dance");

	Mockito.when(todoService.retrieveTodos("Ranga")).thenReturn(allTodo
s);

		TodoBusinessImpl todoBusinessImpl = new
TodoBusinessImpl(todoService);
		todoBusinessImpl.deleteTodosNotRelatedToSpring("Ranga");

	Mockito.verify(todoService).deleteTodo(argumentCaptor.capture());
```

```
            assertEquals("Learn to Dance", argumentCaptor.getValue());

      }

      @Test
      public void captureArgument() {
            ArgumentCaptor<String> argumentCaptor =
ArgumentCaptor.forClass(String.class);

            TodoService todoService = mock(TodoService.class);

            List<String> allTodos = Arrays.asList("Learn Rock and
Roll","Learn Spring", "Learn to Dance");

      Mockito.when(todoService.retrieveTodos("Ranga")).thenReturn(allTodo
s);

            TodoBusinessImpl todoBusinessImpl = new
TodoBusinessImpl(todoService);
            todoBusinessImpl.deleteTodosNotRelatedToSpring("Ranga");

      then(todoService).should(times(2y)).deleteTodo(argumentCaptor.captu
re());

            assertThat( argumentCaptor.getAllValues().size(),is(2));
      }

====================================Hamcrest asserts ========
1.add dependency
      <dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-library</artifactId>
            <version>1.3</version>
            <scope>test</scope>
      </dependency>

2.Create test
      @Test
      public void basicHamcrestMatchers() {
            List<Integer> scores = Arrays.asList(99, 100, 101, 105);
            assertThat(scores, hasSize(4));
            assertThat(scores, hasItems(100, 101));
            assertThat(scores, everyItem(greaterThan(90)));
            assertThat(scores, everyItem(lessThan(200)));

            // String
            assertThat("", isEmptyString());
            assertThat(null, isEmptyOrNullString());

            // Array
            Integer[] marks = { 1, 2, 3 };

            assertThat(marks, arrayWithSize(3));
            assertThat(marks, arrayContainingInAnyOrder(2, 3, 1));

      }
```

```
=====================================Mockito Using Annotations
======================
Mockito Annotations
@Mock
@InjectMocks
@RunWith(MockitoJUnitRunner.class)
@Captor

@RunWith(MockitoJUnitRunner.class)
public class TodoBusinessImplMockitoInjectMocksTest {
	@Mock
	TodoService todoService;

	@InjectMocks
	TodoBusinessImpl todoBusinessImpl;

	@Captor
	ArgumentCaptor<String> stringArgumentCaptor;

	@Test
	public void usingMockito() {
		List<String> allTodos = Arrays.asList("Learn Spring
MVC","Learn Spring", "Learn to Dance");


	when(todoService.retrieveTodos("Ranga")).thenReturn(allTodos);

		List<String> todos =
todoBusinessImpl.retrieveTodosRelatedToSpring("Ranga");
		assertEquals(2, todos.size());
	}

	@Test
	public void usingMockito_UsingBDD() {
		List<String> allTodos = Arrays.asList("Learn Spring
MVC","Learn Spring", "Learn to Dance");

		//given

	given(todoService.retrieveTodos("Ranga")).willReturn(allTodos);

		//when
		List<String> todos =
todoBusinessImpl.retrieveTodosRelatedToSpring("Ranga");

		//then
		assertThat(todos.size(), is(2));
	}

	@Test
	public void letsTestDeleteNow() {

		List<String> allTodos = Arrays.asList("Learn Spring
MVC","Learn Spring", "Learn to Dance");


	when(todoService.retrieveTodos("Ranga")).thenReturn(allTodos);
```

```
            todoBusinessImpl.deleteTodosNotRelatedToSpring("Ranga");

            verify(todoService).deleteTodo("Learn to Dance");

            verify(todoService, Mockito.never()).deleteTodo("Learn Spring
MVC");

            verify(todoService, Mockito.never()).deleteTodo("Learn
Spring");

            verify(todoService, Mockito.times(1)).deleteTodo("Learn to
Dance");
            // atLeastOnce, atLeast

      }

      @Test
      public void captureArgument() {
            List<String> allTodos = Arrays.asList("Learn Spring
MVC","Learn Spring", "Learn to Dance");

      Mockito.when(todoService.retrieveTodos("Ranga")).thenReturn(allTodo
s);

            todoBusinessImpl.deleteTodosNotRelatedToSpring("Ranga");

      Mockito.verify(todoService).deleteTodo(stringArgumentCaptor.capture
());

            assertEquals("Learn to Dance",
stringArgumentCaptor.getValue());
      }
}
====================Spy============================
Comparing Mock With Spy
Mock : It does not retain actual functioanlity
Spy : will retain functionality of class and override certain
functionality which we want by stubbing
Definition of Spy :Spy allows you to keep track of real objects and also
allows to override specific behaviour

Code Example 1: In Below Example mockDemo() --> adding element will not
affect arrayListMock
                  but spyDemo() --> adding element will actually add
element to spy list
      @Test
      public void mockDemo() {
            List arrayListMock=mock(ArrayList.class);
            assertEquals(0, arrayListMock.size());
            arrayListMock.add("dummy");
            assertEquals(0, arrayListMock.size());
      }

