Intro

Hello, everyone, and welcome!

In this survey, you will encounter six tests for each block of questions. Please read each test care
readability and clarity. Although you might notice some similarities between the tests, please fo
best communicate the test's purpose.

Thank you for your time and have a great survey experience!

Demographic questions

How many years of experience do you have in software development?

- ◯ <1 year
- ◯ 1-3 years
- ◯ 3-5 years
- ◯ >5 years

How do you rate your software testing skills?

- ◯ beginner
- ◯ intermediate
- ◯ expert

commons-cli

```
// Test 1
@Test
public void testLong() {
    final Options options = new Options();

    options.addOption("a", "--a", false, "toggle -a");
    options.addOption("b", "--b", true, "set -b");
```

```
    assertTrue(options.hasOption("a"));
    assertTrue(options.hasOption("b"));
}


// Test 2
@Test(timeout = 4000)
public void testModifyOptions() throws Throwable {
    Options modifiedOptions = new Options();
    Options modifiedOptions1 = modifiedOptions.addOption("1G", "org.apache.c
    boolean modifiedBoolean = modifiedOptions1.hasOption("1G");
    assertTrue(modifiedBoolean);
}


// Test 3
@Test(timeout = 4000)
public void testAddingAndCheckingOption() throws Throwable {
    Options testOptions = new Options();
    Options updatedOptions = testOptions.addOption("1G", "org.apache.commons
    boolean isOptionPresent = updatedOptions.hasOption("1G");
    assertTrue(isOptionPresent);
}

// Test 4
@Test(timeout = 4000)
public void testHasOption_withExistingShortOption_shouldReturnTrue()  throws
    Options options = new Options();
    Options optionsWithAddedOption = options.addOption("1G", "org.apache.com
    boolean hasOption = optionsWithAddedOption.hasOption("1G")
    assertTrue(hasOption);
}


// Test 5
@Test(timeout = 4000)
public void improvedTestReadability() throws Throwable {
    Options optionsInstance = new Options();
    Options optionsWithAddedOption = optionsInstance.addOption("1G", "org.ap
    boolean hasTheOptionBeenAdded = optionsWithAddedOption.hasOption("1G");
    assertTrue(hasTheOptionBeenAdded);
}
```

```java
// Test 6
@Test(timeout = 4000)
public void testHasOptionWithLongOption() throws Throwable {
    Options optionsManager = new Options();
    Options optionsWithAddedOption = optionsManager.addOption("1G", "org.apa
    boolean hasOption = optionsWithAddedOption.hasOption("1G");
    assertTrue(hasOption);
}
```

Please evaluate the tests presented in the code by assigning a score from -2 to 2 based on their readability and understandability.

- -2 indicates that the code is very unreadable.
- 0 indicates that the code is normally readable.
- +2 indicates that the code is very readable.

	-2	-1	0	1	2
Test 1	○	○	○	○	○
Test 2	○	○	○	○	○
Test 3	○	○	○	○	○
Test 4	○	○	○	○	○
Test 5	○	○	○	○	○
Test 6	○	○	○	○	○

Please, justify and/or comment on your scores

commons-csv

```java
// Test 1
@Test(timeout = 4000)
public void testRecordConsistencyAndPosition() throws Throwable {
    String[] emptyStringArray = new String[0];
    CSVRecord csvRecordUnderTest = new CSVRecord((CSVParser) null, emptyStri
    boolean isRecordConsistent = csvRecordUnderTest.isConsistent();
    assertEquals((-1L), csvRecordUnderTest.getRecordNumber());
    assertTrue(isRecordConsistent);
    assertEquals(0L, csvRecordUnderTest.getCharacterPosition());
}


// Test 2
@Test(timeout = 4000)
public void recordIsConsistentAndHasExpectedPosition() throws Throwable {
    String[] valuesInRecord = new String[0];
    CSVRecord record = new CSVRecord(null, valuesInRecord, "\u2029", (-1L),
    boolean recordIsConsistent = record.isConsistent();
    assertEquals((-1L), record.getRecordNumber());
    assertTrue(recordIsConsistent);
    assertEquals(0L, record.getCharacterPosition());
}


// Test 3
@Test
public void testIsConsistent() {
    assertTrue(record.isConsistent());
    assertTrue(recordWithHeader.isConsistent());
    final Map map = recordWithHeader.getParser().getHeaderMap();
    map.put("fourth", Integer.valueOf(4));
    assertTrue(recordWithHeader.isConsistent());
}


// Test 4
@Test(timeout = 4000)
public void testDataConsistencyAndPosition() {
    String[] csvRecordValues = new String[0];
    CSVRecord csvRecordInstance = new CSVRecord(null, csvRecordValues, "\u20
    boolean isDataConsistent = csvRecordInstance.isConsistent();
```

```
    assertEquals((-1L), csvRecordInstance.getRecordNumber());
    assertTrue(isDataConsistent);
    assertEquals(0L, csvRecordInstance.getCharacterPosition());
}


// Test 5
@Test(timeout = 4000)
public void testConsistencyAndPosition() throws Throwable {
    String[] values = new String[0];
    CSVRecord record = new CSVRecord(null, values, "\u2029", (-1L), 0L);
    boolean consistent = record.isConsistent();
    assertEquals((-1L), record.getRecordNumber());
    assertTrue(consistent);
    assertEquals(0L, record.getCharacterPosition());
}


// Test 6
@Test(timeout = 4000)
public void testRecordConsistencyAndCharacterPosition() throws Throwable {
    String[] recordValues = new String[0];
    CSVRecord csvRecord = new CSVRecord(null, recordValues, "\u2029", (-1L),
    boolean isRecordDataConsistent = csvRecord.isConsistent();
    assertEquals((-1L), csvRecord.getRecordNumber());
    assertTrue(isRecordDataConsistent);
    assertEquals(0L, csvRecord.getCharacterPosition());
}
```

