Unit Testing using – Junit
Garbage Collection

Presented by

Sagar
Java Consultant

Software Testing

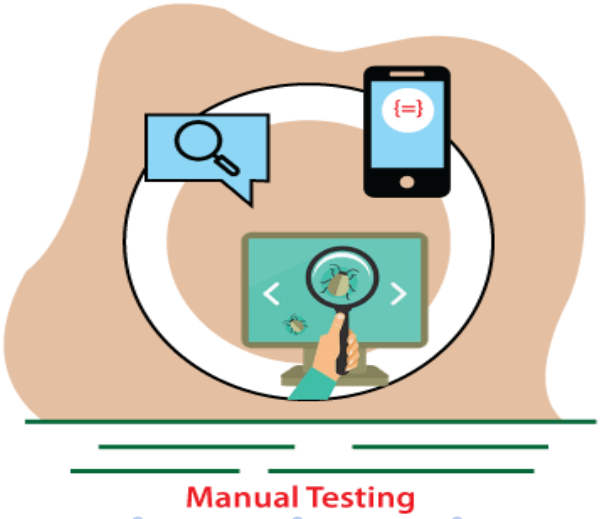
Software testing is a process of analyzing an application's functionality as per the customer prerequisite.

Software testing ensures that our software is bug-free or stable

Types of testing

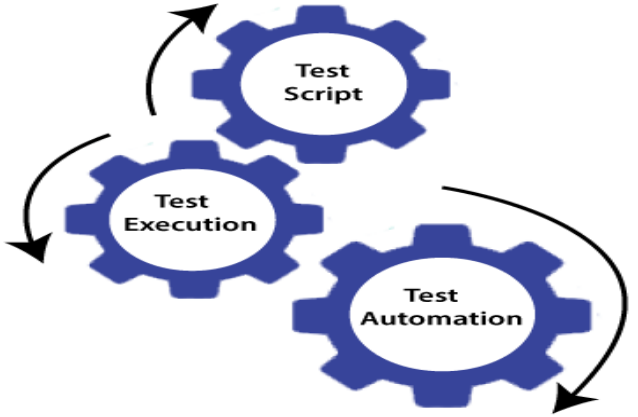
Manual Testing:

Testing without using any automation tool is known as **manual testing**.



Automation Testing:

Testing by using automation tools is known as **automation testing**



JUnit

Unit Testing : Unit Testing is used to verify a small chunk of code by creating a function or a method.

The term "unit" refers to a Java class or a method.

Importance of Unit Testing :

- Unit Testing is used to identify defects early in software development cycle.
- Defects in the design of code affect the development system.

Why Unit Testing? :

- It finds bugs early in the code, which can lead to bug-free code.
- Unit testing is useful for developers to detect bugs early.
- To develop more reliable and bug-free code.

Assetion, Fixure

Assetion :

Unit test assertion is just a Boolean expression.

It contains a true and false binary.

The expression is placed into the testing program and pertains to a certain section of the software being tested.

Ex:

```
import static org.junit.jupiter.api.Assert.*;
public class MainTest {
@Test
public void twoPlusTwoEqualsFalse() {
int result = 2 + 2;
assertEquals(4, result);
}
}
```

```
@Test
void threePlusFiveEqualsEight() {
Calculator calculator = new Calculator();// syntax:
assertEquals(expected value, actual value, message);
assertEquals(8, calculator.add(3, 5));
}
```

Assertion Methods

An assertXxxx() methods are useful in determining Pass or Fail status of a test case.

The assert methods are provided by the class org.junit.Assert

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNotNull;

import org.junit.Test;
public class TestJunit {
    @Test
    public void testAdd() {
        //test data
        int num= 5;
        String str= "Abc";

        //check for equality
        assertEquals("Abc", str);

        //check for false condition
        assertFalse(num > 6);

        //check for not null value
        assertNotNull(str);
    }
}
```

Test Fixtures

Test Fixtures
JUnit test fixtures are a set of objects, data, or code used to prepare a testing environment and provide a known starting point for testing.

That includes the preparation and cleanup tasks necessary for testing a particular unit of code.

Ex:

```
// sets up junit environment
protected void setUp() throws Exception {
        super.setUp();
        calculator = new MyCalculator();
}

