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

ASSESSMENT SUMMARY

```
Compilation:  PASSED
API:          PASSED

SpotBugs:     PASSED
PMD:          PASSED
Checkstyle:   FAILED (0 errors, 2 warnings)

Correctness:  32/32 tests passed
Memory:       No tests available for autograding.
Timing:       No tests available for autograding.

Aggregate score: 100.00%
[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 90% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
 556 Aug  3 16:30 BandMatrix.java
 381 Aug  3 16:30 GeneralizedHarmonic.java
 790 Aug  3 16:30 RandomWalker.java
1.1K Aug  3 16:30 RandomWalkers.java


================================================================================
*  COMPILING
================================================================================


% javac GeneralizedHarmonic.java
*------------------------------------------------------------

% javac BandMatrix.java
*------------------------------------------------------------

% javac RandomWalker.java
*------------------------------------------------------------

% javac RandomWalkers.java
*------------------------------------------------------------


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

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

BandMatrix:

RandomWalker:

RandomWalkers:

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


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


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


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


% pmd .
*------------------------------------------------------------


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


% checkstyle *.java
*------------------------------------------------------------
[WARN] RandomWalker.java:4:40: To specify an array type, put the square brackets before the variable name, e.g., 'String[] args' instead of 'String ar
[WARN] RandomWalkers.java:4:40: To specify an array type, put the square brackets before the variable name, e.g., 'String[] args' instead of 'String a
Checkstyle ends with 0 errors and 2 warnings.

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

% custom checkstyle checks for BandMatrix.java
*------------------------------------------------------------
```

```
% custom checkstyle checks for RandomWalker.java
*-----------------------------------------------------------

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


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


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

Testing correctness of GeneralizedHarmonic
*-----------------------------------------------------------
Running 7 total tests.

Test 1: check output format for inputs from assignment specification
  % java GeneralizedHarmonic 1 1
  1.0

  % java GeneralizedHarmonic 2 1
  1.5

  % java GeneralizedHarmonic 3 1
  1.8333333333333333

  % java GeneralizedHarmonic 4 1
  2.083333333333333

  % java GeneralizedHarmonic 1 2
  1.0

  % java GeneralizedHarmonic 2 2
  1.25

  % java GeneralizedHarmonic 3 2
  1.3611111111111112

  % java GeneralizedHarmonic 4 2
  1.4236111111111112

==> passed

Test 2: check correctness for inputs from assignment specification
  * java GeneralizedHarmonic 1 1
