See the Assessment Guide for information on how to interpret this report.

ASSESSMENT SUMMARY

```
Compilation:  PASSED
API:          PASSED

SpotBugs:     PASSED
PMD:          FAILED (1 warning)
Checkstyle:   PASSED

Correctness:  41/41 tests passed
Memory:       1/1 tests passed
Timing:       41/41 tests passed

Aggregate score: 100.00%
[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 60%, Timing: 10%, Memory: 20% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
3.1K Feb 10 23:52 BruteCollinearPoints.java
3.2K Feb 10 23:52 FastCollinearPoints.java
4.0K Feb 10 23:52 Point.java


********************************************************************************
*  COMPILING
********************************************************************************


% javac Point.java
*-----------------------------------------------------------

% javac LineSegment.java
*-----------------------------------------------------------

% javac BruteCollinearPoints.java
*-----------------------------------------------------------

% javac FastCollinearPoints.java
*-----------------------------------------------------------


================================================================


Checking the APIs of your programs.
*-----------------------------------------------------------
Point:

BruteCollinearPoints:

FastCollinearPoints:

================================================================


********************************************************************************
*  CHECKING STYLE AND COMMON BUG PATTERNS
********************************************************************************


% spotbugs *.class
*-----------------------------------------------------------


================================================================


% pmd .
*-----------------------------------------------------------
BruteCollinearPoints.java:13: The private instance (or static) variable 'numOfSegments' can be made 'final'; it is initialized only in the declaration or c
PMD ends with 1 warning.


================================================================


% checkstyle *.java
*-----------------------------------------------------------

% custom checkstyle checks for Point.java
*-----------------------------------------------------------

% custom checkstyle checks for BruteCollinearPoints.java
```

```
*-----------------------------------------------------------

% custom checkstyle checks for FastCollinearPoints.java
*-----------------------------------------------------------


================================================================


*********************************************************************************
*  TESTING CORRECTNESS
*********************************************************************************

Testing correctness of Point
*-----------------------------------------------------------
Running 3 total tests.

Test 1: p.slopeTo(q)
  * positive infinite slope, where p and q have coordinates in [0, 500)
  * positive infinite slope, where p and q have coordinates in [0, 32768)
  * negative infinite slope, where p and q have coordinates in [0, 500)
  * negative infinite slope, where p and q have coordinates in [0, 32768)
  * positive zero     slope, where p and q have coordinates in [0, 500)
  * positive zero     slope, where p and q have coordinates in [0, 32768)
  * symmetric for random points p and q with coordinates in [0, 500)
  * symmetric for random points p and q with coordinates in [0, 32768)
  * transitive for random points p, q, and r with coordinates in [0, 500)
  * transitive for random points p, q, and r with coordinates in [0, 32768)
  * slopeTo(), where p and q have coordinates in [0, 500)
  * slopeTo(), where p and q have coordinates in [0, 32768)
  * slopeTo(), where p and q have coordinates in [0, 10)
  * throw a java.lang.NullPointerException if argument is null
==> passed

Test 2: p.compareTo(q)
  * reflexive, where p and q have coordinates in [0, 500)
  * reflexive, where p and q have coordinates in [0, 32768)
  * antisymmetric, where p and q have coordinates in [0, 500)
  * antisymmetric, where p and q have coordinates in [0, 32768)
  * transitive, where p, q, and r have coordinates in [0, 500)
  * transitive, where p, q, and r have coordinates in [0, 32768)
  * sign of compareTo(), where p and q have coordinates in [0, 500)
  * sign of compareTo(), where p and q have coordinates in [0, 32768)
  * sign of compareTo(), where p and q have coordinates in [0, 10)
  * throw java.lang.NullPointerException exception if argument is null
==> passed

Test 3: p.slopeOrder().compare(q, r)
  * reflexive, where p and q have coordinates in [0, 500)
  * reflexive, where p and q have coordinates in [0, 32768)
  * antisymmetric, where p, q, and r have coordinates in [0, 500)
  * antisymmetric, where p, q, and r have coordinates in [0, 32768)
  * transitive, where p, q, r, and s have coordinates in [0, 500)
  * transitive, where p, q, r, and s have coordinates in [0, 32768)
  * sign of compare(), where p, q, and r have coordinates in [0, 500)
  * sign of compare(), where p, q, and r have coordinates in [0, 32768)
  * sign of compare(), where p, q, and r have coordinates in [0, 10)
  * throw java.lang.NullPointerException if either argument is null
==> passed


Total: 3/3 tests passed!


