ITSE 2417- Fall 2016
Program Set #5

See Java Grading /Program Guide Sheet for directions and grading/submission information.

1. Judo is a type martial arts that became an Olympic sport in 1964. Write a Java program that determines the winners of individual Olympic Judo matches. Judo has three scores:

- An Ippon is a terminating score. Only one can be scored and the match is terminated.
- A Waza-ari is an accumulation and terminating score. Two waza-ari equals an ippon.
- A Yuko is an accumulating score only.

There is always a winner in Judo match (no ties). The points given will reflect the score differential or winning score. For example, if both players have a waza-ari and one has a yuko - the differential is yuko. In Judo the highest quality score wins, so the scoreboard is laid out left to right to show the score like a number. Scores are from highest to lowest quality moving from left to right. The contestants are referred to by the color of the robes they wear: blue or white.

	I	**W**	**Y**
Blue	0	0	2
White	0	1	0

On the scoreboard above, white's single waza-ari beats the lesser quality of blue's 2 yuko thus wins the match. Therefore, scoring in Judo is lexicographic- a waza-ari beat any number of yuko; however, a waza-ari and yuko beat a waza-ari and no yuko.

Penalties are either minor (shido) or major (hansoku-make). A maximum of four shido can be given in a contest. After a fourth shido the contest is stopped and the opponent will be awarded an ippon. A direct (immediate) hansoku-make can be given and the player is excluded from the contest; the opponent will be awarded an ippon. If there is a tie on the scoreboard (i.e. both contestants have the same score) then the winner is determined by the fewest number of shidos earned. This victory is noted as winner by penalty. On a scoreboard penalties are designated in the rightmost column.

	I	**W**	**Y**	**P**
Blue	0	0	2	0
White	0	1	0	S1

Input will come from a text file. The first line of input will contain a single integer `n` which will contain how many matches to follow. For each match, there will be two lines with each line containing:

- The players color followed by a space
- Four integers indicating the quality score (`I W Y`) and penalties (`P`) in order, each separated by a space. A hansoku-make will be designated with a `-1`.

Output to the screen: the match, a scoreboard, the winning player's color and the winning grade. Penalties are designated with a letter `S` before the number, and a hansoku-make is designated with the letter `H` only. Output should look similar to below.

Sample File:

```
3
Blue 0 0 1 0
White 0 0 0 0
Blue 0 1 1 3
White 0 0 0 2
Blue 0 0 0 1
White 0 2 0 2
```

Sample Run:

```
Enter the file name: judoscores.txt

Match 1:
       I W Y P
Blue : 0 0 1 0
White: 0 0 0 0
===============
Blue winner by Yuko

Match 2:
       I W Y P
Blue : 0 1 1 S3
White: 0 0 0 S2
===============
Blue winner by Waza-ari

       I W Y P
Blue : 0 0 0 S1
White: 1 0 0 S2
===============
White winner by Ippon
```

Note in the third match above two waza-ari equal an ippon; therefore, is noted in the final scoreboard. Let the user input the file name from the keyboard. Use any appropriate Java Collection classes.

Name the program: `JudoMatchesXX.java`, where `XX` are your initials.

2. The aim of wrestling is to have two players snare each other in a described zone, each trying to move their foe into point-scoring holds and hurls or a match-winning fall. At the Olympic level, freestyle wrestling matches consists of two three-minute periods, with a thirty second break between them. A match may be won by:

- A fall
- By injury, withdrawal, default, disqualification of the opponent
- By technical superiority
- By points

In case of a tie by points the winner shall be declared by successively (in order) considering:

- The highest value of holds (points).
- The lowest number of cautions.
- The last technical points scored.

If a wrestler receives three cautions during a match he/she is disqualified `(EX)`. Classification points are also awarded to a wrestler to determine final ranking in a particular contest (with abbreviations).

- 5 points for the winner and 0 for the loser:
    - Victory by fall (with or without technical point for the loser) (VT)
    - Injury (VB)
    - Withdrawal (VA)
    - Default (Forfeit) (VF)
    - Disqualification (competition or contest) (EV or EX)
- 4 points for the winner and 0 for the loser (ST):
    - Victory by technical superiority (10 points in freestyle during one of the two periods), with the loser scoring no technical points
- 4 points for the winner and 1 point for the loser (SP):
    - Victory by technical superiority during one of the two periods with loser scoring technical points.
- 3 points for the winner and 0 for the loser (PO):
    - When the wrestler wins at the end of the two periods by 1 to 9 points in Freestyle with the loser scoring no point.
- 3 points for the winner and 1 point for the loser (PP):
    - When the bout ends by a victory by points at the end of the regular time and the loser scoring one or several technical points.
- 0 point for the red wrestler and 0 point for the blue wrestler (EZ):
    - In case both wrestlers have been disqualified due to infraction to the rules.

