Intro

Hello, everyone, and welcome!

In this survey, you will encounter six tests for each block of questions. Please read each test care
and clarity.
Although you might notice some similarities between the tests, please focus on the identifier na
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(timeout = 4000)
public void testImproved() throws Throwable {
    Options newOptions = PatternOptionBuilder.parsePattern("");
    assertNotNull(newOptions);
}

// Test 2
```

```
@Test
public void testEmptyPattern() {
    final Options options = PatternOptionBuilder.parsePattern("");
    assertTrue(options.getOptions().isEmpty());
}

// Test 3
@Test(timeout = 4000)
public void testParseEmptyPatternReturnsValidOptions() throws Throwable {
    Options parsedOptions = PatternOptionBuilder.parsePattern("");
    assertNotNull(parsedOptions);
}

// Test 4
@Test(timeout = 4000)
public void testEmptyPatternParsing() throws Throwable {
    Options parsedOptions = PatternOptionBuilder.parsePattern("");
    assertNotNull(parsedOptions);
}

// Test 5
@Test(timeout = 4000)
public void testParseEmptyPattern() {
    Options emptyPatternOptions = PatternOptionBuilder.parsePattern("");
    assertNotNull(emptyPatternOptions);
}

// Test 6
@Test(timeout = 4000)
public void shouldParseEmptyPatternSuccessfully() throws Throwable {
    Options parsedOptions = PatternOptionBuilder.parsePattern("");
    assertNotNull(parsedOptions);
}
```

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

```
// Test 1
@Test
public void testReadingInDifferentBuffer() throws Exception {
    final char[] tmp1 = new char[2], tmp2 = new char[4];
    try (ExtendedBufferedReader reader = createBufferedReader("1\r\n2\r\n"))
        reader.read(tmp1, 0, 2);
        reader.read(tmp2, 2, 2);
        assertEquals(2, reader.getCurrentLineNumber());
    }
}
```

```
// Test 2
@Test(timeout = 4000)
public void testReadAndGetCurrentLineNumberOfExtendedBufferedReader() throws
    StringReader inputReaderWithContent = new StringReader("~x");
    ExtendedBufferedReader extendedBufferedReaderWithContent = new ExtendedB
```

```java
        int readResultFromExtendedBufferedReader = extendedBufferedReaderWithCon
        assertEquals(126, readResultFromExtendedBufferedReader);

        long currentLineNumberOfExtendedBufferedReader = extendedBufferedReaderW
        assertEquals(1L, currentLineNumberOfExtendedBufferedReader);

        StringReader inputReaderWithoutContent = new StringReader("");
        ExtendedBufferedReader extendedBufferedReaderWithoutContent = new Extend

        char[] bufferForReading = new char[5];
        int readEmptyResultFromExtendedBufferedReader = extendedBufferedReaderWi
        assertEquals((-1), readEmptyResultFromExtendedBufferedReader);
    }


    // Test 3
    @Test(timeout = 4000)
    public void testReadAndGetLineNumber() throws Throwable {
        StringReader stringReaderWithContent = new StringReader("~x");
        ExtendedBufferedReader extendedBufferedReaderWithContent = new ExtendedB
        int charRead = extendedBufferedReaderWithContent.read();
        assertEquals(126, charRead);

        long actualLineNumber = extendedBufferedReaderWithContent.getCurrentLine
        assertEquals(1L, actualLineNumber);

        StringReader stringReaderWithoutContent = new StringReader("");
        ExtendedBufferedReader extendedBufferedReaderWithoutContent = new Extend

        char[] readingBuffer = new char[5];
        int emptyStreamReadResult = extendedBufferedReaderWithoutContent.read(re
        assertEquals((-1), emptyStreamReadResult);
    }


    // Test 4
    @Test(timeout = 4000)
    public void testReadAndGetCurrentLineNumber() throws Throwable {
        StringReader inputReader1 = new StringReader("~x");
        ExtendedBufferedReader bufferedReader1 = new ExtendedBufferedReader(inpu
        int readResult1 = bufferedReader1.read();
        assertEquals(126, readResult1);
        long lineNumber1 = bufferedReader1.getCurrentLineNumber();
        assertEquals(1L, lineNumber1);

        StringReader inputReader2 = new StringReader("");
        ExtendedBufferedReader bufferedReader2 = new ExtendedBufferedReader(inpu

        char[] buffer = new char[5];
        int readResult2 = bufferedReader2.read(buffer);
        assertEquals((-1), readResult2);
    }


    // Test 5
    @Test(timeout = 4000)
    public void testReadingAndCurrentLineNumberTracking() throws Throwable {
        StringReader readerWithContent = new StringReader("~x");
        ExtendedBufferedReader bufferedReaderWithContent = new ExtendedBufferedR
        int readResult = bufferedReaderWithContent.read();
        assertEquals(126, readResult);

        long currentLineNumber = bufferedReaderWithContent.getCurrentLineNumber(
        assertEquals(1L, currentLineNumber);

