

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

ASSESSMENT SUMMARY

Compilation: **PASSED**
API: **PASSED**

SpotBugs: **PASSED**
PMD: **PASSED**
Checkstyle: **PASSED**

Correctness: **38/37 tests passed**
Memory: **No tests available for autograding.**
Timing: **No tests available for autograding.**

Aggregate score: 102.43%
[Compilation: 5%, API: 5%, Style: 0%, Correctness: 90%]

ASSESSMENT DETAILS

The following files were submitted:

```
-----  
673 Aug  7 01:46 CMYKtoRGB.java  
682 Aug  7 01:46 GreatCircle.java  
215 Aug  7 01:46 HelloGoodbye.java  
192 Aug  7 01:46 HelloWorld.java  
603 Aug  7 01:46 RightTriangle.java
```

```
*****  
*   COMPILING  
*****
```

```
% javac HelloWorld.java
```

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

```
% javac HelloGoodbye.java
```

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

```
% javac RightTriangle.java
```

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

```
% javac GreatCircle.java
```

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

```
% javac CMYKtoRGB.java
```

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

```
=====
```

Checking the APIs of your programs.

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

HelloWorld:

HelloGoodbye:

RightTriangle:

GreatCircle:

CMYKtoRGB:

=====

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

```
% spotbugs *.class
```

```
*_-----
```

=====

```
% pmd .
```

```
*_-----
```

=====

```
% checkstyle *.java
```

```
*_-----
```

```
% custom checkstyle checks for HelloWorld.java
```

```
*_-----
```

```
% custom checkstyle checks for HelloGoodbye.java
```

```
*_-----
```

```
% custom checkstyle checks for RightTriangle.java
```

```
*_-----
```

```
% custom checkstyle checks for GreatCircle.java
```

```
*_-----
```

```
% custom checkstyle checks for CMYKtoRGB.java
```

```
*_-----
```

=====

```
*****
*   TESTING CORRECTNESS
*****
```

```
Testing correctness of HelloWorld
```

```
*_-----
```

```
Running 2 total tests.
```

```
Test 1: check output format
```

```
% java HelloWorld
```

```
Hello, World
```

```
==> passed
```

```
Test 2: check correctness
```

```
* java HelloWorld
```

```
==> passed
```

HelloWorld Total: 2/2 tests passed!

=====

Testing correctness of HelloGoodbye

*-----

Running 6 total tests.

Test 1: check output format

```
% java HelloGoodbye Kevin Bob
Hello Kevin and Bob.
Goodbye Bob and Kevin.
```

```
% java HelloGoodbye Alejandra Bahati
Hello Alejandra and Bahati.
Goodbye Bahati and Alejandra.
```

==> passed

Test 2: check correctness using names from assignment specification

```
* java HelloGoodbye Kevin Bob
* java HelloGoodbye Alejandra Bahati
```

==> passed

Test 3: check correctness using fixed names

```
* java HelloGoodbye Chandra Deshi
* java HelloGoodbye Ayşe María
* java HelloGoodbye Wayan Taiyeo
* java HelloGoodbye Ástfriður Bedřiška
```

==> passed

Test 4: check correctness when two names are the same

```
* java HelloGoodbye Turing Turing
* java HelloGoodbye Lovelace Lovelace
* java HelloGoodbye Hopper Hopper
* java HelloGoodbye Knuth Knuth
```

==> passed

Test 5: check correctness using random names

```
* java HelloGoodbye Sulaiman Ljubodrag
* java HelloGoodbye Yauhen Dinju
* java HelloGoodbye Vikis Lionel
* java HelloGoodbye Albi Herfried
* java HelloGoodbye Dorel Chitra
```

==> passed

Test 6: test correctness using many random names

```
* 10 pairs of random names
* 100 pairs of random names
* 1000 pairs of random names
* 10000 pairs of random names
```

==> passed

HelloGoodbye Total: 6/6 tests passed!

=====

Testing correctness of RightTriangle

*-----

Running 11 total tests.

Test 1: check output format for inputs from assignment specification

```
% java RightTriangle 3 4 5
true
```

```
% java RightTriangle 13 12 5
true
```

```
% java RightTriangle 1 2 3
false
```

```
% java RightTriangle -3 4 -5
false
```

==> passed

Test 2: check correctness of inputs from assignment specification

```
* java RightTriangle 3 4 5
* java RightTriangle 13 12 5
* java RightTriangle -3 4 -5
```

==> passed

Test 3: inputs with $a^2 + b^2 = c^2$

```
* java RightTriangle 4 3 5
* java RightTriangle 5 12 13
* java RightTriangle 15 8 17
* java RightTriangle 7 24 25
* java RightTriangle 20 21 29
* java RightTriangle 35 12 37
* java RightTriangle 9 40 41
* java RightTriangle 28 45 53
* java RightTriangle 12 35 37
* java RightTriangle 60 11 61
* java RightTriangle 16 63 65
* java RightTriangle 16 63 65
* java RightTriangle 56 35 65
* java RightTriangle 55 48 73
* java RightTriangle 13 84 85
* java RightTriangle 13 84 85
* java RightTriangle 36 77 85
* java RightTriangle 39 80 89
* java RightTriangle 65 72 97
```

