CSC 360 Assignment # 4
Total Points: 2 x 20 = 40

Submission Guidelines:

- Problem 1: Submit only the **PowersUsername.java** file on Canvas, by substituting the Username with your NKU username.
- Problem 2: Submit only the **PlayerSorter.java** file on Canvas.

Problem 1: Computing Powers

This problem is a case study in using recursive thinking to improve the efficiency of an iterative algorithm. You will write a sequence of methods for the exponentiation of floating point numbers. Parts I and II involve writing rather simple-minded iterative and recursive methods for the task. In Parts III and IV you use more sophisticated recurrence relations in order to write more efficient recursive methods. In Part V you convert your method from Part IV into an iterative method that is far more efficient than the Part I iterative method.

Put all of your code for this project into a **single class file – PowersUsername.java,** by substituting the Username with your NKU username. All of your methods should be **static**.

Part I:
Write an iterative method called *power1* to compute b^n, where *b* is of type double and *n* is an integer ≥ 0. Use a simple for-loop that repeatedly (*n* times) multiplies an accumulator variable by *b*.

Part II:
Write a recursive method *power2* that accomplishes the same task as *power1*, but is based on the following recurrence relation:

$b^0 = 1$
$b^n = b * b^{n-1}$ if $n > 0$

Part III:
Write a recursive method *power3* that is identical to *power2* except that it is based on this recurrence relation

$b^0 = 1$
$b^n = (b^{n/2})^2$ if $n > 0$ and n is even
$b^n = b * (b^{n/2})^2$ if $n > 0$ and n is odd (Note: This equation is not true in math. Why is it true in Java?)

Note: If n is a large exponent, then *power3* should perform far fewer multiplications than *power2*. In particular, when computing something like $(b^{n/2})^2$, there is no need to compute $b^{n/2}$ twice. Rather, compute it once, store it in a variable, and then compute the result of multiplying the variable by itself.

Part IV:
Write a tail recursive helper method called *multPow*, that computes the value of $a*b^n$. Base your implementation on the following recurrence relation:

$a*b^0 = a$
$a*b^n = a*(b^2)^{n/2}$ if $n > 0$ and n is even

$a*b^n = (a*b)*(b^2)^{n/2}$ if n > 0 and n is odd

Then write a method called *power4* that computes b^n simply by making the call *multPow(1, b, n)*. Note that in this approach, the extra parameter *a* used by the helper method is serving as an accumulator for the result.

Part V:
Write an iterative method *power5* to compute b^n. Write it in such a way that the number of multiplications performed by *power5* is no more than the number performed by *power4*. (Hint: Base your solution to Part V on your solution to Part IV. Declare *a* as a local variable that is initialized to 1 and that eventually accumulates the result of the calculation.) Note that, in general, *power5* requires far fewer multiplications than *power1*.

Main method:
Write a main method to test your methods from Parts I – V. It should ask the user for *b* and *n*, then compute and display the results. Call the *Math.pow* method in order to check your results. Display the results from *Math.pow* first. Then display the results from your methods. Also display the number of multiplications performed by each of your methods. In order to count the number of multiplications, you may use a "global variable" that is modified by the power methods as a side-effect, as demonstrated below:

```
public class Powers
{
	private static int multiplications;  // "global variable" for counting the
	                                     //    number    of    multiplications
	                                     performed // by each method
	public static void main(String[] args)
	{
		...
		multiplications = 0;
		System.out.println("\npower1(" + base + ", " + n + ") = " + power1(base,
		n));
		System.out.println("Multiplications = " + multiplications);

		multiplications = 0;
		...
	}

	public static double power1(double base, int n)
	// Returns base to the n-th power.
	// Iterative method.
	{
		...
		for (...)
		{
			multiplications++;
			result *= base;
		}
		return result;
	}
	...
}
```

Note: It is a good idea to test code as you develop it. It is recommended that you write the main method code that tests Part I as soon as (or even before) Part I is finished.

Sample session with a completed program:

Enter a decimal number: 1.001
Enter a non-negative integer exponent: 1000

Computing 1.001 to the power 1000:
Math.pow(1.001, 1000) = 2.7169239322355936

power1(1.001, 1000) = 2.7169239322355985
Multiplications = 1000

power2(1.001, 1000) = 2.7169239322355985
Multiplications = 1000

power3(1.001, 1000) = 2.716923932235485
Multiplications = 16

power4(1.001, 1000) = 2.7169239322355203
Multiplications = 16

power5(1.001, 1000) = 2.7169239322355203
Multiplications = 16

Note:
In Parts I-V, your code should ***not*** call any methods from java.lang.Math. For example, in those methods, if you need to compute b^2, **do not** write Math.pow(b, 2). Instead, write b * b, and count the multiplication.

