CSE/IT 213: Homework 2

- Due: Sunday, February 18 @ 11:59pm
- This is an individual homework

Please read Google's Java Style:
URL: https://google.github.io/styleguide/javaguide.html

Make sure your code conforms to Google's Java Style and that you use Javadocs for all methods. The UML does not show setter and getter methods but you need to provide setter and getter methods for all private instance variables.

Problems

Update Rectangle class

1. Update the Rectangle class from Homework 1 so it uses a Point class to store the x and y coordinates of the lower, left hand corner. Your classes should have the following structure:

```
Rectangle
---------------------
p : Point
width : double
height : double
---------------------
Rectangle()
Rectangle(width : double, height : double)
Rectangle(p : Point, width : double, height : double)
Rectangle(x : double, y : double, width : double, height : double)
---------------------
area() : double
perimeter() : double
diagonalLength() : double
distanceFromOrigin() : double
```

```
manhattanDistanceFromOrigin() : double
toString() : String //override method
```

```
Point
---------------------
x : double
y : double
r : double
theta : double
type : enum Type{CARTESIAN, POLAR}
---------------------
Point(x : double, y : double, type: Type)

cartesianInstance(x : double, y : double)
polarInstance(r : double, theta : double)
---------------------
//Euclidean distance in two dimensions
euclideanDistance(q : Point) //object method
euclideanDistance(p : Point, q : Point) //static method
//Manahattan (taxicab) distance
manhattanDistance(q : Point) //object method
manhattanDistance(p : Point, q : Point) //static method
toString : String
```

Rectangle class

You only need one constructor that does initialization. All other constructors should call that constructor using `this`. The `Rectangle(x : double, y : double, width : double, height : double)` is the constructor that does the initialization. Assume all points are Cartesian.

Update the Rectangle distance methods so they call the Point's distance methods and add the `manhattanDistanceFromOrigin()` method to the Rectangle class. Like the other distance methods, it calls the distance methods from the Point class.

The Rectangle class, should still allow you to change the `x` and `y` of the lower left hand corner of the rectangle. However, the Rectangle's setters and getters for x and y use Point's setters and getters.

Point class

For the Point class, you are going to store both the Cartesian and polar coordinates of the point. You need to keep track of the type of coordinate system you initialized the point with. Use the type in `toString()` to return the appropriate representation.

The Manhattan Distance (taxicab distance) between two points (x_1, y_1) and (x_2, y_2) is defined in two dimensions as $M_d = |x_2 - x_1| + |y_2 - y_1|$.

Make sure you supply setters and getters for the Point class and that they change all appropriate values. That is, as you are storing both the Cartesian and polar representation of the point, a change to x changes r and a change to r changes x and y, etc.

Wind Chill

2. Write a program that calculates the Wind Chill. You will write four classes:

```
Temperature
------------------
kelvin : double
fahrenheit : double
celsius : double
type : enum Type {FAHRENHEIT, CELSIUS, KELVIN}
-------------------
Temperature(value : double, Type)

kelvinInstance(value: double)
celsiusInstance(value : double)
fahrenheitInstance(value : double)
-------------------
toString()


WindSpeed
-------------------
mph : double //the wind speed in miles per hour
-------------------
WindSpeed(double)
-------------------
toMetersPerSecond()
toString()


WindChill
```

```
-------------------

getWindChill(t : Temperature, w : WindSpeed)
getWindChillWatts(t : Temperature, w : WindSpeed)


Weather
-------------------
main() //the driver class
```

Temperature class

For each temperature object, you will store all three representations of the temperature (Celsius, Fahrenheit, Kelvin). The constructor should perform all the conversions. The conversion are as follows:

to convert Celsius to Kelvin: $K = C + 273.15$

to convert Fahrenheit to Kelvin: $K = (F + 459.67) \times \frac{5}{9}$

to convert Celsius to Fahrenheit: $C = \frac{5}{9}(F - 32)$

Write setters and getters for all instance variables. Like the Point class, the setters will change multiple values as changing the Fahrenheit temperature changes the Celsius and Kelvin temperature, etc.

The `toString()` method should return a string with the temperature value and the correct unit.

WindSpeed class

This class has only one instance variable, `mph`, which stores the wind speed in miles per hour. Add the appropriate setter and getter. To convert from miles per hour to meters per second use 1 mile per hour = 0.44704 meters per second

The `toString()` method should return a string with the wind speed in miles per hour.

WindChill class

You can find information on how to calculate wind chill from:

`http://www.nws.noaa.gov/om/cold/wind_chill.shtml`

and

`https://www.weather.gov/media/epz/wxcalc/windChill.pdf`

You are calculating the New Wind Chill Index and the Wind Chill in Watts per meter squared. As the class just does calculations you do not need any constructors only static methods for the two calculations.

Heed the warning on the webpage that wind chill is only defined for temperatures at or below 50 degrees F and wind speeds above 3 mph. Return 0.0 for wind chill if it cannot be calculated.

Weather class

The Weather class is a driver program that uses a Scanner to get user input of the temperature, the unit of the temperature, the wind speed in miles per hour, and calculates both the New Wind Chill Index and the Wind Chill in Watts per meter squared. Use the DecimalFormat class to format the output to two decimal places. Do not use printf.

Sample Output

```
enter a temperature: 32
enter temperature units [C/F/K]: F
enter wind speed (mph): 10
the current wind chill is 23.73
or in Watts per meter square the wind chill is 1040.24
```

Unit testing

In addition to the Weather program, unit test your Temperature (include setters), WindSpeed, and WindChill classes.

Javadoc

Create Javadoc documentation for the Temperature, WindSpeed, and WindChill classes.

Submission

Create a jar named `cse213_hw2_firstname_lastname.jar` with all your *.java files, your unit test classes, and the Javadoc documentation for the weather classes.

Upload the jar file to Canvas before the due date.