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

ASSESSMENT SUMMARY

```
Compilation:  PASSED
API:          FAILED (1 error)

SpotBugs:     PASSED
PMD:          PASSED
Checkstyle:   FAILED (0 errors, 1 warning)

Correctness:  48/48 tests passed
Memory:       No tests available for autograding.
Timing:       No tests available for autograding.

Aggregate score: 95.00%
[ Compilation: 5%, API: 5%, Style: 0%, Correctness: 90% ]
```

ASSESSMENT DETAILS

```
The following files were submitted:
----------------------------------
 869 May 24 22:31 RecursiveSquares.java
1.1K May 24 22:31 RevesPuzzle.java
 448 May 24 22:31 TrinomialBrute.java
1.1K May 24 22:31 TrinomialDP.java


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


% javac TrinomialBrute.java
*-----------------------------------------------------------

% javac TrinomialDP.java
*-----------------------------------------------------------

% javac RevesPuzzle.java
*-----------------------------------------------------------

% javac RecursiveSquares.java
*-----------------------------------------------------------


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


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

TrinomialDP:

RevesPuzzle:
The following methods must either be removed or made private:
  * public static void hanoi(int,char,char,char,char,int)
  * public static void honoi(int,char,char,char,int)


RecursiveSquares:

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


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


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


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


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


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

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

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

% custom checkstyle checks for RevesPuzzle.java
*-----------------------------------------------------------
[WARN] RevesPuzzle.java:28:24: The 'main()' method must directly call the public method 'honoi()'. [MainCallsAllPublicMethods]
Checkstyle ends with 0 errors and 1 warning.

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


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


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

Testing correctness of TrinomialBrute
*-----------------------------------------------------------
Running 14 total tests.

Test 1: check output format of main()
  % java TrinomialBrute 3 3
  1

  % java TrinomialBrute 3 2
  3

  % java TrinomialBrute 3 1
  6

  % java TrinomialBrute 3 0
  7

  % java TrinomialBrute 3 -1
  6

==> passed

Test 2: check that main() prints correct value for given n and k
  * n = 3, k = 3
  * n = 3, k = 2
  * n = 3, k = 1
  * n = 3, k = 0
  * n = 3, k = -1
==> passed

Test 3: check that main() is consistent with trinomial()
 * n = 4, |k| <= 4
 * n = 5, |k| <= 5
 * n = 6, |k| <= 6
 * n = 9, |k| <= 9
==> passed

Test 4: check correctness of trinomial() for n = 1
  * trinomial(1, 1)
  * trinomial(1, 0)
  * trinomial(1, -1)
==> passed

Test 5: check correctness of trinomial() for n = 2
  * trinomial(2, 2)
  * trinomial(2, 1)
  * trinomial(2, 0)
  * trinomial(2, -1)
  * trinomial(2, -2)
==> passed

Test 6: check correctness of trinomial() for n = 3
  * trinomial(3, 3)
```

```
    * trinomial(3, 2)
    * trinomial(3, 1)
    * trinomial(3, 0)
    * trinomial(3, -1)
    * trinomial(3, -2)
    * trinomial(3, -3)
  ==> passed

  Test 7: check correctness of trinomial() for k = 0
    * trinomial(1, 0)
    * trinomial(2, 0)
    * trinomial(3, 0)
    * trinomial(4, 0)
    * trinomial(5, 0)
    * trinomial(6, 0)
    * trinomial(7, 0)
    * trinomial(8, 0)
    * trinomial(9, 0)
  ==> passed

  Test 8: check correctness of trinomial() for fixed n and random k > 0