================================================================
*********************************************************************************
*  TESTING CORRECTNESS (substituting reference Point and LineSegment)
*********************************************************************************

Testing correctness of BruteCollinearPoints
*-----------------------------------------------------------
Running 17 total tests.

The inputs satisfy the following conditions:
  - no duplicate points
  - no 5 (or more) points are collinear
  - all x- and y-coordinates between 0 and 32,767

Test 1: points from a file
  * filename = input8.txt
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
==> passed

Test 2a: points from a file with horizontal line segments
  * filename = horizontal5.txt
  * filename = horizontal25.txt
==> passed

Test 2b: random horizontal line segments
  *  1 random horizontal line segment
  *  5 random horizontal line segments
  * 10 random horizontal line segments
  * 15 random horizontal line segments
==> passed

Test 3a: points from a file with vertical line segments
```

```
  * filename = vertical5.txt
  * filename = vertical25.txt
==> passed

Test 3b: random vertical line segments
  *  1 random vertical line segment
  *  5 random vertical line segments
  * 10 random vertical line segments
  * 15 random vertical line segments
==> passed

Test 4a: points from a file with no line segments
  * filename = random23.txt
  * filename = random38.txt
==> passed

Test 4b: random points with no line segments
  *  5 random points
  * 10 random points
  * 20 random points
  * 50 random points
==> passed

Test 5: points from a file with fewer than 4 points
  * filename = input1.txt
  * filename = input2.txt
  * filename = input3.txt
==> passed

Test 6: check for dependence on either compareTo() or compare()
        returning { -1, +1, 0 } instead of { negative integer,
        positive integer, zero }
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
==> passed

Test 7: check for fragile dependence on return value of toString()
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
==> passed

Test 8: random line segments, none vertical or horizontal
  *  1 random line segment
  *  5 random line segments
  * 10 random line segments
  * 15 random line segments
==> passed

Test 9: random line segments
  *  1 random line segment
  *  5 random line segments
  * 10 random line segments
  * 15 random line segments
==> passed

Test 10: check that data type is immutable by testing whether each method
         returns the same value, regardless of any intervening operations
  * input8.txt
  * equidistant.txt
==> passed

Test 11: check that data type does not mutate the constructor argument
  * input8.txt
  * equidistant.txt
==> passed

Test 12: numberOfSegments() is consistent with segments()
  * filename = input8.txt
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
  * filename = horizontal5.txt
  * filename = vertical5.txt
  * filename = random23.txt
==> passed

Test 13: throws an exception if either the constructor argument is null
         or any entry in array is null
  * argument is null
  * Point[] of length 10, number of null entries = 1
  * Point[] of length 10, number of null entries = 10
  * Point[] of length 4, number of null entries = 1
  * Point[] of length 3, number of null entries = 1
  * Point[] of length 2, number of null entries = 1
  * Point[] of length 1, number of null entries = 1
==> passed

Test 14: check that the constructor throws an exception if duplicate points
  * 50 points
  * 25 points
  * 5 points
  * 4 points
  * 3 points
  * 2 points
==> passed
```

```
Total: 17/17 tests passed!


================================================================
Testing correctness of FastCollinearPoints
*-----------------------------------------------------------
Running 21 total tests.