  * java GeneralizedHarmonic 1 2
  * java GeneralizedHarmonic 1 3
  * java GeneralizedHarmonic 1 4
  * java GeneralizedHarmonic 2 1
  * java GeneralizedHarmonic 2 2
  * java GeneralizedHarmonic 2 3
  * java GeneralizedHarmonic 3 4
==> passed

Test 3: check correctness when n = r
  * java GeneralizedHarmonic 1 1
  * java GeneralizedHarmonic 2 2
  * java GeneralizedHarmonic 3 3
  * java GeneralizedHarmonic 4 4
  * java GeneralizedHarmonic 5 5
  * java GeneralizedHarmonic 6 6
  * java GeneralizedHarmonic 7 7
  * java GeneralizedHarmonic 8 8
==> passed

Test 4: check when r is fixed and n varies
  * r = 1
  * r = 2
  * r = 3
  * r = 4
  * r = 5
  * r = 6
  * r = 7
==> passed

Test 5: check when n is fixed and r varies
  * n = 1
  * n = 2
  * n = 3
  * n = 4
  * n = 5
  * n = 6
  * n = 7
==> passed

Test 6: check when r is 0
  * r = 0
==> passed

Test 7: check when r is negative
  * r = -1
  * r = -2
  * r = -3
==> passed


GeneralizedHarmonic Total: 7/7 tests passed!


================================================================
Testing correctness of BandMatrix
*-----------------------------------------------------------
```

```
Running 7 total tests.

Test 1: check output format
  % java BandMatrix 8 0
  *  0  0  0  0  0  0  0
  0  *  0  0  0  0  0  0
  0  0  *  0  0  0  0  0
  0  0  0  *  0  0  0  0
  0  0  0  0  *  0  0  0
  0  0  0  0  0  *  0  0
  0  0  0  0  0  0  *  0
  0  0  0  0  0  0  0  *

  % java BandMatrix 8 1
  *  *  0  0  0  0  0  0
  *  *  *  0  0  0  0  0
  0  *  *  *  0  0  0  0
  0  0  *  *  *  0  0  0
  0  0  0  *  *  *  0  0
  0  0  0  0  *  *  *  0
  0  0  0  0  0  *  *  *
  0  0  0  0  0  0  *  *

  % java BandMatrix 8 2
  *  *  *  0  0  0  0  0
  *  *  *  *  0  0  0  0
  *  *  *  *  *  0  0  0
  0  *  *  *  *  *  0  0
  0  0  *  *  *  *  *  0
  0  0  0  *  *  *  *  *
  0  0  0  0  *  *  *  *
  0  0  0  0  0  *  *  *

  % java BandMatrix 8 3
  *  *  *  *  0  0  0  0
  *  *  *  *  *  0  0  0
  *  *  *  *  *  *  0  0
  *  *  *  *  *  *  *  0
  0  *  *  *  *  *  *  *
  0  0  *  *  *  *  *  *
  0  0  0  *  *  *  *  *
  0  0  0  0  *  *  *  *

==> passed

Test 2: check correctness for inputs from assignment specification
  * java BandMatrix 8 0
  * java BandMatrix 8 1
  * java BandMatrix 8 2
  * java BandMatrix 8 3
==> passed

Test 3: check correctness when width = 0
  * java BandMatrix 2 0
  * java BandMatrix 3 0
  * java BandMatrix 4 0
  * java BandMatrix 5 0
  * java BandMatrix 6 0
  * java BandMatrix 7 0
==> passed

Test 4: check correctness when n = width
  * java BandMatrix 2 2
  * java BandMatrix 3 3
  * java BandMatrix 4 4
  * java BandMatrix 5 5
  * java BandMatrix 6 6
  * java BandMatrix 7 7
==> passed

Test 5: check corner cases
  * java BandMatrix 0 0
  * java BandMatrix 1 0
  * java BandMatrix 2 0
  * java BandMatrix 8 9
  * java BandMatrix 8 20
==> passed

Test 6: check correctness when n is fixed and width varies
  * n = 1
  * n = 2
  * n = 3
  * n = 4
  * n = 5
  * n = 6
  * n = 7
==> passed

Test 7: check correctness when width is fixed and n varies
  * width = 0
  * width = 1
  * width = 2
  * width = 3
  * width = 4
  * width = 5
  * width = 6
  * width = 7
==> passed


BandMatrix Total: 7/7 tests passed!


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