        StringReader emptyReader = new StringReader("");
        ExtendedBufferedReader bufferedReaderWithoutContent = new ExtendedBuffer

        char[] buffer = new char[5];
        int readEmptyResult = bufferedReaderWithoutContent.read(buffer);
        assertEquals((-1), readEmptyResult);
    }


    // Test 6
    @Test(timeout = 4000)
    public void improveReadAndGetCurrentLineNumberTest() throws Throwable {
        StringReader contentReader = new StringReader("~x");
        ExtendedBufferedReader contentBufferedReader = new ExtendedBufferedReade
        int readValue = contentBufferedReader.read();
        assertEquals(126, readValue);

        long lineNumber = contentBufferedReader.getCurrentLineNumber();
        assertEquals(1L, lineNumber);
```

```
      StringReader emptyReader = new StringReader("");
      ExtendedBufferedReader emptyBufferedReader = new ExtendedBufferedReader(

      char[] buffer = new char[5];
      int emptyReadValue = emptyBufferedReader.read(buffer);
      assertEquals((-1), emptyReadValue);
  }
```

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-lang

```
// Test 1
@Test(timeout = 4000)
public void improveTestNamingAndIdentifiers() throws Throwable {
    BasicThreadFactory.Builder builder = new BasicThreadFactory.Builder();
    MockThread mockThread = new MockThread("4Cave");
    BasicThreadFactory.Builder updatedBuilder = builder.namingPattern("4Cave
    BasicThreadFactory factory = updatedBuilder.build();
    factory.newThread(mockThread);
    long threadCount = factory.getThreadCount();
    assertEquals(1L, threadCount);
}


// Test 2
@Test(timeout = 4000)
  public void shouldCreateAndCountThread()  throws Throwable  {
      BasicThreadFactory.Builder threadFactoryBuilder = new BasicThreadFacto
      MockThread testThread = new MockThread("4Cave");
      BasicThreadFactory.Builder threadFactoryBuilderWithPattern = threadFac
      BasicThreadFactory basicThreadFactory = threadFactoryBuilderWithPatter
      basicThreadFactory.newThread(testThread);
      long actualThreadCount = basicThreadFactory.getThreadCount();
      assertEquals(1L, actualThreadCount);
}


// Test 3
@Test(timeout = 4000)
public void testNewThreadCreationWithNamingPattern() throws Throwable {
    BasicThreadFactory.Builder threadFactoryBuilder = new BasicThreadFactory
    MockThread mockRunnable = new MockThread("SampleRunnable");
    BasicThreadFactory.Builder threadFactoryBuilderWithNamingPattern = threa
    BasicThreadFactory threadFactory = threadFactoryBuilderWithNamingPattern
    Thread newThread = threadFactory.newThread(mockRunnable);
    long threadCount = threadFactory.getThreadCount();
    assertEquals(1L, threadCount);
}


// Test 4
@Test
public void testNewThreadExHandler() {
    final ThreadFactory wrapped = EasyMock.createMock(ThreadFactory.class);
```

```
        final Runnable r = EasyMock.createMock(Runnable.class);
        final Thread.UncaughtExceptionHandler handler = EasyMock
                .createMock(Thread.UncaughtExceptionHandler.class);
        final Thread t = new Thread();
        EasyMock.expect(wrapped.newThread(r)).andReturn(t);
        EasyMock.replay(wrapped, r, handler);
        final BasicThreadFactory factory = builder.wrappedFactory(wrapped)
                .uncaughtExceptionHandler(handler).build();
        assertSame(t, factory.newThread(r), "Wrong thread");
        assertEquals(handler, t.getUncaughtExceptionHandler(), "Wrong exception
        EasyMock.verify(wrapped, r, handler);
    }


    // Test 5
    @Test(timeout = 4000)
    public void shouldIncreaseThreadCountWhenNewThreadIsCreated() throws Throwab
        BasicThreadFactory.Builder threadFactoryBuilder = new BasicThreadFactory
        MockThread mockThread = new MockThread("ThreadName");
        BasicThreadFactory.Builder configuredThreadFactoryBuilder = threadFactor
        BasicThreadFactory threadFactory = configuredThreadFactoryBuilder.build(
        threadFactory.newThread(mockThread);
        long threadCount = threadFactory.getThreadCount();
        assertEquals(1L, threadCount);
    }


    // Test 6
    @Test(timeout = 4000)
    public void testThreadFactoryCreation() throws Throwable {
        BasicThreadFactory.Builder threadFactoryBuilder = new BasicThreadFactory
        MockThread mockThread = new MockThread("threadName");
        BasicThreadFactory.Builder threadFactoryBuilderWithNamePattern = threadF
        BasicThreadFactory threadFactory = threadFactoryBuilderWithNamePattern.b
        threadFactory.newThread(mockThread);
        long threadCount = threadFactory.getThreadCount();
        assertEquals(1L, threadCount);
    }
```

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

gson