    * n = 4, k in [1, 4]
    * n = 5, k in [1, 5]
    * n = 6, k in [1, 6]
    * n = 7, k in [1, 7]
    * n = 8, k in [1, 8]
    * n = 9, k in [1, 9]
  ==> passed

  Test 9: check correctness of trinomial() for fixed n and random k < 0
    * n = 4, k in [-4, -1]
    * n = 5, k in [-5, -1]
    * n = 6, k in [-6, -1]
    * n = 7, k in [-7, -1]
    * n = 8, k in [-8, -1]
    * n = 9, k in [-9, -1]
  ==> passed

  Test 10: check correctness of trinomial() for n = 0 and k = 0
    * trinomial(0, 0)
  ==> passed

  Test 11: check correctness of trinomial() for k > n
    * n = 1, k in [2, 10]
    * n = 2, k in [3, 10]
    * n = 3, k in [4, 10]
    * n = 4, k in [5, 10]
    * n in [5, 10], k in [11, 50]
    * n in [10, 50], k in [51, 100]
  ==> passed

  Test 12: check correctness of trinomial() for k < -n
    * n = 1, k in [-10, -2]
    * n = 2, k in [-10, -3]
    * n = 3, k in [-10, -4]
    * n = 4, k in [-10, -5]
    * n in [5, 10], k in [-50, -11]
    * n in [10, 50], k in [-100, -51]
  ==> passed

  Test 13: check that trinomial() returns same value when called with same arguments
   * n = 3, |k| <= 3
   * n = 4, |k| <= 4
   * n = 5, |k| <= 5
   * n = 9, |k| <= 9
  ==> passed

  Test 14: check correctness of trinomial() for medium values of n
    * n = 10, |k| <= 10
    * n = 11, |k| <= 11
    * n = 12, |k| <= 12
    * n = 13, |k| <= 13
    * n = 14, |k| <= 14
    * n = 15, |k| <= 15
    * n = 16, |k| <= 16
    * n = 17, |k| <= 17
    * n = 18, |k| <= 18
  ==> passed


  TrinomialBrute Total: 14/14 tests passed!


  ================================================================
  Testing correctness of TrinomialDP
  *-----------------------------------------------------------
  Running 17 total tests.
```

```
Test 1: check output format of main()
  % java TrinomialDP 3 3
  1

  % java TrinomialDP 3 2
  3

  % java TrinomialDP 3 1
  6

  % java TrinomialDP 3 0
  7

  % java TrinomialDP 3 -1
  6

==> passed

Test 2: check that main() prints correct value for given n and k
  * n = 3, k = 3
  * n = 3, k = 2
  * n = 3, k = 1
  * n = 3, k = 0
  * n = 3, k = -1
==> passed

Test 3: check that main() is consistent with trinomial()
 * n = 4, |k| <= 4
 * n = 5, |k| <= 5
 * n = 6, |k| <= 6
 * n = 9, |k| <= 9
==> passed

Test 4: check correctness of trinomial() for n = 1
  * trinomial(1, 1)
  * trinomial(1, 0)
  * trinomial(1, -1)
==> passed

Test 5: check correctness of trinomial() for n = 2
  * trinomial(2, 2)
  * trinomial(2, 1)
  * trinomial(2, 0)
  * trinomial(2, -1)
  * trinomial(2, -2)
==> passed

Test 6: check correctness of trinomial() for n = 3
  * trinomial(3, 3)
  * trinomial(3, 2)
  * trinomial(3, 1)
  * trinomial(3, 0)
  * trinomial(3, -1)
  * trinomial(3, -2)
  * trinomial(3, -3)
==> passed

Test 7: check correctness of trinomial() for k = 0
  * trinomial(1, 0)
  * trinomial(2, 0)
  * trinomial(3, 0)
  * trinomial(4, 0)
  * trinomial(5, 0)
  * trinomial(6, 0)
  * trinomial(7, 0)
  * trinomial(8, 0)
  * trinomial(9, 0)
==> passed

Test 8: check correctness of trinomial() for fixed n and random k > 0
  * n = 4, k in [1, 4]
  * n = 5, k in [1, 5]
  * n = 6, k in [1, 6]
  * n = 7, k in [1, 7]
  * n = 8, k in [1, 8]
  * n = 9, k in [1, 9]
==> passed

Test 9: check correctness of trinomial() for fixed n and random k < 0
  * n = 4, k in [-4, -1]
  * n = 5, k in [-5, -1]
  * n = 6, k in [-6, -1]
  * n = 7, k in [-7, -1]
  * n = 8, k in [-8, -1]
  * n = 9, k in [-9, -1]
==> passed
```

```
Test 10: check correctness of trinomial() for n = 0 and k = 0
  * trinomial(0, 0)
==> passed

Test 11: check correctness of trinomial() for k > n
  * n = 1, k in [2, 10]
  * n = 2, k in [3, 10]
  * n = 3, k in [4, 10]
  * n = 4, k in [5, 10]
  * n in [5, 10], k in [11, 50]
  * n in [10, 50], k in [51, 100]
==> passed