// releases junit environment
protected void tearDown() throws Exception {
        super.tearDown();
}
```

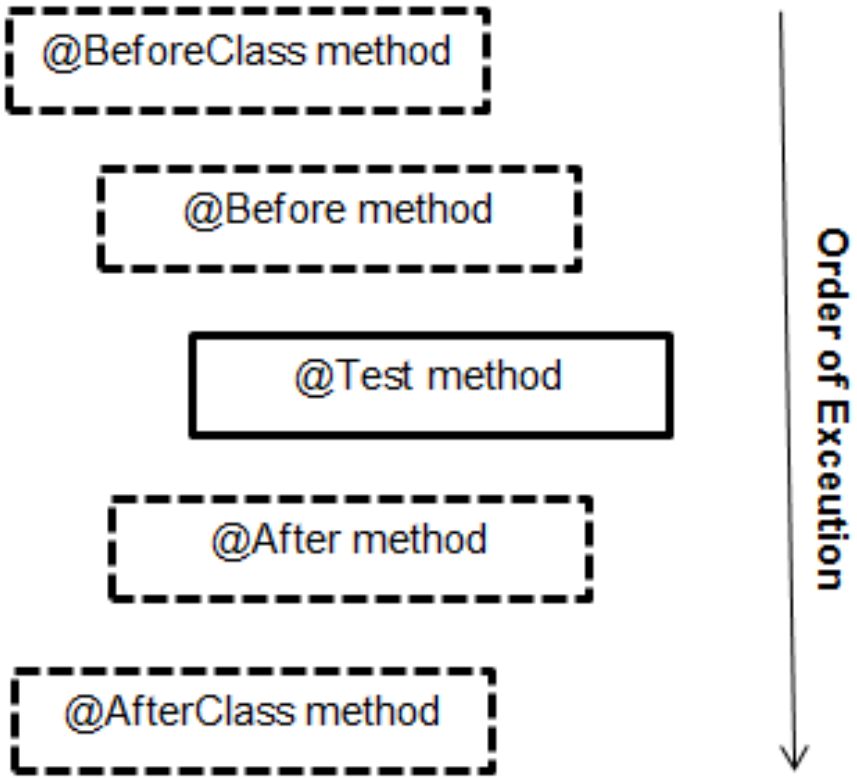
Test Fixtures …

- There are four fixture annotations: two for class-level fixtures and two for method-level fixtures.

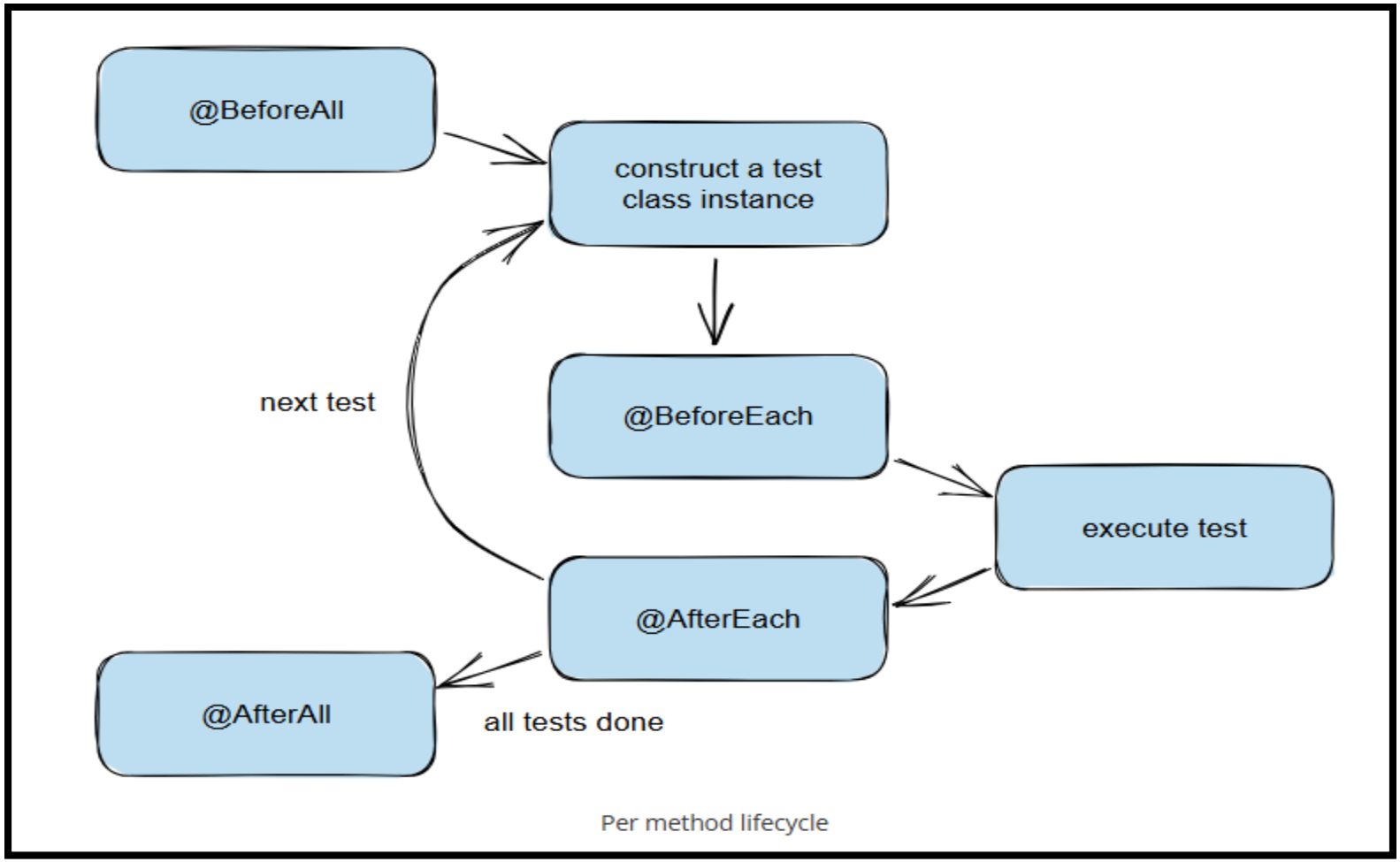
- Junit - 4
- At the class level, there are *@BeforeClass* and *@AfterClass*, and at the method (test) level,

- At the method level, there are *@Before* and *@After*.
- ** Junit 5 - *@BeforeEach, @AfterEach, @BeforeAll, @AfterAll*

Test Fixtures … Junit 4.0



Test Fixtures . . . Junit 5.0



Unit Test Case for Exception

Using **expected =**

