JUnit (Java Unit Testing) interview questions

BY CHAITANYA SINGH | FILED UNDER: JAVA Q&A

This is Part 2 of Q&A. Read first part here – JUnit (Java Unit Testing) interview questions and answers – Part1.

Question1: What is Junit?

Answer: Java + unit testing = Junit

1. Junit is open source testing framework developed for unit testing java code and is now the default framework for testing Java development.
2. It has been developed by Erich Gamma and Kent Beck.
3. It is an application programming interface for developing test cases in java which is part of the XUnit Family.
4. It helps the developer to write and execute repeatable automated tests.
5. **Eclipse IDE** comes with both Junit and it's plug-in for working with Junit.
6. Junit also has been ported to various other languages like PHP, Python, C++ etc.

Question2: Who should use Junit, developers or testers?

Answer: Used by developers to implement unit tests in Java. Junit is designed for unit testing, which is really a coding process, not a testing process. But many testers or QA engineers are also required to use Junit for unit testing.

Question3: Why do you use Junit to test your code?

Answer: Junit provides a framework to achieve all the following-:

1. Test early and often automated testing.
2. Junit tests can be integrated with the build so regression testing can be done at unit level.
3. Test Code reusage.
4. Also when there is a transfer Junit tests act as a document for the unit tests.

Question4: How do you install Junit?

Answer: Let's see the installation of Junit on windows platform:

1. Download the latest version of Junit
from http://download.sourceforge.net/junit/

2. Uncompress the zip to a directory of your choice.

3. Add JUnit to the classpath:

```
set CLASSPATH=%CLASSPATH%;%JUNIT_HOME%junit.jar
```

4. Test the installation by running the sample tests that come along with Junit located in the installation directory. Therefore, make sure that the JUnit installation directory is on your CLASSPATH. Then simply type:

java org.junit.runner.JUnitCore org.junit.tests.AllTests

All the tests should pass with an “OK” message. If the tests don’t pass, verify that junit.jar is in the CLASSPATH.

Question5: What are unit tests?

Answer: A unit test is nothing more than the code wrapper around the application code that can be executed to view pass – fail results.

Question6: When are Unit Tests written in Development Cycle?

Answer: Tests are written before the code during development in order to help coders write the best code. Test-driven development is the ideal way to create a bug free code. When all tests pass, you know you are done! Whenever a test fails or a bug is reported, we must first write the necessary unit test(s) to expose the bug(s), and then fix them. This makes it almost impossible for that particular bug to resurface later.

Question7: How to write a simple Junit test class?

Answer: To write a test case, follow these steps:

1. Define a subclass of TestCase.
2. Override the setUp() method to initialize object(s) under test.
3. Optionally override the tearDown() method to release object(s) under test.

Define one or more public testXYZ() methods that exercise the object(s) under test and assert expected results.

Question8: What Is Junit TestCase?

Answer: JUnit TestCase is the base class, junit.framework.TestCase, that allows you to create a test case. (Although, TestCase class is no longer supported in JUnit 4.4.)

A test case defines the fixture to run multiple tests. To define a test case

-Implement a subclass of TestCase

-Define instance variables that store the state of the fixture

-Initialize the fixture state by overriding setUp

-Clean-up after a test by overriding tearDown

Each test runs in its own fixture so there can be no side effects among test runs.

Question9: What Is Junit TestSuite?

Answer: JUnit TestSuite is a container class under package junit.framework.TestSuite. It allows us to group multiple test cases into a collection and run them together. (JUnit 4.4 does not support TestSuite class now).

Question10: What is Junit Test Fixture?

Answer: A test fixture is a fixed state of a set of objects used as a baseline for running tests. Their purpose is to ensure that there is a well known and fixed environment in which tests are run so that results are repeatable. Examples of fixtures:

- Loading a database with a specific, known set of data
- Copying a specific known set of files
- Preparation of input data and setup/creation of fake or mock objects

If a group of tests shares the same fixtures, you should write a separate setup code to create the common test fixture. If a group of tests requires different test fixtures, you can write code inside the test method to create its own test fixture.

Question11: What happens if a Junit test method is declared as "private"?

Answer: If a Junit test method is declared as "private", the compilation will pass ok. But the execution will fail. This is because Junit requires that all test methods must be declared as "public".

Question12: What happens If a Junit test method Is declared to return "String"?

Answer: If a Junit test method is declared to return "String", the compilation will pass ok. But the execution will fail. This is because Junit requires that all test methods must be declared to return "void".

Question13: Why not just use system.out.println () for Unit Testing?

Answer: Debugging the code using system.out.println() will lead to manual scanning of the whole output every time the program is run to ensure the code is doing the expected operations. Moreover, in the long run, it takes lesser time to code Junit methods and test them on our files.

Question14: The methods Get () and Set () should be tested for which conditions?

Answer: Unit tests performed on java code should be designed to target areas that might break. Since the set() and get() methods on simple data types are unlikely to break, there is no need to test them explicitly. On the other hand, set() and get() methods on complex data types are vulnerable to break. So they should be tested.

Question15: For which conditions, the methods Get () and Set () can be left out for testing?

Answer: You should do this test to check if a property has already been set (in the constructor) at the point you wish to call getX(). In this case you must test the constructor, and not the getX() method. This kind of test is especially useful if you have multiple constructors.

Question16: Do you need to write a test class for every class that needs to be tested?

Answer: No. We need not write an independent test class for every class that needs to be tested. If there is a small group of tests sharing a common test

fixture, you may move those tests to a new test class. If you have two groups of tests that you think you'd like to execute separately from one another, it is wise to place them in separate test classes.