==> passed

Test 4: inputs with $a^2 + c^2 = b^2$

```
* java RightTriangle 3 5 4
* java RightTriangle 5 13 12
* java RightTriangle 8 17 15
* java RightTriangle 7 25 24
* java RightTriangle 20 29 21
* java RightTriangle 12 37 35
* java RightTriangle 9 41 40
* java RightTriangle 28 53 45
* java RightTriangle 12 37 35
* java RightTriangle 11 61 60
```

==> passed

Test 5: inputs with $b^2 + c^2 = a^2$

```
* java RightTriangle 5 4 3
* java RightTriangle 13 5 12
* java RightTriangle 17 15 8
* java RightTriangle 25 7 24
* java RightTriangle 29 21 20
* java RightTriangle 37 12 35
* java RightTriangle 41 40 9
* java RightTriangle 53 45 28
* java RightTriangle 37 12 35
* java RightTriangle 61 11 60
```

==> passed

Test 6: inputs that are not Pythagorean triples

```
* java RightTriangle 5 5 5
* java RightTriangle 3 4 6
* java RightTriangle 5 12 14
```

==> passed

Test 7: inputs with zeros

```
* java RightTriangle 0 0 1
* java RightTriangle 0 0 0
* java RightTriangle 0 1 1
* java RightTriangle 0 10 10
==> passed
```

Test 8: inputs with negative values

```
* java RightTriangle 3 4 -5
* java RightTriangle -3 4 5
* java RightTriangle -3 -4 5
* java RightTriangle -3 -4 -5
* java RightTriangle -2147483648 -2147483648 -2147483648
* java RightTriangle 0 0 -2147483648
* java RightTriangle -5 -12 13
==> passed
```

Test 9: random Pythagorean triples

```
* 10000 random Pythagorean triples between 1 and 100
* 10000 random Pythagorean triples between 1 and 1000
* 10000 random Pythagorean triples between 1 and 10000
==> passed
```

Test 10: random non-Pythagorean triples

```
* 10000 random non-Pythagorean triples between 1 and 100
* 10000 random non-Pythagorean triples between 1 and 1000
* 10000 random non-Pythagorean triples between 1 and 10000
==> passed
```

Test 11: random Pythagorean triples (large integers)

```
* 10000 random Pythagorean triples between 1 and 100000
* 10000 random Pythagorean triples between 1 and 1000000
* 10000 random Pythagorean triples between 1 and 10000000
* 10000 random Pythagorean triples between 1 and 100000000
==> passed
```

Bonus Test: random non-Pythagorean triples with $(a*a + b*b == c*c)$ or $(a*a + c*c == b*b)$ or $(b*b + c*c == a*a)$ due to arithmetic overflow

```
* 50 random overflow Pythagorean triples between 1 and 100000
* 50 random overflow Pythagorean triples between 1 and 1000000
* 50 random overflow Pythagorean triples between 1 and 10000000
* 50 random overflow Pythagorean triples between 1 and 100000000
==> passed
```

RightTriangle Total: 12/11 tests passed!

=====

Testing correctness of GreatCircle

*-----

Running 11 total tests.

Test 1: check output format for points from assignment specification

```
% java GreatCircle 40.35 74.65 48.87 -2.33
5902.927099258561 kilometers
```

```
% java GreatCircle 60.0 15.0 120.0 105.0
4604.53989281927 kilometers
```

==> passed

Test 2: check distance for points from assignment specification

```
* java GreatCircle 40.35 74.65 48.87 -2.33
* java GreatCircle 60.0 15.0 120.0 105.0
```

==> passed

Test 3: check distance for random pairs of cities

```
* Sitapur, India and Termiz, Uzbekistan
* Las Cruces, United States and Valletta, Malta
```

```
* Kislovodsk, Russia and Jember, Indonesia
* Aracaju, Brazil and Isiro, Congo (Kinshasa)
* Bataysk, Russia and Braşov, Romania
* Qom, Iran and El Oued, Algeria
* Pinsk, Belarus and Rialto, United States
* Vallejo, United States and Banghazi, Libya
* Nukus, Uzbekistan and Faisalabad, Pakistan
* Ravenna, Italy and Nsukka, Nigeria
==> passed
```

Test 4: check distance for corner cases

```
* java GreatCircle 0 0 0 0
* java GreatCircle 90 90 90 90
* java GreatCircle 0 90 0 -90
* java GreatCircle 90 0 -90 0
* java GreatCircle 90 90 -90 0
* java GreatCircle 90 90 -90 -90
* java GreatCircle 0 180 0 0
* java GreatCircle 0 180 0 180
* java GreatCircle 0 0 0 -180
==> passed
```