```java
// Test 1
@Test(timeout = 4000)
public void testLazilyParsedNumberHashCodeGeneration() {
    LazilyParsedNumber numberWithEmptyValue = new LazilyParsedNumber("");
    numberWithEmptyValue.hashCode();
}


// Test 2
public void testHashCode() {
  LazilyParsedNumber n1 = new LazilyParsedNumber("1");
```

```
    LazilyParsedNumber n1Another = new LazilyParsedNumber("1");
    assertEquals(n1.hashCode(), n1Another.hashCode());
  }


  // Test 3
  @Test(timeout = 4000)
  public void testHashCodeOfEmptyString() throws Throwable {
      LazilyParsedNumber parsedNumberWithEmptyString = new LazilyParsedNumber(
      parsedNumberWithEmptyString.hashCode();
  }


  // Test 4
  @Test(timeout = 4000)
  public void improveReadabilityOfTest28() throws Throwable {
      LazilyParsedNumber parsedNumber = new LazilyParsedNumber("");
      parsedNumber.hashCode();
  }


  // Test 5
  @Test(timeout = 4000)
  public void testEmptyStringHashCode() throws Throwable {
      LazilyParsedNumber numberFromString = new LazilyParsedNumber("");
      numberFromString.hashCode();
  }


  // Test 6
  @Test(timeout = 4000)
  public void shouldCalculateHashCodeFromValue()  throws Throwable  {
      LazilyParsedNumber number = new LazilyParsedNumber("");
      number.hashCode();
  }
```

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 testIntervalCoordinates()  throws Throwable  {
    XYInterval interval = new XYInterval(0.0, 0.0, 0.0, 0.0, (-1.0));
    double yLowValue = interval.getYLow();
    assertEquals(0.0, interval.getXLow(), 0.01);
    assertEquals(0.0, interval.getY(), 0.01);
    assertEquals(0.0, yLowValue, 0.01);
    assertEquals((-1.0), interval.getYHigh(), 0.01);
    assertEquals(0.0, interval.getXHigh(), 0.01);
}


// Test 2
@Test(timeout = 4000)
```

```
public void testGettersWithZeroAndNegativeValues() throws Throwable {
    XYInterval xyInterval = new XYInterval(0.0, 0.0, 0.0, 0.0, -1.0);
    double expectedYLowValue = xyInterval.getYLow();
    assertEquals(0.0, xyInterval.getXLow(), 0.01);
    assertEquals(0.0, xyInterval.getY(), 0.01);
    assertEquals(0.0, expectedYLowValue, 0.01);
    assertEquals(-1.0, xyInterval.getYHigh(), 0.01);
    assertEquals(0.0, xyInterval.getXHigh(), 0.01);
}


// Test 3
@Test(timeout = 4000)
public void testIntervalValues() throws Throwable {
    XYInterval interval = new XYInterval(0.0, 0.0, 0.0, 0.0, (-1.0));
    double lowY = interval.getYLow();
    assertEquals(0.0, interval.getXLow(), 0.01);
    assertEquals(0.0, interval.getY(), 0.01);
    assertEquals(0.0, lowY, 0.01);
    assertEquals((-1.0), interval.getYHigh(), 0.01);
    assertEquals(0.0, interval.getXHigh(), 0.01);
}


// Test 4
@Test(timeout = 4000)
public void testGettersReturnValue() throws Throwable {
    XYInterval interval = new XYInterval(0.0, 0.0, 0.0, 0.0, (-1.0));
    double actualYLow = interval.getYLow();
    assertEquals(0.0, interval.getXLow(), 0.01);
    assertEquals(0.0, interval.getY(), 0.01);
    assertEquals(0.0, actualYLow, 0.01);
    assertEquals((-1.0), interval.getYHigh(), 0.01);
    assertEquals(0.0, interval.getXHigh(), 0.01);
}



// Test 5
@Test
public void testCloning() {
    XYInterval i1 = new XYInterval(1.0, 2.0, 3.0, 2.5, 3.5);
    assertFalse(i1 instanceof Cloneable);
}


// Test 6
@Test(timeout = 4000)
public void testVerifyXYIntervalConstruction() throws Throwable {
    XYInterval xyInterval = new XYInterval(0.0, 0.0, 0.0, 0.0, (-1.0));
    double yLow = xyInterval.getYLow();
    assertEquals(0.0, xyInterval.getXLow(), 0.01);
    assertEquals(0.0, xyInterval.getY(), 0.01);
    assertEquals(0.0, yLow, 0.01);
    assertEquals((-1.0), xyInterval.getYHigh(), 0.01);
    assertEquals(0.0, xyInterval.getXHigh(), 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