Question17: How do you test a "protected" method?

Answer: A protected method can only be accessed within the same package where the class is defined. So, testing a protected method of a target class means we need to define your test class in the same package as the target class.

Question18: How do you test a "private" method?

Answer: A private method only be accessed within the same class. So there is no way to test a "private" method of a target class from any test class. A way out is that you can perform unit testing manually or can change your method from "private" to "protected".

Question19: Do you need to write a main () method compulsorily in a Junit test case class?

Answer: No. But still developers write the main() method in a JUnit test case class to call a JUnit test runner to run all tests defined in this class like:

```
public static void main(String[] args) {
   junit.textui.TestRunner.run(Calculator.class);
}
```

Since you can call a JUnit runner to run a test case class as a system command, explicit main() for a Junit test case is not recommended. junit.textui.TestRunner.run() method takes the test class name as its argument. This method automatically finds all class methods whose name starts with test. Thus it will result in below mentioned findings:

1. testCreateLogFile()
2. testExists()
3. testGetChildList()

It will execute each of the 3 methods in unpredictable sequence (hence test case methods should be independent of each other) and give the result in console.

Question20: What happens if a test method throws an exception?

Answer: If you write a test method that throws an exception by itself or by the method being tested, the JUnit runner will declare that test as fail.

The example test below is designed to let the test fail by throwing the uncaught IndexOutOfBoundsException exception:

```
import org.junit.*;
import java.util.*;
public class UnexpectedExceptionTest2 {
// throw any unexpected exception
@Test public void testGet() throws Exception {
      ArrayList emptyList = new ArrayList();
      Exception anyException = null; // don't catch any exception
      Object o = emptyList.get(1); }
}
```

If you run this test, it will fail:

OUTPUT:
There was 1 failure:

testGet(UnexpectedExceptionTest2)

java.lang.IndexOutOfBoundsException: Index: 1, Size: 0

at java.util.ArrayList.RangeCheck(ArrayList.java:547)

at java.util.ArrayList.get(ArrayList.java:322)

at UnexpectedExceptionTest2.testGet(UnexpectedExceptionTest2.ja

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAcce

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMe

at java.lang.reflect.Method.invoke(Method.java:597)

at org.junit.internal.runners.TestMethod.invoke(TestMethod.java

at org.junit.internal.runners.MethodRoadie.runTestMethod(Method

at org.junit.internal.runners.MethodRoadie$2.run(MethodRoadie.j

at org.junit.internal.runners.MethodRoadie.runBeforesThenTestTh

at org.junit.internal.runners.MethodRoadie.runTest(MethodRoadie

at org.junit.internal.runners.MethodRoadie.run(MethodRoadie.jav

at org.junit.internal.runners.JUnit4ClassRunner.invokeTestMetho

at org.junit.internal.runners.JUnit4ClassRunner.runMethods(JUni

at org.junit.internal.runners.JUnit4ClassRunner$1.run(JUnit4Cla

at org.junit.internal.runners.ClassRoadie.runUnprotected(ClassR

at org.junit.internal.runners.ClassRoadie.runProtected(ClassRoa

at org.junit.internal.runners.JUnit4ClassRunner.run(JUnit4Class

at org.junit.internal.runners.CompositeRunner.runChildren(Compo

at org.junit.internal.runners.CompositeRunner.run(CompositeRunn

at org.junit.runner.JUnitCore.run(JUnitCore.java:130)

at org.junit.runner.JUnitCore.run(JUnitCore.java:109)

at org.junit.runner.JUnitCore.run(JUnitCore.java:100)

at org.junit.runner.JUnitCore.runMain(JUnitCore.java:81)

at org.junit.runner.JUnitCore.main(JUnitCore.java:44)

FAILURES!!!

Tests run: 1, Failures: 1

Question21: When objects are garbage collected after a test is executed?

Answer: By design, the tree of Test instances is built in one pass. Then the tests are executed in a second pass. The test runner holds strong references to all Test instances for the duration of the test execution. This means that for a very long test run with many Test instances, none of the tests may be garbage collected until the end of the entire test run. Therefore, if you allocate external or limited resources in a test, you are responsible for freeing those resources. Explicitly setting an object to null in the tearDown() method, for example, allows it to be garbage collected before the end of the entire test run.

Question22: What is Java "assert" statement?

Answer: Java assertions allow the developer to put "assert" statements in Java source code to help unit testing and debugging.

An "assert" statement has the following format:

```
assert boolean_expression : string_expression;
```

When this statement is executed:

- If boolean_expression evaluates to true, the statement will pass normally.
- If boolean_expression evaluates to false, the statement will fail with an "AssertionError" exception.

Helper methods which help to determine if the methods being tested are performing correctly or not

- **assertEquals([String message], expected, actual)** –any kind of object can be used for testing equality –native types and objects, also specify tolerance when checking for equality in case of floating point numbers.
- **assertNull([String message], java.lang.Objectobject)** –asserts that a given object is null
- **assertNotNull([String message], java.lang.Objectobject)** –asserts that a given object isnotnull
- **assertTrue([String message], Boolean condition)** –Asserts that the given Boolean condition is true
- **assertFalse([String message], Boolean condition)** –Asserts that the given Boolean condition is false
- **fail([String message])** –Fails the test immediately with the optional message

Example-:

```
public void testSum() {
```

```
  int num1 = 15;
  int num2 = 50;
  int total = 35;
  int sum = 0;
  sum = num1 + num2;
  assertEquals(sum, total);
}
```