Please evaluate the tests presented in the code by assigning a score from -2 to 2 based on their readability and understandability.

- -2 indicates that the code is very unreadable.
- 0 indicates that the code is normally readable.
- +2 indicates that the code is very readable.

Test 3	○	○	○	○	○
Test 4	○	○	○	○	○
Test 5	○	○	○	○	○
Test 6	○	○	○	○	○

Please, justify and/or comment on your scores

commons-lang

```
//Test 1
@Test(timeout = 4000)
public void testSortingShortArray_WhenArrayHasOneElement_ShouldReturnSameArr
    short[] inputShortArray = new short[1];
    short[] sortedShortArray = ArraySorter.sort(inputShortArray);
    assertSame(sortedShortArray, inputShortArray);
}

//Test 2
@Test(timeout = 4000)
public void testSortShortArray() throws Throwable {
    short[] newArray = new short[1];
    short[] sortedArray = ArraySorter.sort(newArray);
    assertSame(sortedArray, newArray);
}


//Test 3
@Test(timeout = 4000)
public void shouldSortShortArrayInPlace() throws Throwable {
    short[] inputArray = new short[1];
```

```
    short[] sortedArray = ArraySorter.sort(inputArray);
    assertSame(sortedArray, inputArray);
}


//Test 4
@Test(timeout = 4000)
public void testSortShortArrayWithSingleElement() throws Throwable {
    short[] singleElementShortArray = new short[1];
    short[] sortedArray = ArraySorter.sort(singleElementShortArray);
    assertSame(sortedArray, singleElementShortArray);
}


//Test 5
@Test(timeout = 4000)
public void shouldSortShortArrayWithoutChangingOriginalReference() throws Th
    short[] unsortedShortArray = new short[1];
    short[] sortedShortArray = ArraySorter.sort(unsortedShortArray);
    assertSame(sortedShortArray, unsortedShortArray);
}


//Test 6
@Test
public void testSortShortArray() {
    final short[] array1 = {2, 1};
    final short[] array2 = array1.clone();
    Arrays.sort(array1);
    assertArrayEquals(array1, ArraySorter.sort(array2));
    assertNull(ArraySorter.sort((short[]) null));
}
```

Please evaluate the tests presented in the code by assigning a score from -2 to 2 based on their readability and understandability.

- -2 indicates that the code is very unreadable.
- 0 indicates that the code is normally readable.
- +2 indicates that the code is very readable.

Please, justify and/or comment on your scores

gson

```
// Test 1
@Test(timeout = 4000)
public void testClosedWriter_ThrowsException() throws Throwable {
  JsonTreeWriter jsonWriter = new JsonTreeWriter();
  jsonWriter.close();
  Boolean sampleBoolean = new Boolean(false);
  // Undeclared exception!
  try {
    jsonWriter.value(sampleBoolean);
    fail("Expecting exception: IllegalStateException");

  } catch (IllegalStateException e) {
    //
    // no message in exception (getMessage() returned null)
    //
    verifyException("com.google.gson.internal.bind.JsonTreeWriter", e);
  }
```

```
}


// Test 2
@Test(timeout = 4000)
public void testWritingBooleanValueAfterClosingShouldThrowIllegalStateExcept
    JsonTreeWriter jsonTreeWriter = new JsonTreeWriter();
    jsonTreeWriter.close();
    Boolean booleanValue = Boolean.FALSE;
    try {
        jsonTreeWriter.value(booleanValue);
        fail("Expected IllegalStateException to be thrown");
    } catch (IllegalStateException e) {
        verifyException("com.google.gson.internal.bind.JsonTreeWriter", e);
    }
}


// Test 3
public void testPrematureClose() throws Exception {
    JsonTreeWriter writer = new JsonTreeWriter();
    writer.setLenient(true);
    writer.beginArray();
    try {
      writer.close();
      fail();
    } catch (IOException expected) {}
}

// Test 4
@Test(timeout = 4000)
public void testWriterThrowsExceptionWhenCalledAfterClose() throws Throwable
    JsonTreeWriter jsonTreeWriter = new JsonTreeWriter();
    jsonTreeWriter.close();

    Boolean booleanValue = new Boolean(false);
    try {
        jsonTreeWriter.value(booleanValue);
        fail("Expecting exception: IllegalStateException");
    } catch (IllegalStateException e) {
        verifyException("com.google.gson.internal.bind.JsonTreeWriter", e);
    }
}
```

```
// Test 5
@Test(timeout = 4000)
public void testJsonTreeWriter() throws Throwable {
    JsonTreeWriter writer = new JsonTreeWriter();
    writer.close();
    Boolean boolValue = new Boolean(false);
    // Undeclared exception!
    try {
        writer.value(boolValue);
        fail("Expecting exception: IllegalStateException");
    } catch (IllegalStateException e) {
        verifyException("com.google.gson.internal.bind.JsonTreeWriter", e);
    }
}
```

```
// Test 6
@Test(timeout = 4000)
public void shouldThrowExceptionWhenValueIsAddedAfterClose() throws Throwabl
    JsonTreeWriter jsonWriterUnderTest = new JsonTreeWriter();
    jsonWriterUnderTest.close();
    Boolean testBooleanValue = new Boolean(false);
    // Undeclared exception!
    try {
        jsonWriterUnderTest.value(testBooleanValue);
        fail("Expecting exception: IllegalStateException");
    } catch(IllegalStateException e) {
        //
        // no message in exception (getMessage() returned null)
        //
        verifyException("com.google.gson.internal.bind.JsonTreeWriter", e);
    }
}
```

