Java – Calculations

5.01 CurrentTime.java

Write a program that outputs the current military time by invoking System.currentTimeMillis() and performing the necessary calculations. The currentTimeMillis() method returns the current number of milliseconds that has elapsed since midnight, January 1, 1970 GMT(Greenwich Mean Time).

% and / will be useful.

You may write your code in main and google GMT Time to validate the result.

Sample Execution

```
Current Time 15:32:25 GMT
```

5.02 Triangle.java

Update the Triangle class by adding a perimeter and area method. The formula for semi-perimeter and area:

s = perimeter / 2

area = $\sqrt{s(s-a)(s-b)(s-c)}$

Note: you cannot multiply using parenthesis in java. Only * multiplies.

Sample Execution

```
Triangle{a=3, b=3, c=3}
Area == 3.8971

Triangle{a=7, b=8, c=9}
Area == 26.8328

Triangle{a=10, b=9, c=11}
Area == 42.4264
```

5.03 Point.java

Write a Point class that represents a location in (x, y) coordinate space, specified in integer precision.

Point() - constructs and initializes a point at the origin (0, 0) of the coordinate space.

Point(int x, int y) - initializes a point at the specified (x,y) location in the coordinate space.

Point(Point p) - initializes a point with the same location as the specified Point object.

void move(int x, int y) - moves this point to the specified location in the (x,y) coordinate plane.

void setLocation(Point p) -sets the location of the point to the specified location.

double distance(Point p) -returns the distance from this Point to a specified Point.

double slope(Point p) -returns the slope formed from the line by this Point to a specified Point.

void translate(int dx, int dy) - translates this point, at location (x,y), by dx along the x axis and dy along the y axis so that it now represents the point (x+dx,y+dy).

Add **getX**(), **getY**(), **setX**(int x), **setY**(int y) and a **toString**(). The toString should match the output precisely (only has one space).

Sample Data

```
Point origin = new Point();
Point pt1 = new Point(4,7);
Point pt2 = new Point(-22,13);
Point clone = new Point(pt1);

System.out.printf("%s\n", origin);
System.out.printf("%s\n", pt1);
System.out.printf("%s\n", clone);
System.out.printf("%s\n", pt2);

// test slopes
System.out.printf("Slope: %.2f\n", origin.slope(pt1));
System.out.printf("Slope: %.2f\n", origin.slope(pt2));
System.out.printf("Slope: %.2f\n", pt1.slope(pt2));

// test distance
System.out.printf("Distance: %.2f\n", origin.distance(pt1));
System.out.printf("Distance: %.2f\n", origin.distance(pt2));
System.out.printf("Distance: %.2f\n", pt1.distance(pt2));

// add test cases for all methods
```

Sample Execution

```
Point{x=0, y=0}
Point{x=4, y=7}
Point{x=4, y=7}
Point{x=-22, y=13}

Slope: 1.75
Slope: -0.59
Slope: -0.23

Distance: 8.06
Distance: 25.55
Distance: 26.68
```

5.04 Pentagon.java

Write a program that prompts the user to enter the length from the center of a polygon to a vertex. The formula for side length and area are below. Call the static function from main to test the code.

s = 2 r sin $\frac{\pi}{5}$

area = $\frac{5 * s^2}{4 * \tan\frac{\pi}{5}}$

static double **areaOfPentagon**(double r) – returns the area of a polygon

Sample Execution

```
Enter the length from vertex to center: 7.5
The area is 133.742
```

5.05 Triangle.java

Update the Triangle class by adding three instance variables for angles A, B and C. Update the toString to include the angles as depicted in the sample execution. The Law of Cosines may be used to calculate all 3 angles. The Math method acos(double d) returns the angle in radians(not required to know or use but π == 180 degrees) and toDegrees(double d) converts radians to degrees. Add one static method to the class and invoke it three times.

$c^2 = a^2 + b^2$ - 2ab cos C

You must rearrange this formula using basic arithmetic to solve for cos C. Then apply the arccosine. Watch out for *integer division* as this will yield incorrect results.

static double **lawOfCosines**(int sideA, int sideB, int sideC) - returns the angle C given 3 sides of a triangle. The parameters sideA, sideB, and sideC can be any 3 sides of a triangle where side angle C is opposite of sideC.

