JUnit

Course Objective

To understand and execute test using Junit

- Junit API
- Junit Annotations
- Assert classes

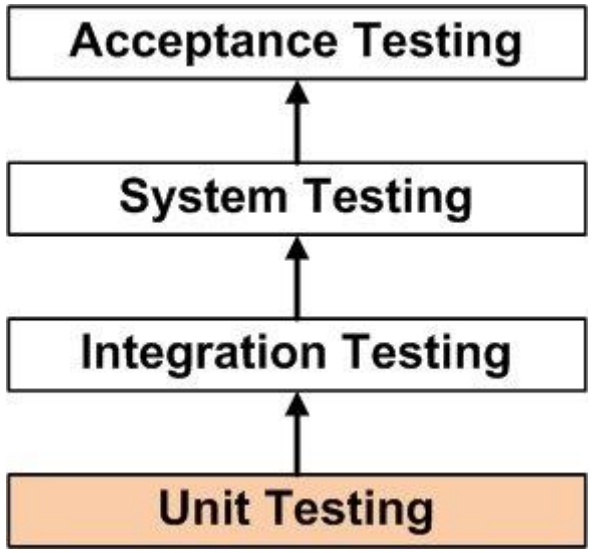
Unit Testing

What is Unit Testing?

- **Unit Testing** is a level of the software testing process where individual units/components of a software/system are tested.
- The purpose is to validate that each unit of the software performs as designed.
- A unit of work can span a single method, a whole class or multiple classes working together to achieve one single logical purpose that can be verified.
- Unit Testing is normally performed by software developers

Unit Testing Hierarchy

- The percentage of code which is tested by unit tests is typically called *test coverage*.
- Layers in testing



Good Unit Test

Characteristics of good unit test

- Able to be fully automated
- Good Coverage
- Can be run in any order if part of many other tests
- Readable
- Maintainable
- Consistently returns the same result
- Runs in memory

Unit Testing Frameworks

- There are several testing frameworks available for Java.
  - Jtest
  - JUnit
  - JWalk
  - Cactus
  - TestNG
  - EasyMock
  - JMock
  - Mockito
- The most popular ones are Junit and TestNG.

Junit - Introduction

What is Junit?

- JUnit is a simple open source Java testing framework used to write and run repeatable automated tests.
- It’s written by Erich Gamma and Kent Beck.
- Runs a bunch of tests and reports their results.

Features of JUnit

- Assertions for testing expected results.
- Test fixtures for sharing common test data.
- Test suites for easily organizing and running tests.
- Provides Annotation to identify the test methods

Environment Setup

JUnit is a framework for Java, so the very first requirement is to have JDK installed in your machine.

JDK	1.5 or above.
Junit Jar	junit4.11.jar
hamcrest-core jar	hamcrest-core-1.3.jar

Basic Sample

- Sample Program

```
public class MessageUtil {
     private String message;
     public MessageUtil(String message){
     this.message = message;
     }
     public String printMessage(){
     System.out.println(message); return message;
  }
```

- Test Case Class

```
public class TestJunit {
    String message = "Hello World";
    MessageUtil messageUtil = new MessageUtil(message);
    @Test
    public void testPrintMessage() {
        assertEquals(message,messageUtil.printMessage());
     }
  }
```

Coding Convention for Junit class

- Name of the test class must end with “Test”
- Name of the method must begin with “test”
- Return type of a test method must be void
- Test method must not throw any exception
- Test method must not have any parameter

JUnit Api

The most important package in JUnit is **junit.framework** which contain all the core classes

Class Name	Functionality
Assert	A set of assert methods.
TestCase	A test case defines the fixture to run multiple tests.
TestResult	A TestResult collects the results of executing a test case.
TestSuite	A TestSuite is a Composite of Tests.
Test Runner	A Test runner is an executable program that runs tests

JUnit - Annotations Cont…

Every time we run a JUnit test class, a new instance of it is created. JUnit framework provides basic lifecycle annotations

@BeforeClass:

- This method is called only once, whereas, other instance lifecycle methods are called every time before calling each test method.
- This annotation is useful for initializing static resources which would, otherwise, be expensive to create during each test invocation

@AfterClass:

- Similar to @BeforeClass but is called at the very end of all test/other lifecycle methods.
- It is called only once. Useful for static resource clean up.

