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

ASSESSMENT SUMMARY

Compilation: PASSED

API: PASSED

Findbugs: FAILED (5 warnings)

Checkstyle: FAILED (9 warnings)

Correctness: 40/41 tests passed

Memory: 1/1 tests passed

Timing: 41/41 tests passed

Aggregate score: 98.54%

[Compilation: 5%, API: 5%, Findbugs: 0%, Checkstyle: 0%, Correctness: 60%, Memory: 10%, Timing: 20%]

ASSESSMENT DETAILS

The following files were submitted:

----------------------------------

3.6K Jun 13 23:37 BruteCollinearPoints.java

4.8K Jun 13 23:37 FastCollinearPoints.java

4.5K Jun 13 23:37 Point.java

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* COMPILING

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% javac Point.java

\*-----------------------------------------------------------

% javac BruteCollinearPoints.java

\*-----------------------------------------------------------

% javac FastCollinearPoints.java

\*-----------------------------------------------------------

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

Checking the APIs of your programs.

\*-----------------------------------------------------------

Point:

BruteCollinearPoints:

FastCollinearPoints:

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* CHECKING STYLE AND COMMON BUG PATTERNS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% findbugs \*.class

\*-----------------------------------------------------------

H D DLS\_DEAD\_LOCAL\_STORE DLS: Assigns a value to the local variable 'p' but that value is never used. At Point.java:[line 153]

L D UC\_USELESS\_VOID\_METHOD UC: The void method 'main()' appears to serve no purpose. At Point.java:[line 154]

M C RCN\_REDUNDANT\_NULLCHECK\_WOULD\_HAVE\_BEEN\_A\_NPE RCN: Checks whether the variable 'points' is null [line 32], even though it can't be null because it was previously dereferenced. At BruteCollinearPoints.java:[line 29]

L D FE\_FLOATING\_POINT\_EQUALITY FE: Tests for exact floating-point equality. Because floating-point calculations may involve rounding, the calculated values may be imprecise. At BruteCollinearPoints.java:[line 45]

M C RCN\_REDUNDANT\_NULLCHECK\_WOULD\_HAVE\_BEEN\_A\_NPE RCN: Checks whether the variable 'points' is null [line 20], even though it can't be null because it was previously dereferenced. At FastCollinearPoints.java:[line 18]

Warnings generated: 5

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

% checkstyle \*.java

\*-----------------------------------------------------------

Point.java:1:3: '//' or '/\*' is not followed by whitespace. [IllegalTokenText]

BruteCollinearPoints.java:1:3: '//' or '/\*' is not followed by whitespace. [IllegalTokenText]

BruteCollinearPoints.java:26:5: Define constructors after static and instance variables but before methods. [DeclarationOrder]

BruteCollinearPoints.java:53:40: 'for' is not followed by whitespace. [WhitespaceAfter]

BruteCollinearPoints.java:77:12: 'for' is not followed by whitespace. [WhitespaceAfter]

FastCollinearPoints.java:1:3: '//' or '/\*' is not followed by whitespace. [IllegalTokenText]

FastCollinearPoints.java:70: Line is longer than 128 characters (currently 144). [LineLength]

FastCollinearPoints.java:71:25: Control variable 'last' is modified inside loop. [ModifiedControlVariable]

FastCollinearPoints.java:74:97: '{' is not followed by whitespace. [WhitespaceAround]

Checkstyle ends with 9 errors.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 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 exception either if argument to constructor is null

or if 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

- data type mutated the points[] array

- data type should have no side effects unless documented in API

\* equidistant.txt

- data type mutated the points[] array

- data type should have no side effects unless documented in API

==> FAILED

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 exception either if argument to constructor is null

or if 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: 20/21 tests passed!

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MEMORY

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Computing 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 0.01 546 0 546 48

=> passed 32 0.00 4930 0 4930 124

=> passed 64 0.00 41602 0 41602 308

=> passed 128 0.01 341250 0 341250 744

=> passed 256 0.03 2763266 0 2763266 1727

==> 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 0.00 633 0 633 44

=> passed 32 0.00 5293 0 5293 121

=> passed 64 0.00 43196 0 43196 311

=> passed 128 0.00 347827 0 347827 741

=> passed 256 0.03 2788759 0 2788759 1733

==> 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 0.02 7564 17721 43006 17902

=> passed 128 0.01 31500 87648 206796 87879

=> passed 256 0.08 128524 408101 944726 410362

=> passed 512 0.26 519180 1881960 4283100 1884844

=> passed 1024 0.44 2086924 8521211 19129346 8528427

=> passed 2048 1.44 8368140 38043671 84455482 38203138

==> 6/6 tests passed

lg ratio(slopeTo() + 2\*compare()) = lg (84455482 / 19129346) = 2.14

=> 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 7564 4566 16696 6763

=> passed 128 0.00 31500 17406 66312 22562

=> passed 256 0.00 128524 67943 264410 79124

=> passed 512 0.01 519180 267857 1054894 291387

=> passed 1024 0.04 2086924 1061948 4210820 1110684

=> passed 2048 0.11 8368140 4225064 16818268 4325226

=> passed 4096 0.36 33513484 16846869 67207222 17052017

==> 7/7 tests passed

lg ratio(slopeTo() + 2\*compare()) = lg (67207222 / 16818268) = 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 7564 14386 36336 16247

=> passed 128 0.00 31500 42920 117340 62659

=> passed 256 0.01 128524 147881 424286 241650

=> passed 512 0.03 519180 544828 1608836 938301

=> passed 1024 0.09 2086924 2081014 6248952 3676692

=> passed 2048 0.32 8368140 8109673 24587486 14488431

=> passed 4096 1.24 33513484 31965619 97444722 57407557

==> 7/7 tests passed

lg ratio(slopeTo() + 2\*compare()) = lg (97444722 / 24587486) = 1.99

=> 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 7564 17251 42066 18176

=> passed 128 0.00 31500 74449 180398 83585

=> passed 256 0.01 128524 229652 587828 336682

=> passed 512 0.04 519180 849527 2218234 1333873

=> passed 1024 0.13 2086924 3251178 8589280 5302543

=> passed 2048 0.48 8368140 12679845 33727830 21139546

=> passed 4096 1.79 33513484 50004769 133523022 84299825

==> 7/7 tests passed

lg ratio(slopeTo() + 2\*compare()) = lg (133523022 / 33727830) = 1.99

=> passed

==> 8/8 tests passed

Total: 31/31 tests passed!

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