String.format("%.2f", 3.14159) -> returns "3.14"

angleA = lawOfCosines(…)

// call two more times for angleB and angleC

Sample Execution

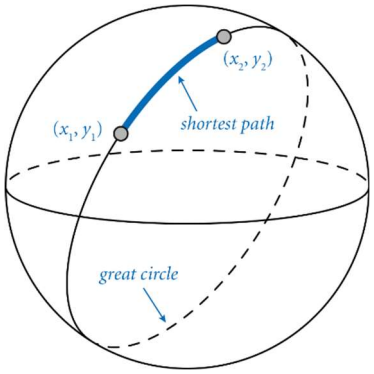
```
Triangle{a=3, b=3, c=3, A=60.00, B=60.00, C=60.00}

Triangle{a=3, b=4, c=5, A=36.87, B=53.13, C=90.00}

Triangle{a=7, b=8, c=9, A=48.19, B=58.41, C=73.40}
```

5.06 GreatCircle.java (advanced from Princeton)

The *great-circle distance* is the length of the shortest path between two points (x1, y1) and (x2, y2) on the surface of a sphere, where the path is constrained to be along the surface.



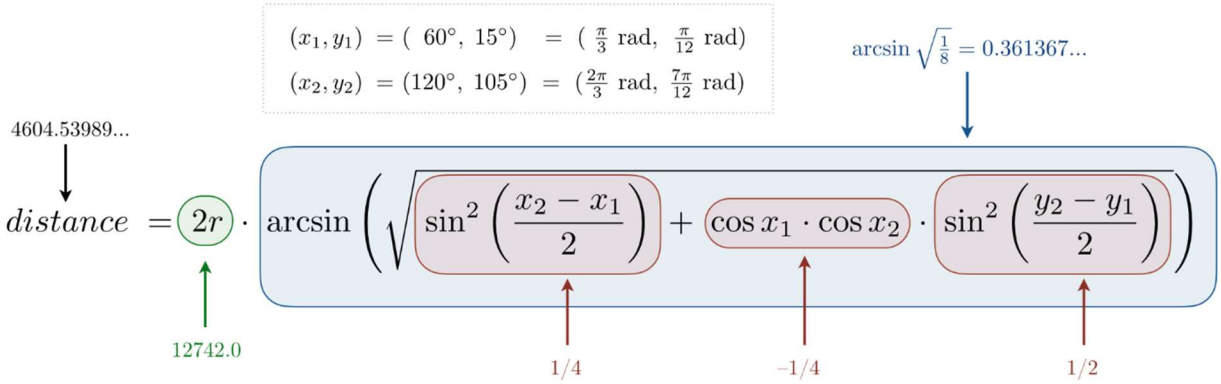
Write a program GreatCircle.java that takes four double command-line arguments x1, y1, x2, and y2—the latitude and longitude (in degrees) of two points on the surface of the earth—and prints the great-circle distance (in kilometers) between them. Use the following Haversine formula:

distance = 2r arcsin($\sqrt{sin^2(\frac{x2-x1}{2}) + \cos x1 \cos x2 \ sin^2(\frac{y2-y1}{2})}$

where r equals 6,371.0 is the mean radius of the Earth (in kilometers).

Use Math.toRadians() to convert from degrees to radians.

Helpful Hints: First, decompose a formula into smaller pieces and store each piece in a separate variable. Next, pick input values for which computing the pieces by hand are relatively easy. Finally, check that each piece matches the value you expected. For example, the great-circle distance between (60°, 15°) and (120°, 105°) is approximately 4,604.53989 kilometers.



Sample Execution

```
Desktop/CS1/Computations> javac GreatCircle.java

Desktop/CS1/Computations> java GreatCircle 40.35 74.65 48.87 -2.33    // Princeton to Paris
5902.927099258561 kilometers

Desktop/CS1/Computations> java GreatCircle 60.0 15.0 120.0 105.0      // for debugging
4604.53989281927 kilometers
```