Problem 2: Priority queue using multiple comparators

In this program, we will use an input file containing the names of players and their points, and then sort the list of players in two ways: (a) based on the length of their names, and (b) their scores. Follow the descriptions below to develop your program:

- Input file: ***players.txt***
    - Input format in file: *FirstName LastName Score*

- Create a file ***PlayerSorter.java***. The file will have 4 classes:
    - *PlayerSorter, Player, NameLengthComparator, ScoreCompartor*

- **public class PlayerSorter**
    - This class will contain the *main()* method, and will perform the following actions:
        - Read the inputs from the *players.txt* file. You must reach the input file with using the try-with-resources format, and catch the *FileNotFoundException*. Save the inputs as *Player* instances in *ArrayList<Player> players*.
        - Create a priority queue *playerNameLenPQ* with the comparator *NameLengthComparator* and add all the players stored in the arraylist *players*.
        - Create another priority queue *playerScorePQ* with the comparator *ScoreComparator* and add all the players stored in the arraylist *players*.
        - Use the *removeAndPrintQueue()* method to print out the details of the *Player* instances stored in *playerNameLenPQ*.
        - Use the *removeAndPrintQueue()* method to print out the details of the *Player* instances stored in *playerScorePQ*.
    - Write a method *removeAndPrintQueue(PriorityQueue<Player> pq)* to remove and print out the instances of *Player* stored in the priority queue one by one, in this format:
        - *FirstName <single space> LastName <tab space> Score*

- **class Player**
    - Data variables: *String fName*, *String lName*, and *Double score*.
    - Public constructor
        - *public Player(String fName, String lName, Double score)*

- **class NameLengthComparator implements java.util.Comparator<Player>**
    - Override compare method
    - Compare length of (first-name + last-name) and return -1/0/+1 accordingly.

- **class ScoreComparator implements java.util.Comparator<Player>**
    - Override compare method.
    - Compare score and return -1/0/+1 accordingly.

The standard implementation description for the *compare()* method is as follows: *Returns a negative value if element1 is less than element2, a positive value if element1 is greater than/ element2, and zero if they are equal.*

Submission: Submit the PlayerSorter.java file on Canvas.
Sample input file: players.txt (included)
Sample output:

```
--------- Name comparator -----------
John Drew       20.69
Bob Pettit      26.36
Larry Bird      24.29
Bob McAdoo      22.05
Jerry West      27.03
Rick Barry      23.17
Dwyane Wade     21.98
Elvin Hayes     20.96
Karl Malone     25.02
Kobe Bryant     24.99
Paul Arizin     22.81
Elgin Baylor    27.36
Geoff Petrie    21.82
Alex English    21.47
Bradley Beal    20.96
LeBron James    27.07
James Harden    25.16
Chris Webber    20.68
George Mikan    23.13
Kevin Durant    27.02
Bernard King    22.49
Kyrie Irving    22.43
Julius Erving   21.97
Pete Maravich   24.24
Blake Griffin   21.67
George Gervin   26.18
Anthony Davis   24.01
Patrick Ewing   20.98
Stephen Curry   23.49
Dirk Nowitzki   20.74
John Havlicek   20.78
Allen Iverson   26.66
Glenn Robinson  20.69
Michael Jordan  30.12
Adrian Dantley  24.27
David Robinson  21.06
Damian Lillard  24.21
Mitch Richmond  21.0
David Thompson  22.13
Carmelo Anthony 23.57
Oscar Robertson 25.68
Hakeem Olajuwon 21.77
Charles Barkley 22.14
Shaquille O'Neal        23.69
DeMarcus Cousins        21.25
Wilt Chamberlain        30.07
Billy Cunningham        20.83
Dominique Wilkins       24.83
Russell Westbrook       23.25
Kareem Abdul-Jabbar     24.61
--------- Score Comparator ----------
Chris Webber    20.68
Glenn Robinson  20.69
John Drew       20.69
Dirk Nowitzki   20.74
John Havlicek   20.78
Billy Cunningham        20.83
Elvin Hayes     20.96
Bradley Beal    20.96
Patrick Ewing   20.98
```

Mitch Richmond	21.0	
David Robinson	21.06	
DeMarcus Cousins		21.25
Alex English	21.47	
Blake Griffin	21.67	
Hakeem Olajuwon	21.77	
Geoff Petrie	21.82	
Julius Erving	21.97	
Dwyane Wade	21.98	
Bob McAdoo	22.05	
David Thompson	22.13	
Charles Barkley	22.14	
Kyrie Irving	22.43	
Bernard King	22.49	
Paul Arizin	22.81	
George Mikan	23.13	
Rick Barry	23.17	
Russell Westbrook		23.25
Stephen Curry	23.49	
Carmelo Anthony	23.57	
Shaquille O'Neal		23.69
Anthony Davis	24.01	
Damian Lillard	24.21	
Pete Maravich	24.24	
Adrian Dantley	24.27	
Larry Bird	24.29	
Kareem Abdul-Jabbar		24.61
Dominique Wilkins		24.83
Kobe Bryant	24.99	
Karl Malone	25.02	
James Harden	25.16	
Oscar Robertson	25.68	
George Gervin	26.18	
Bob Pettit	26.36	
Allen Iverson	26.66	
Kevin Durant	27.02	
Jerry West	27.03	
LeBron James	27.07	
Elgin Baylor	27.36	
Wilt Chamberlain		30.07
Michael Jordan	30.12	