Test 5: check that distance between (x1, y1) and (x2, y2)
equals the distance between (x2, y2) and (x1, y1)

```
* 1000 random points with latitude and longitude in [20.0, 70.0]
* 1000 random points with latitude and longitude in [-70.0, -20.0]
* 1000 random points with latitude and longitude in [-90.0, 90.0]
* 1000 random points with latitude in [-90.0, 90.0] and longitude in [-180.0, 180.0]
==> passed
```

Test 6: check that distance between a point and itself is 0

```
* 1000 random points with latitude and longitude [20.0, 70.0]
* 1000 random points with latitude and longitude [-70.0, -20.0]
* 1000 random points with latitude and longitude [-90.0, 90.0]
* 1000 random points with latitude [-90.0, 90.0] and longitude [-180.0, 180.0]
==> passed
```

Test 7: check that distance between two antipodal points = $\pi * \text{radius}$

```
* 10 random antipodal points
* 100 random antipodal points
* 1000 random antipodal points
==> passed
```

Test 8: check distance of random pairs of cities

```
* 100 random pairs of cities
* 1000 random pairs of cities
* 10000 random pairs of cities
==> passed
```

Test 9: check distance of random pairs of points

```
* 1000 random points with latitude and longitude [20.0, 70.0]
* 1000 random points with latitude and longitude [-70.0, -20.0]
* 1000 random points with latitude and longitude [-90.0, 90.0]
* 1000 random points with latitude [-90.0, 90.0] and longitude [-180.0, 180.0]
==> passed
```

Test 10: check distance of random pairs of nearby points

```
* 1000 random pairs of points within 1.000000 kilometers
* 1000 random pairs of points within 0.010000 kilometers
* 1000 random pairs of points within 0.000100 kilometers
* 1000 random pairs of points within 0.000001 kilometers
==> passed
```

Test 11: check distance of random pairs of nearly antipodal points

```
* 1000 random pairs of points within 1.000000 kilometers of being antipodal
* 1000 random pairs of points within 0.010000 kilometers of being antipodal
* 1000 random pairs of points within 0.000100 kilometers of being antipodal
* 1000 random pairs of points within 0.000001 kilometers of being antipodal
==> passed
```

GreatCircle Total: 11/11 tests passed!

=====

Testing correctness of CMYKtoRGB

*-----

Running 7 total tests.

Test 1: check output format

% java CMYKtoRGB 0.0 1.0 0.0 0.0

red = 255

green = 0

blue = 255

% java CMYKtoRGB 0.0 0.4392156862745098 1.0 0.0

red = 255

green = 143

blue = 0

Test 2: check correctness of inputs from assignment specification

* java CMYKtoRGB 0.0 1.0 0.0 0.0

* java CMYKtoRGB 0.0 0.4392156862745098 1.0 0.0

Test 3: check various inputs

* java CMYKtoRGB 0.18 0.32 0.0 0.29

* java CMYKtoRGB 1.0 0.58 0.0 0.33

* java CMYKtoRGB 0.0 1.0 0.75 0.50

* java CMYKtoRGB 0.0 0.14 0.70 0.15

Test 4: check corner cases

* java CMYKtoRGB 0.0 0.0 0.0 0.0

* java CMYKtoRGB 1.0 0.0 0.0 0.0

* java CMYKtoRGB 0.0 1.0 0.0 0.0

* java CMYKtoRGB 0.0 0.0 1.0 0.0

* java CMYKtoRGB 0.0 0.0 0.0 1.0

* java CMYKtoRGB 1.0 1.0 0.0 0.0

* java CMYKtoRGB 1.0 0.0 1.0 0.0

* java CMYKtoRGB 1.0 0.0 0.0 1.0

* java CMYKtoRGB 0.0 1.0 1.0 0.0

* java CMYKtoRGB 0.0 1.0 0.0 1.0

* java CMYKtoRGB 0.0 0.0 1.0 1.0

* java CMYKtoRGB 1.0 1.0 1.0 0.0

* java CMYKtoRGB 1.0 1.0 0.0 1.0

* java CMYKtoRGB 1.0 0.0 1.0 1.0

* java CMYKtoRGB 0.0 1.0 1.0 1.0

* java CMYKtoRGB 1.0 1.0 1.0 1.0

Test 5: check that various RGB values can be generated

* (228, 132, 0) Fulvous

* (238, 130, 238) Violet (Web)

* (159, 29, 53) Vivid Burgundy

* (251, 153, 2) Orange (Ryb)

* (139, 0, 139) Dark Magenta

* (255, 239, 0) Yellow (Process)

Test 6: check that various RGB values can be generated

* 10 random RGB values

* 100 random RGB values

* 1000 random RGB values

* 10000 random RGB values

Test 7: check random inputs

- * 100 random CMYK values that are multiples of 0.5
- * 100 random CMYK values that are multiples of 0.25
- * 100 random CMYK values that are multiples of 0.125
- * 1000 random CMYK values that are multiples of 0.0625
- * 1000 random CMYK values that are multiples of 0.03125
- * 1000 random CMYK values that are multiples of 0.015625

==> passed

CMYKtoRGB Total: 7/7 tests passed!

=====