The inputs satisfy the following conditions:
  - no duplicate points
  - all x- and y-coordinates between 0 and 32,767

Test 1: points from a file
  * filename = input8.txt
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
  * filename = input299.txt
==> passed

Test 2a: points from a file with horizontal line segments
  * filename = horizontal5.txt
  * filename = horizontal25.txt
  * filename = horizontal50.txt
  * filename = horizontal75.txt
  * filename = horizontal100.txt
==> passed

Test 2b: random horizontal line segments
  *  1 random horizontal line segment
  *  5 random horizontal line segments
  * 10 random horizontal line segments
  * 15 random horizontal line segments
==> passed

Test 3a: points from a file with vertical line segments
  * filename = vertical5.txt
  * filename = vertical25.txt
  * filename = vertical50.txt
  * filename = vertical75.txt
  * filename = vertical100.txt
==> passed

Test 3b: random vertical line segments
  *  1 random vertical line segment
  *  5 random vertical line segments
  * 10 random vertical line segments
  * 15 random vertical line segments
==> passed

Test 4a: points from a file with no line segments
  * filename = random23.txt
  * filename = random38.txt
  * filename = random91.txt
  * filename = random152.txt
==> passed

Test 4b: random points with no line segments
  *  5 random points
  * 10 random points
  * 20 random points
  * 50 random points
==> passed

Test 5a: points from a file with 5 or more on some line segments
  * filename = input9.txt
  * filename = input10.txt
  * filename = input20.txt
  * filename = input50.txt
  * filename = input80.txt
  * filename = input300.txt
  * filename = inarow.txt
==> passed

Test 5b: points from a file with 5 or more on some line segments
  * filename = kw1260.txt
  * filename = rs1423.txt
==> passed

Test 6: points from a file with fewer than 4 points
  * filename = input1.txt
  * filename = input2.txt
  * filename = input3.txt
==> passed

Test 7: check for dependence on either compareTo() or compare()
        returning { -1, +1, 0 } instead of { negative integer,
        positive integer, zero }
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
  * filename = input299.txt
==> passed

Test 8: check for fragile dependence on return value of toString()
  * filename = equidistant.txt
  * filename = input40.txt
```

```
  * filename = input48.txt
==> passed

Test 9: random line segments, none vertical or horizontal
  *  1 random line segment
  *  5 random line segments
  * 25 random line segments
  * 50 random line segments
  * 100 random line segments
==> passed

Test 10: random line segments
  *  1 random line segment
  *  5 random line segments
  * 25 random line segments
  * 50 random line segments
  * 100 random line segments
==> passed

Test 11: random distinct points in a given range
  * 5 random points in a 10-by-10 grid
  * 10 random points in a 10-by-10 grid
  * 50 random points in a 10-by-10 grid
  * 90 random points in a 10-by-10 grid
  * 200 random points in a 50-by-50 grid
==> passed

Test 12: m*n points on an m-by-n grid
  * 3-by-3 grid
  * 4-by-4 grid
  * 5-by-5 grid
  * 10-by-10 grid
  * 20-by-20 grid
  * 5-by-4 grid
  * 6-by-4 grid
  * 10-by-4 grid
  * 15-by-4 grid
  * 25-by-4 grid
==> passed

Test 13: check that data type is immutable by testing whether each method
         returns the same value, regardless of any intervening operations
  * input8.txt
  * equidistant.txt
==> passed

Test 14: check that data type does not mutate the constructor argument
  * input8.txt
  * equidistant.txt
==> passed

Test 15: numberOfSegments() is consistent with segments()
  * filename = input8.txt
  * filename = equidistant.txt
  * filename = input40.txt
  * filename = input48.txt
  * filename = horizontal5.txt
  * filename = vertical5.txt
  * filename = random23.txt
==> passed

Test 16: throws an exception if either constructor argument is null
         or any entry in array is null
  * argument is null
  * Point[] of length 10, number of null entries = 1
  * Point[] of length 10, number of null entries = 10
  * Point[] of length 4, number of null entries = 1
  * Point[] of length 3, number of null entries = 1
  * Point[] of length 2, number of null entries = 1
  * Point[] of length 1, number of null entries = 1
==> passed

Test 17: check that the constructor throws an exception if duplicate points
  * 50 points
  * 25 points
  * 5 points
  * 4 points
  * 3 points
  * 2 points
==> passed


Total: 21/21 tests passed!


================================================================
********************************************************************************
*  MEMORY
********************************************************************************

Analyzing memory of Point
*-----------------------------------------------------------
Running 1 total tests.

The maximum amount of memory per Point object is 32 bytes.