      @Test
      public void spyDemo() {
            List arrayListSpy=spy(ArrayList.class);
            assertEquals(0, arrayListSpy.size());
            arrayListSpy.add("dummy");
```

```
            assertEquals(0, arrayListSpy.size()); --> **** This Test will
Fail ****
            assertEquals(1, arrayListSpy.size());
      }
```

Code Example 2:Mocked list will not get affected by adding element and will override the functioanlity as per mocked method
Spy List will actually get affected by adding element but when u override a particular method then it will behave as it is expected to do so

```
      @Test
      public void mockDemo() {
            List arrayListMock=mock(ArrayList.class);
            when(arrayListMock.size()).thenReturn(5);
            arrayListMock.add("dummy");
            assertEquals(5, arrayListMock.size());
      }

      @Test
      public void spyDemo() {
            List arrayListSpy=spy(ArrayList.class);
            when(arrayListSpy.size()).thenReturn(5);
            arrayListSpy.add("dummy");
            assertEquals(5, arrayListSpy.size());---> here size will not
be 6 bcoz size() functionality is overriden
      }
```

Note : We should avoid using Spy ..why ?
1.spy is used for legacy systems "where u do not have control over dependency" and we want to track whats actually happening with dependency
2.also in spy we are using actual and overriding some functioanlity thatincrease the complexity

Please Go through Link to understand Limitationd of Mockito :https://github.com/in28minutes/MockitoTutorialForBeginners/blob/master/Step14.md

We cannot mock Static Methods,private methods and constructors ..that can be achieved using PowerMock

PowerMock For Static Methods:
Dependencies :

```
      powermock-api-mockito
      powermock-module-junit4

public class UtilityClass {
      static int staticMethod(long value) {
            // Some complex logic is done here...
            throw new RuntimeException(
                        "I dont want to be executed. I will anyway be
mocked out.");
      }
}

interface Dependency {
      List<Integer> retrieveAllStats();
}

public class SystemUnderTest {
```

```
	private Dependency dependency;

	public int methodUsingAnArrayListConstructor() {
		ArrayList list = new ArrayList();
		return list.size();
	}

	public int methodCallingAStaticMethod() {
		//privateMethodUnderTest calls static method
SomeClass.staticMethod
		List<Integer> stats = dependency.retrieveAllStats();
		long sum = 0;
		for (int stat : stats)
			sum += stat;
		return UtilityClass.staticMethod(sum);
	}

	private long privateMethodUnderTest() {
		List<Integer> stats = dependency.retrieveAllStats();
		long sum = 0;
		for (int stat : stats)
			sum += stat;
		return sum;
	}
}

@RunWith(PowerMockRunner.class)
@PrepareForTest({ UtilityClass.class})
public class PowerMockitoMockingStaticMethodTest {

	@Mock
	Dependency dependencyMock;

	@InjectMocks
	SystemUnderTest systemUnderTest;

	@Test
	public void powerMockito_MockingAStaticMethodCall() {


	when(dependencyMock.retrieveAllStats()).thenReturn(Arrays.asList(1,
2, 3));

		PowerMockito.mockStatic(UtilityClass.class);

		when(UtilityClass.staticMethod(anyLong())).thenReturn(150);

		assertEquals(150,
systemUnderTest.methodCallingAStaticMethod());

		//To verify a specific method call
		//First : Call PowerMockito.verifyStatic()
		//Second : Call the method to be verified
		PowerMockito.verifyStatic();
		UtilityClass.staticMethod(1 + 2 + 3);

		// verify exact number of calls
		//PowerMockito.verifyStatic(Mockito.times(1));
```

```
	}
}

PowerMock For Private Methods :
@RunWith(PowerMockRunner.class)
public class PowerMockitoTestingPrivateMethodTest {

	@Mock
	Dependency dependencyMock;

	@InjectMocks
	SystemUnderTest systemUnderTest;

	@Test
	public void powerMockito_CallingAPrivateMethod() throws Exception {
		when(dependencyMock.retrieveAllStats()).thenReturn(
				Arrays.asList(1, 2, 3));
		long value = (Long) Whitebox.invokeMethod(systemUnderTest,
				"privateMethodUnderTest");
		assertEquals(6, value);
	}
}

Power Mock With Constructor :
@RunWith(PowerMockRunner.class)
@PrepareForTest({ SystemUnderTest.class /*To be able to mock the
Constructor, we need to add in the Class that creates the new object*/})
public class PowerMockitoMockingConstructorTest {

	private static final int SOME_DUMMY_SIZE = 100;

	@Mock
	Dependency dependencyMock;

	@InjectMocks
	SystemUnderTest systemUnderTest;

	@Test
	public void powerMockito_MockingAConstructor() throws Exception {

		ArrayList<String> mockList = mock(ArrayList.class);

		stub(mockList.size()).toReturn(SOME_DUMMY_SIZE);


	PowerMockito.whenNew(ArrayList.class).withAnyArguments().thenReturn
(
				mockList);

		int size =
systemUnderTest.methodUsingAnArrayListConstructor();

		assertEquals(SOME_DUMMY_SIZE, size);
	}
}
```