Test 12: check correctness of trinomial() for k < -n
  * n = 1, k in [-10, -2]
  * n = 2, k in [-10, -3]
  * n = 3, k in [-10, -4]
  * n = 4, k in [-10, -5]
  * n in [5, 10], k in [-50, -11]
  * n in [10, 50], k in [-100, -51]
==> passed

Test 13: check that trinomial() returns same value when called with same arguments
 * n = 3, |k| <= 3
 * n = 4, |k| <= 4
 * n = 5, |k| <= 5
 * n = 9, |k| <= 9
==> passed

Test 14: check correctness of trinomial() for medium values of n
  * n = 10, |k| <= 10
  * n = 11, |k| <= 11
  * n = 12, |k| <= 12
  * n = 13, |k| <= 13
  * n = 14, |k| <= 14
  * n = 15, |k| <= 15
  * n = 16, |k| <= 16
  * n = 17, |k| <= 17
  * n = 18, |k| <= 18
  * n = 19, |k| <= 19
==> passed

Test 15: check correctness of trinomial() for larger values of n
  * n = 20, |k| <= 20
  * n = 21, |k| <= 21
  * n = 22, |k| <= 22
  * n = 23, |k| <= 23
  * n = 24, |k| <= 24
  * n = 25, |k| <= 25
  * n = 26, |k| <= 26
  * n = 27, |k| <= 27
  * n = 28, |k| <= 28
  * n = 29, |k| <= 29
==> passed

Test 16: check correctness of trinomial() for larger values of n
  * n = 30, |k| <= 30
  * n = 31, |k| <= 31
  * n = 32, |k| <= 32
  * n = 33, |k| <= 33
  * n = 34, |k| <= 34
  * n = 35, |k| <= 35
  * n = 36, |k| <= 36
  * n = 37, |k| <= 37
  * n = 38, |k| <= 38
  * n = 39, |k| <= 39
==> passed

Test 17: check correctness of trinomial() for large random n and k
         that don't overflow a long
  * n in [40, 45]
  * n in [45, 50]
  * n in [50, 75]
  * n in [75, 100]
  * n in [100, 150]
  * n in [150, 200]
  * n in [200, 500]
==> passed


TrinomialDP Total: 17/17 tests passed!


================================================================
Testing correctness of RevesPuzzle
*-----------------------------------------------------------
```

```
Running 8 total tests.

Test 1: check output format for given values of n
  % java RevesPuzzle 1
  Move disc 1 from A to D

  % java RevesPuzzle 2
  Move disc 1 from A to C
  Move disc 2 from A to D
  Move disc 1 from C to D

  % java RevesPuzzle 3
  Move disc 1 from A to B
  Move disc 2 from A to C
  Move disc 3 from A to D
  Move disc 2 from C to D
  Move disc 1 from B to D

  % java RevesPuzzle 4
  Move disc 1 from A to D
  Move disc 2 from A to B
  Move disc 1 from D to B
  Move disc 3 from A to C
  Move disc 4 from A to D
  Move disc 3 from C to D
  Move disc 1 from B to A
  Move disc 2 from B to D
  Move disc 1 from A to D

  % java RevesPuzzle 5
  Move disc 1 from A to C
  Move disc 2 from A to D
  Move disc 3 from A to B
  Move disc 2 from D to B
  Move disc 1 from C to B
  Move disc 4 from A to C
  Move disc 5 from A to D
  Move disc 4 from C to D
  Move disc 1 from B to C
  Move disc 2 from B to A
  Move disc 3 from B to D
  Move disc 2 from A to D
  Move disc 1 from C to D

==> passed

Test 2: check that output is a valid sequence of moves, ending in goal state
  * n = 1
  * n = 2
  * n = 3
  * n = 4
  * n = 5
==> passed

Test 3: check that output is a valid sequence of moves, ending in goal state
  * n = 6
  * n = 7
  * n = 8
  * n = 9
  * n = 10
==> passed

Test 4: check that output uses a minimum number of moves
  * n = 1
  * n = 2
  * n = 3
  * n = 4
  * n = 5
==> passed

Test 5: check that output uses a minimum number of moves
  * n = 6
  * n = 7
  * n = 8
  * n = 9
  * n = 10
==> passed

Test 6: check that output is correct for given values of n
  * n = 11
  * n = 12
  * n = 13
  * n = 14
  * n = 15
==> passed

Test 7: check that output is correct for a range of values of n
```

```
    * n = [16, 20)
    * n = [20, 30)
    * n = [30, 50)
    * n = [50, 75)
  ==> passed

  Test 8: check program outputs the same results when called with same value for n
    * java RevesPuzzle 4 [ twice ]
    * java RevesPuzzle 5 [ twice ]
    * java RevesPuzzle 6 [ twice ]
  ==> passed


  RevesPuzzle Total: 8/8 tests passed!