Student memory = 24 bytes (passed)
```

```
Total: 1/1 tests passed!


================================================================


********************************************************************************
*  TIMING
********************************************************************************

Timing BruteCollinearPoints
*------------------------------------------------------------
Running 10 total tests.

Test 1a-1e: Find collinear points among n random distinct points


                                              slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
-----------------------------------------------------------------------------------------------
=> passed    16   0.00        3640           0           3640                 120
=> passed    32   0.00       71920           0          71920                 496
=> passed    64   0.03     1270752           0        1270752                2016
=> passed   128   0.16    21336000           0       21336000                8128
=> passed   256   2.57   349585280           0      349585280               32640
==> 5/5 tests passed

Test 2a-2e: Find collinear points among n/4 arbitrary line segments


                                              slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
-----------------------------------------------------------------------------------------------
=> passed    16   0.00        3764           0           3764                 152
=> passed    32   0.00       72440           0          72440                 560
=> passed    64   0.06     1272654           0        1272654                2144
=> passed   128   0.16    21344208           0       21344208                8384
=> passed   256   2.48   349618086           0      349618086               33152
==> 5/5 tests passed

Total: 10/10 tests passed!


================================================================


Timing FastCollinearPoints
*------------------------------------------------------------
Running 31 total tests.

Test 1a-1g: Find collinear points among n random distinct points


                                              slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
-----------------------------------------------------------------------------------------------
=> passed    64   0.01        4096       18640          41376                4343
=> passed   128   0.01       16384       88703         193790               16994
=> passed   256   0.06       65536      414779         895094               67006
=> passed   512   0.31      262144     1897735        4057614              265597
=> passed  1024   0.65     1048576     8523532       18095640             1056496
=> passed  2048   1.43     4194304    38125145       80444594             4212225
==> 6/6 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (80444594 / 18095640) = 2.15
=> passed

==> 7/7 tests passed

Test 2a-2g: Find collinear points among the n points on an n-by-1 grid

                                              slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
-----------------------------------------------------------------------------------------------
=> passed    64   0.00        4096        4764          13624                4401
=> passed   128   0.00       16384       17796          51976               17123
=> passed   256   0.01       65536       68717         202970               67260
=> passed   512   0.02      262144      269399         800942              266111
=> passed  1024   0.05     1048576     1065026        3178628             1057566
=> passed  2048   0.07     4194304     4231214       12656732             4214293
=> passed  4096   0.20    16777216    16859163       50495542            16821311
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (50495542 / 12656732) = 2.00
=> passed

==> 8/8 tests passed

Test 3a-3g: Find collinear points among the n points on an n/4-by-4 grid

                                              slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
-----------------------------------------------------------------------------------------------
=> passed    64   0.00        4096       14906          33908                4745
=> passed   128   0.00       16384       43854         104092               18486
=> passed   256   0.01       65536      149618         364772               72745
```

```
=> passed   512   0.03      262144      548156        1358456              287979
=> passed  1024   0.09     1048576     2087496        5223568             1144947
=> passed  2048   0.17     4194304     8122445       20439194             4563827
=> passed  4096   0.55    16777216    31990953       80759122            18219374
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (80759122 / 20439194) = 1.98
=> passed

==> 8/8 tests passed

Test 4a-4g: Find collinear points among the n points on an n/8-by-8 grid

                                                slopeTo()
             n    time     slopeTo()   compare()  + 2*compare()        compareTo()
-----------------------------------------------------------------------------------------------
=> passed    64   0.00        4096       18045          40186                4736
=> passed   128   0.00       16384       75863         168110               18403
=> passed   256   0.00       65536      232229         529994               72407
=> passed   512   0.04      262144      854545        1971234              286616
=> passed  1024   0.13     1048576     3260991        7570558             1139502
=> passed  2048   0.25     4194304    12699218       29592740             4542012
=> passed  4096   0.84    16777216    50043244      116863704            18132014
==> 7/7 tests passed

lg ratio(slopeTo() + 2*compare()) = lg (116863704 / 29592740) = 1.98
=> passed

==> 8/8 tests passed

Total: 31/31 tests passed!


================================================================
```