```
Testing correctness of RandomWalker
*-----------------------------------------------------------
Running 11 total tests.

Test 1: check output format for inputs from assignment specification
  % java RandomWalker 3
  (0, 0)
  (0, -1)
  (0, -2)
  (0, -1)
  (0, -2)
  (0, -1)
  (1, -1)
  (0, -1)
  (0, 0)
  (0, 1)
  (1, 1)
  (1, 0)
  (0, 0)
  (-1, 0)
  (0, 0)
  (1, 0)
  (0, 0)
  (1, 0)
  (2, 0)
  (3, 0)
  steps = 19

  % java RandomWalker 5
  (0, 0)
  (-1, 0)
  (0, 0)
  (0, -1)
  (-1, -1)
  (-1, 0)
  (-1, 1)
  (-2, 1)
  (-1, 1)
  (-2, 1)
  (-2, 0)
  (-2, 1)
  (-3, 1)
  (-4, 1)
  steps = 13

==> passed

Test 2: check correctness of inputs from assignment specification
  * java RandomWalker 3
  * java RandomWalker 5
==> passed

Test 3: check that random walk stops when distance r from origin
  * java RandomWalker 3
  * java RandomWalker 5
  * java RandomWalker 10
==> passed

Test 4: check that first point in random walk is the origin
  * java RandomWalker 3
  * java RandomWalker 5
  * java RandomWalker 10
==> passed

Test 5: check that successive points in random walk are neighbors
  * java RandomWalker 3
  * java RandomWalker 5
  * java RandomWalker 10
==> passed

Test 6: check that number of steps printed is consistent with number of points printed
  * java RandomWalker 3
  * java RandomWalker 5
  * java RandomWalker 10
==> passed

Test 7: check correctness for corner cases
  * java RandomWalker 0
  * java RandomWalker 1
==> passed

Test 8: check that program produces different walks each time
  * java RandomWalker 6 [ twice ]
  * java RandomWalker 10 [ twice ]
  * java RandomWalker 20 [ twice ]
==> passed

Test 9: check randomness of individual steps in walk
  * java RandomWalker 32
  * java RandomWalker 128
  * java RandomWalker 512
==> passed

Test 10: check randomness of number of steps
  * java RandomWalker 2 [ repeated 1024 times ]
  * java RandomWalker 3 [ repeated 8192 times ]
  * java RandomWalker 4 [ repeated 32768 times ]
  * java RandomWalker 5 [ repeated 131072 times ]
==> passed

Test 11: check what happens when Math.random() always returns the same value
  * Math.random() always returns 0.0
  * Math.random() always returns 0.25
  * Math.random() always returns 0.5
```

```
   * Math.random() always returns 0.75
==> passed


RandomWalker Total: 11/11 tests passed!


*-----------------------------------------------------------
Testing correctness of RandomWalkers
*-----------------------------------------------------------
Running 7 total tests.

Test 1: check output format
  % java RandomWalkers 5 10000
  average number of steps = 15.1604

  % java RandomWalkers 10 1000
  average number of steps = 61.86

  % java RandomWalkers 20 123456
  average number of steps = 236.19736586314153

  % java RandomWalkers 40 1
  average number of steps = 1464.0

  % java RandomWalkers 1 1000
  average number of steps = 1.0

  % java RandomWalkers 1000 1
  average number of steps = 391234.0

  % java RandomWalkers 0 333
  average number of steps = 0.0

==> passed

Test 2: check average number of steps (trials = 10000)
  * java RandomWalkers 1 10000
  * java RandomWalkers 2 10000
  * java RandomWalkers 3 10000
  * java RandomWalkers 4 10000
  * java RandomWalkers 5 10000
  * java RandomWalkers 10 10000
  * java RandomWalkers 20 10000
  * java RandomWalkers 40 10000
==> passed

Test 3: check average number of steps (radius = 5)
  * java RandomWalkers 5 100
  * java RandomWalkers 5 1000
  * java RandomWalkers 5 10000
  * java RandomWalkers 5 100000
  * java RandomWalkers 5 1000000
==> passed

Test 4: check average number of steps (radius = 0)
  * java RandomWalkers 0 1000
  * java RandomWalkers 0 100
  * java RandomWalkers 0 1
==> passed

Test 5: check that the average number of steps is not an integer
  * java RandomWalkers 10 1000
  * java RandomWalkers 7 2500
  * java RandomWalkers 3 10000
==> passed

Test 6: check that program produces different result each time
  * java RandomWalkers 10 10000 [ repeated twice ]
  * java RandomWalkers 20 1000 [ repeated twice ]
  * java RandomWalkers 40 2000 [ repeated twice ]
==> passed

Test 7: check randomness of average number of steps when trials = 1
  * java RandomWalkers 2 1 [ repeated 1024 times ]
  * java RandomWalkers 3 1 [ repeated 8192 times ]
  * java RandomWalkers 4 1 [ repeated 65536 times ]
  * java RandomWalkers 5 1 [ repeated 1048576 times ]
==> passed


RandomWalkers Total: 7/7 tests passed!


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