JUnit - Annotations Cont…

@Before:

- It is invoked every time before each test method invocation.
- Used to setup instance variables/resources which can be used during a test method execution.

@After:

- Similar to @Before but runs after target test method execution.
- Useful for cleaning up instance resources.

JUnit - Annotations Cont…

@Test:

- Perform one or more assertions by using static methods of org.junit.Assert.
- Assert methods throw org.junit.AssertionError on assertion failure.
- This exception or any other exception is reported as test failure. If no exceptions are thrown then the test will pass.

Annotations - Sample Demo

```
public class ExecutionProcedureJunit {
    //execute only once, in the starting
    @BeforeClass
    public static void beforeClass() {
    System.out.println("in before class");
    }
    //execute only once, in the end
    @AfterClass
    public static void afterClass()
    { System.out.println("in after class");
    }
    //execute for each test, before executing test
    @Before
    public void before() {
    System.out.println("in before");
    }
    //execute for each test, after executing test
    @After
    public void after() {
    System.out.println("in after");
    }
    //test case 1
    @Test
    public void testCase1() {
    System.out.println("in test case 1");
    }
    //test case 2
    @Test
    public void testCase2() {
    System.out.println("in test case 2");
    }
}
```

Assert Methods

S.No.	Method	Description
1.	void assertEquals(boolean expected, boolean actual)	It checks whether two values are equals similar to equals method of Object class
2.	void assertFalse(boolean condition)	functionality is to check that a condition is false.
3.	void assertNotNull(Object object)	"assertNotNull" functionality is to check that an object is not null.
4.	void assertNull(Object object)	"assertNull" functionality is to check that an object is null.
5.	void assertTrue(boolean condition)	"assertTrue" functionality is to check that a condition is true.
6.	void fail()	If you want to throw any assertion error, you have fail() that always results in a fail verdict.
7.	void assertSame([String message]	"assertSame" functionality is to check that the two objects refer to the same object.
8.	void assertNotSame([String message]	"assertNotSame" functionality is to check that the two objects do not refer to the same object.

Junit – Sample code

Cont…

Steps to create Junit Test case

STEP1 :Create a Maven project

STEP2:Create a Java class

```
public class Addition {
	public int add(int a,int b){
		return (a+b);
	}
}
```

Junit – Sample code Cont…

STEP 3:Test location

- src/main/java - for Java classes
- src/test/java - for test classes

Junit – Sample code

Cont…

STEP 4: Create a Test class with Test method

```
public class AdditionTest {
        private Addition addition;

        /** * Initialization */
        @Before
        public void setUp() {
                addition = new Addition();
        }

        /** * Test case for add method */
        @Test
        public void test() {
                int i = addition.add(3, 7);
                assert Equals(10, i);
                }

                /** * destroy the object */
                @After
                public void tearDown() {
                        addition = null;
                }
}
```

Junit – Sample code Cont…

STEP 5:Add the following dependencies in POM.XML

```
<dependencies>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.12</version>
                <scope>test</scope>
                <exclusions>
                        <exclusion>
                                <groupId>org.hamcrest</groupId>
                                <artifactId>hamcrest-core</artifactId>
                        </exclusion>
                </exclusions>
        </dependency>
```

Junit – Sample code Cont…

```
<!-- This will get hamcrest-core automatically -->
        <dependency>
                <groupId>org.hamcrest</groupId>
                <artifactId>hamcrest-library</artifactId>
                <version>1.3</version>
                <scope>test</scope>
        </dependency>
</dependencies>
```

Junit – Sample code

STEP 6:Execute the junit test

Right click(project) -> Run AS -> Junit Test

Summary

- A ***unit test*** targets a small unit of code, e.g., a method or a class.
- An ***integration test*** aims to test the behavior of a component or the integration between a set of components.
- ***Performance tests*** are used to benchmark software components repeatedly. Their purpose is to ensure that the code under test runs fast enough even if it's under high load
- **test automation** is the use of special software to control the execution of tests and the comparison of actual outcomes with predicted outcomes

Thank you

Innovative Services

Passionate Employees

Delighted Customers