==========

Unit Testing with Junit and Mocking

==========

1) What is Unit Testing & Why ?

2) What is Junit & How to use Junit ?

3) What is Mocking & How to use Mock Objects for UT ?

4) What is Code Coverage & How to improve that ?

==========

Unit Testing ?

==========

=> It is the process of testing individual components of software application.

=> Unit Testing is used to identify the bugs available in our code.

=> With the help of Unit testing we can provide quality code for higher environments.

=> Developers are responsible to perform Unit Testing.

=> To perform Unit Testing we will use Junit and Mocking.

==========

What is Junit ?

==========

-> Junit is a java based framework which is used to implement unit testing for Java applications.

-> The current version of Junit is Junit 5

Junit 5 = Junit Platform + Junit Jupiter + Junit vintage

Platform => Provides Runtime to run junit tests on Java

Jupiter => Provided Annotation to implement unit test cases

Vintage => Provides Backward compatability (Junit 3 & Junit 4 tests support)

-> Junit framework provided several annotations to perform Unit Testing like below

@Test

@ParameterizedTest

@ValueSource

@BeforeAll

@AfterAll

@BeforeEach

@AfterEach

-> Junit framework provided several assertXXX methods to verify Unit Results.

assertEquals(expected, actual)

assertNotEquals(expected, actual)

assertNull( )

assertNotNull ( )

assertTrue( )

assertFalse ( ) etc...

################ Unit Testing we have to implement with Isolation
##############################

==========

What is Mocking ?

==========

=> The process of creating Substiute object for the real object is called as Mocking.

Mock Object = Dummy Object

=> Mock Objects are used for Unit Testing.

=> By using Mock Objects we can achieve Isolated Unit Testing.

=> Isolated unit testing means testing only our target method functionality.

=> There are serveral frameworks available to implement Mocking...

Ex: Easy Mock, Wire Mock, JMockito, Power Mock etc.....

==========

What is Code Coverage ?

==========

=> Code Coverage is the process of identifying how many lines of code is tested as part of unit testing.

1) Which lines covered in unit testing

2) Which lines not-covered in unit testing

=> Industry standard is 80% code coverage.

=> We have several tools to generate Code Coverage Report

Ex: Jacocco, Cobertura etc...

==========

```
public class Calculator {

	public int add(int i, int j) {
		return i + j;
	}

	public int mul(int i, int j) {
		return i * j;
	}
}
```

```
================================================================================
=============
public class CalculatorTest {

	private Calculator c = new Calculator();

	@Test
	public void testAdd() {
		int actualResult = c.add(1, 2);
		int expectedResult = 3;
		assertEquals(expectedResult, actualResult);
	}

	@Test
	public void testMul() {
		int actual = c.mul(2, 2);
		int expected = 5;
		assertEquals(expected, actual);
	}
}
================================================================================
===========
public class PalindromeCheck {

	public boolean isPalidrome(String str) {
		String reverse = "";
		int length = str.length();

		for (int i = length - 1; i >= 0; i--) {
```

```
                reverse = reverse + str.charAt(i);
            }

            if (str.equals(reverse)) {
                return true;
            } else {
                return false;
            }
        }
}
```

==========

```
public class PalindromeTest {

    @ParameterizedTest
    @ValueSource(strings = {"liril", "madam", "racecar", "ashok"})
    public void testIsPalindrome(String str) {

        PalindromeCheck pc = new PalindromeCheck();
        boolean actual = pc.isPalidrome(str);

        assertTrue(actual);
    }
}
```

==========

```
public class StringUtils {
```

```
	public Integer stringToInt(String str) {

		if (str == null || str.trim().length() == 0) {
			throw new IllegalArgumentException("Input is null or empty");
		}

		return Integer.parseInt(str);
	}
}
```

==========

```
public class StringUtilsTest {

	private StringUtils su = new StringUtils();

	@Test
	public void testStringToInt1() {
		Integer actual = su.stringToInt("123");
		assertEquals(123, actual);
	}

	@Test
	public void testStringToInt2() {
		assertThrows(IllegalArgumentException.class,  () -> su.stringToInt(null));
	}

	@Test
```

```
	public void testStringToInt3() {
		assertThrows(IllegalArgumentException.class,  () -> su.stringToInt(""));
	}

}
```

==========

```
@RestController
public class WelcomeRestController {

	@Autowired
	private WelcomeService welcomeService;

	@GetMapping("/welcome")
	public ResponseEntity<String> welcome() {
		String responseMsg = welcomeService.getWelcomeMsg();
		return new ResponseEntity<>(responseMsg, HttpStatus.OK);
	}

	@GetMapping("/greet")
	public ResponseEntity<String> greet() {
		String responseMsg = welcomeService.getGreetMsg();
		return new ResponseEntity<>(responseMsg, HttpStatus.OK);
	}
}
```

==========

```
@WebMvcTest(value = WelcomeRestController.class)
public class WelcomeRestControllerTest {

	@MockBean
	private WelcomeService welcomeService;

	@Autowired
	private MockMvc mockMvc;

	@Test
	public void testGreet() throws Exception {
		when(welcomeService.getGreetMsg()).thenReturn("Good Luck..!!");

		MockHttpServletRequestBuilder reqBuilder = MockMvcRequestBuilders.get("/greet");

		MvcResult andReturn = mockMvc.perform(reqBuilder).andReturn();

		MockHttpServletResponse response = andReturn.getResponse();

		int status = response.getStatus();

		assertEquals(200, status);
	}
```

```
	@Test
	public void testWelcome() throws Exception {

		when(welcomeService.getWelcomeMsg()).thenReturn("Good Evening");

		MockHttpServletRequestBuilder requestBuilder =
MockMvcRequestBuilders.get("/welcome");

		MvcResult result = mockMvc.perform(requestBuilder).andReturn();

		MockHttpServletResponse response = result.getResponse();

		int status = response.getStatus();

		assertEquals(200, status);

	}
}
```

==========

==========