Write a Java program that determines a winner between two wrestlers (red or blue) and the classification points associated with the victory. A sample input file would look like the following:

```
5
Red   0  1  1  -
Blue  1  1  0  -
Red   0  0  1  -
Blue  1  2  0  -
Red   4  1 2  1  -
Blue  1 1  2  0  -
Red   1  1  0  -
Blue  0  2  2  -
Red   1  1  2  -
Blue  1  1  2  X
```

where the first line will indicate the number matches in the data set (`m`). For each of the next `m` lines:
The `Red` player and `Blue` players on separate lines where the first number(s) indicate the point value(s) earned in the first period each separated by a single space, then two spaces, followed by the point value(s) earned in the second period each separated by a single space, then two spaces, the number of cautions earned in the game, then two spaces followed by the player who scored the last technical point (where an `X` indicates the player who scored last, otherwise a – dash will be indicated). Output should look similar to the following (based on the example above):

```
File name: wrestle.txt

Match 1
Blue Winner- Technical Points: 2-1
             Classification Points: PP 3:1

Match 2
Blue Winner- Technical Points: 3-0
             Classification Points: PO 3:0

Match 3
Red Winner- Technical Points: 7-4
            Classification Points: PP 3:1

Match 4
Blue Winner- Technical Points: 2-2
             Classification Points: PP 3:1

Match 5
Blue Winner- Technical Points: 2-2
             Classification Points: PP 3:1
```

From the output from above, in Match 1 the Blue is the victor and Red has one caution. Note the PP abbreviation for the classification. Red scored 3 of its 7 points in period two, while Blue score 2 of its 4 points in the first period in Match 3. With Match 4, Blue wins since it scored a higher point value hold (2) which has a higher priority than the number of cautions, which Red had none. Finally, in Match 5, Blue wins, with all other values being equal, the X notation signified Blue scored the last technical point in the match. Let the user enter the file name from the keyboard. Use any appropriate Java Collection classes.

Name the program: `WrestlingXX.java`, where XX are your initials.

3. Ackermann's function is a recursive mathematical algorithm named after W. Ackermann that can be used to test how well a computer performs recursion. It uses the following logic:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

Write a non-recursive method and Java test program to calculate Ackermann's function. Use a stack to keep track of the parameters for each recursive call. Since the value of the method can appear as a parameter in a recursive call, we can simplify the algorithm by, at each pass, first popping the parameters from the stack and then pushing the method value or new pair of parameters onto the stack. The method must be non-recursive and generic for full credit. Output must show the steps of the recursion through each pass of the entered values. Input only positive values up to and including 3. Output should look similar to below.

Sample Run:

```
Enter the arguments:  2 2

A(2,2)

A(1,A(2,1))
A(1,A(1,A(2,0)))
A(1,A(1,A(1,1)))
A(1,A(1,A(0,A(1,0))))
A(1,A(1,A(0,A(0,1))))
A(1,A(1,A(0,2)))
A(1,A(1,3))
A(1,A(0,A(1,2)))
A(1,A(0,A(0,A(1,1))))
A(1,A(0,A(0,A(0,A(1,0)))))
A(1,A(0,A(0,A(0,A(0,1)))))
A(1,A(0,A(0,A(0,2))))
A(1,A(0,A(0,3)))
A(1,A(0,4))
A(1,5)
A(0,A(1,4))
A(0,A(0,A(1,3)))
A(0,A(0,A(0,A(1,2))))
A(0,A(0,A(0,A(0,A(1,1)))))
A(0,A(0,A(0,A(0,A(0,A(1,0))))))
A(0,A(0,A(0,A(0,A(0,A(0,1))))))
A(0,A(0,A(0,A(0,A(0,2)))))
A(0,A(0,A(0,A(0,3))))
A(0,A(0,A(0,4)))
A(0,A(0,5))
A(0,6)

Value: 7
```

Name the program: `IterativeAckXX.java`, where XX are your initials.

4 (**). Write a Java program to implement the family relations problem from Prof. Fruehr. The assignment is on a PDF named `RelationsProgram` and the text files to test are provided. Make the following modifications to the assignment:

- Do not create a GUI-write a console application.
- The program builds the tree, and outputs the tree either row-by-row with the top of the tree at the top or the left of the screen (on its side). The user will then enter two names from the keyboard. Check to make sure they are both unique. For each query, print a single line indicating the

relationship of Name1 to Name2, from the point of view of Name1. Use the following definitions for output- mandatory items are delimited with (and), optional items are delimited with [and], options are separated by |. Elements which may require repetition (1 to many) are followed by *.

- For a sibling relationship i.e. sister, output a sentence of the following form:

  ```
  Name1 is the (sister) of Name2
  ```

- If the relationship is some kind of cousin, output a sentence which includes the degree of cousinship i.e. 1st, 2nd, 3rd etc. followed by the word cousins, then the number of times removed (1-time removed, 2-times removed, 3-times removed, etc.).

  ```
  Name1 and Name2 are (1st|2nd|3rd|...) cousins [(1-time|2-times|3-times|...)
  removed]
  ```

- If the relationship is aunt or niece, the output may require one or more instances of the word great.

  ```
  Name1 is the [great ]*(aunt|niece) of Name2
  ```

- Otherwise the relationship will be daughter or mother. Relationships which are two generations apart will require the use of the word grand before the relationship. Relationships which are more than two generations apart will require the use of one or more instances of the word great before the word grand.

  ```
  Name1 is the [[great ]*grand](daughter|mother) of Name2
  ```

- If it is not possible to describe the relationship of Name1 to Name2 under the above limitations, then print "No relation".

Let the user input the file name from the keyboard. Use appropriate spacing for the output of the tree. Output should look similar to below.

Sample Run:

```
Enter the file name: relations.txt

The family tree:

                                     Amarantha

                Bethesda                Callipygia                 Depilitoria

             Edna            Flegma                 Ganache              Hildegarde

                    Indicia   Jaundice   Kalahari       Ludmilla    Miasma
```

```
Enter two names (Type Q to end):  Amarantha Bethesda

Amarantha is the mother of Bethesda

Enter two names (Type Q to end):  Ganache Hildegarde

Ganache and Hildegarde are 1st cousins

Enter two names (Type Q to end):  q
```

Name the program: `FamilyTreeXX`, where `XX` are your initials.