Please evaluate the tests presented in the code by assigning a score from -2 to 2 based on their readability and understandability.

- -2 indicates that the code is very unreadable.

- 0 indicates that the code is normally readable.
- +2 indicates that the code is very readable.

	-2	-1	0	1	2
Test 1	○	○	○	○	○
Test 2	○	○	○	○	○
Test 3	○	○	○	○	○
Test 4	○	○	○	○	○
Test 5	○	○	○	○	○
Test 6	○	○	○	○	○

Please, justify and/or comment on your scores

jfreechart

```
// Test 1
@Test(timeout = 4000)
public void testOutlierGettersWithNegativeValues() throws Throwable {
    Outlier outlierWithNegativeValues = new Outlier(-1063.769739825398, 465.
    double outlierRadius = outlierWithNegativeValues.getRadius();
    assertEquals(0.0, outlierWithNegativeValues.getX(), 0.01);
    assertEquals(-1063.769739825398, outlierRadius, 0.01);
    assertEquals(1529.427504405398, outlierWithNegativeValues.getY(), 0.01);
}
```

```
// Test 2
@Test(timeout = 4000)
public void testOutlierCreationAndAccessors() {
    Outlier outlierUnderTest = new Outlier((-1063.769739825398), 465.6577645
```

```
    double outlierRadius = outlierUnderTest.getRadius();
    assertEquals(0.0, outlierUnderTest.getX(), 0.01);
    assertEquals((-1063.769739825398), outlierRadius, 0.01);
    assertEquals(1529.427504405398, outlierUnderTest.getY(), 0.01);
}


// Test 3
@Test
public void testConstructor() {
    Outlier out = new Outlier(1.0, 2.0, 3.0);
    assertEquals(-2.0, out.getX(), EPSILON);
    assertEquals(-1.0, out.getY(), EPSILON);
    assertEquals(3.0, out.getRadius(), EPSILON);
}


// Test 4
@Test(timeout = 4000)
public void testOutlierCreationAndProperties()  throws Throwable  {
    Outlier testOutlier = new Outlier((-1063.769739825398), 465.65776458, (-
    double radiusValue = testOutlier.getRadius();
    assertEquals(0.0, testOutlier.getX(), 0.01);
    assertEquals((-1063.769739825398), radiusValue, 0.01);
    assertEquals(1529.427504405398, testOutlier.getY(), 0.01);
}


// Test 5
@Test(timeout = 4000)
public void testGetRadiusAndCoordinates() throws Throwable {
  Outlier outlier = new Outlier((-1063.769739825398), 465.65776458, (-1063.7
  double actualRadius = outlier.getRadius();
  assertEquals(0.0, outlier.getX(), 0.01);
  assertEquals((-1063.769739825398), actualRadius, 0.01);
  assertEquals(1529.427504405398, outlier.getY(), 0.01);
}


// Test 6
@Test(timeout = 4000)
public void testOutlierRadiusAndCoordinates() throws Throwable {
    Outlier testOutlier = new Outlier((-1063.769739825398), 465.65776458, (-
```

```
    double testRadius = testOutlier.getRadius();
    assertEquals(0.0, testOutlier.getX(), 0.01);
    assertEquals((-1063.769739825398), testRadius, 0.01);
    assertEquals(1529.427504405398, testOutlier.getY(), 0.01);
}
```

Please evaluate the tests presented in the code by assigning a score from -2 to 2 based on their readability and understandability.

- -2 indicates that the code is very unreadable.
- 0 indicates that the code is normally readable.
- +2 indicates that the code is very readable.

	-2	-1	0	1	2
Test 1	○	○	○	○	○
Test 2	○	○	○	○	○
Test 3	○	○	○	○	○
Test 4	○	○	○	○	○
Test 5	○	○	○	○	○
Test 6	○	○	○	○	○

Please, justify and/or comment on your scores

Powered by Qualtrics