  ================================================================
  Testing correctness of RecursiveSquares
  *-----------------------------------------------------------
  Running 9 total tests.

  Test 1: check that main() uses neither standard input nor standard output
    % java RecursiveSquares 1
    [no output]

    % java RecursiveSquares 2
    [no output]

    % java RecursiveSquares 3
    [no output]

    % java RecursiveSquares 4
    [no output]

    % java RecursiveSquares 5
    [no output]

    % java RecursiveSquares 6
    [no output]

  ==> passed

  Test 2: check that main() calls only allowable standard draw methods
    * java RecursiveSquares 1
    * java RecursiveSquares 2
    * java RecursiveSquares 3
    * java RecursiveSquares 4
    * java RecursiveSquares 5
    * java RecursiveSquares 6
  ==> passed

  Test 3: check drawSquare() with fixed arguments
    * drawSquare(0.5, 0.5, 0.5)
    * drawSquare(0.75, 0.25, 0.25)
    * drawSquare(0.25, 0.75, 0.25)
    * drawSquare(0.625, 0.875, 0.125)
    * drawSquare(0.8125, 0.6875, 0.0625)
  ==> passed

  Test 4: check drawSquare() with random arguments
    * calling drawSquare() with x, y, and length in [0.0, 1.0]
    * calling drawSquare() with x, y, and length in [0.0, 10.0]
    * calling drawSquare() with x, y, and length in [-10.0, 10.0]
  ==> passed

  Test 5: check that draw() draws the correct set of squares for fixed arguments
    * draw(1, 0.5, 0.5, 0.5)
    * draw(1, 0.75, 0.25, 0.25)
    * draw(1, 0.25, 0.75, 0.25)
    * draw(1, 0.625, 0.875, 0.125)
    * draw(2, 0.5, 0.5, 0.5)
    * draw(2, 0.75, 0.25, 0.25)
    * draw(3, 0.25, 0.75, 0.25)
  ==> passed

  Test 6: check that draw() draws the correct set of squares for random arguments
    * calling draw() with n = 1 and x, y, and length in [0.0, 1.0]
    * calling draw() with n = 2 and x, y, and length in [0.0, 1.0]
    * calling draw() with n = 1 and x, y, and length in [0.0, 10.0]
    * calling draw() with n = 3 and x, y, and length in [-10.0, 10.0]
  ==> passed

  Test 7: check that main() is consistent with draw(n, 0.5, 0.5, 0.5)
    * n = 1
    * n = 2
    * n = 3
    * n = 4
    * n = 5
    * n = 6
  ==> passed

  Test 8: check that main() draws the correct set of squares
    * java RecursiveSquares 1
    * java RecursiveSquares 2
    * java RecursiveSquares 3
    * java RecursiveSquares 4
    * java RecursiveSquares 5
    * java RecursiveSquares 6
  ==> passed

  Test 9: check that main() draws correct squares in correct order
    * java RecursiveSquares 1
    * java RecursiveSquares 2
    * java RecursiveSquares 3
    * java RecursiveSquares 4
    * java RecursiveSquares 5
    * java RecursiveSquares 6
  ==> passed


  RecursiveSquares Total: 9/9 tests passed!


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