```
@Test(expected = ArithmeticException.class)
public void functionTest() {
        obj.getQuotient();
}
```

Using **assertThrows()** :

```
@Test
public void testDivideByZero() {
        Calculator calculator = new Calculator();
        assertThrows(IllegalArgumentException.class, () ->
        calculator.divide(10, 0));
}
```

Unit Test Time Out

A timeout configured test should fail if its execution time exceeds a given duration

If a test case takes more time than the specified number of milliseconds, then JUnit will automatically mark it as failed.

Using **timeout=**

```
@Test(timeout = 1000)
public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
        messenger.printMessage();
}
```

Using **assertTimeout() :**

```
@Test
public void test7Seconds() {
        Assertions.assertTimeout( Duration.ofSeconds(7), () ->
                                                delaySeconds(6));
System.out.println("Test Passed within the time");
}
```

Unit Test TestSuite

- **A test case** is the smallest possible testing unit in developing automated tests.
- **A test suite** is a collection of related test cases that can be managed and run as a single unit
- Eg.

```
@RunWith(Suite.class)

@Suite.SuiteClasses(
        {TestMessengerOne.class,TestMessengerTwo.class}
)
public class JunitTestSuite {
}
```

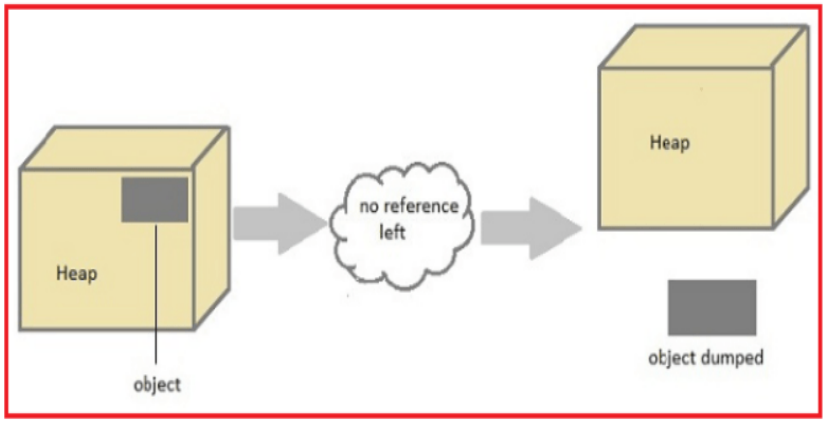
Garbage Collection

Garbage Collection :

Garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.

An in-use object, or a referenced object mean the object is being used.

An unused object, or a referenced object mean the object is not in use.



Unit Test TestSuite

- Eg.

```
Employee e1 = new Employee();
Employee e2 = new Employee();
e1=e2;
//now the first object referred by e1 is available for garbage collection
```

Types of JVM GCs:

The Serial GC :

With the serial collector, garbage collections are done serially (using a single virtual CPU)

The Parallel GC :

The parallel garbage collector uses multiple threads to perform the young generation garbage collection by using N CPUs

Types of garbage collectors

The Concurrent Mark Sweep (CMS) Collector :

The Concurrent Mark Sweep (CMS) collector (also referred to as the concurrent low pause collector) collects the tenured generation.

The G1 Garbage Collector :

The Garbage First or G1 garbage collector is available in Java 7 and is designed to be the long-term replacement for the CMS collector.

The finalize() method

finalize() Method :

Just before destroying an object, Garbage Collector calls the finalize() method on the object to perform cleanup activities.

Eg.

```
public void finalize(){
   // code goes here

}
```

```
public void finalize() {
        System.out.println ("Garbage Collection performed by JVM");
}
```


return
THANK
YOU
control