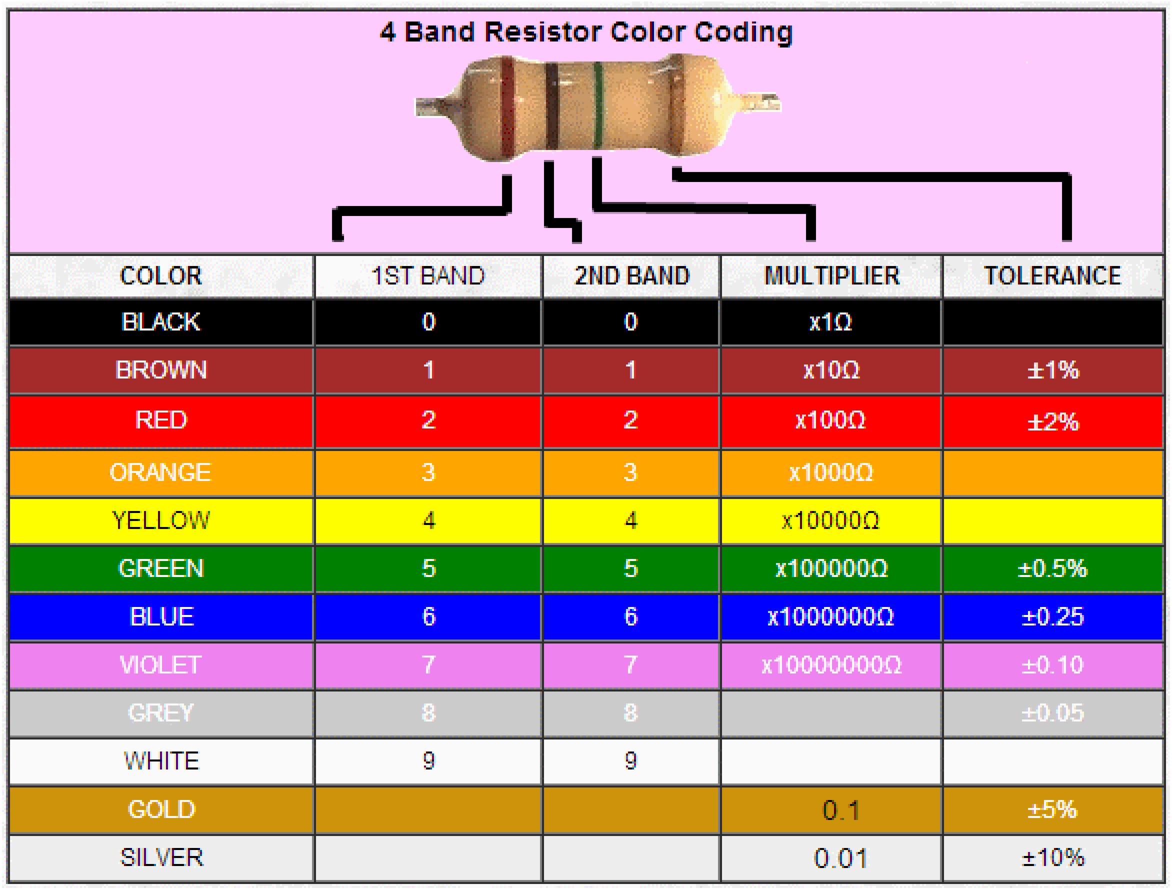
ITSE 2417- Fall 2016
Program Set #3

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

1. Implement Programming Exercise #1- Chapter 11 (Liang) p. 445. Do not worry about the UML diagrams. Class `GeometricObject` will be provided by the instructor. Place all classes into one file. Output should be user friendly.

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

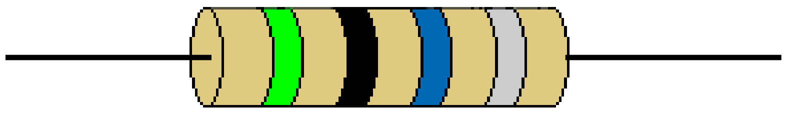
2. Resistors are electrical components, which reduce the amount of voltage flowing through a circuit. This reduction is measured in Ohms. On each resistor there are four color bands, which represent how much resistance the resistor can produce:



4 Band Resistor Color Coding

COLOR	1ST BAND	2ND BAND	MULTIPLIER	TOLERANCE
BLACK	0	0	x1Ω	
BROWN	1	1	x10Ω	±1%
RED	2	2	x100Ω	±2%
ORANGE	3	3	x1000Ω	
YELLOW	4	4	x10000Ω	
GREEN	5	5	x100000Ω	±0.5%
BLUE	6	6	x1000000Ω	±0.25
VIOLET	7	7	x10000000Ω	±0.10
GREY	8	8		±0.05
WHITE	9	9		
GOLD			0.1	±5%
SILVER			0.01	±10%

For, example with the following resistor:



The first two bands represent digits, the third is a multiplier, and the fourth is the tolerance. The resistor given above (with green, black, blue and silver bands) is a `50,000,000` Ohm resistor with a `±10%` tolerance. Tolerance is simply a measure of the accuracy of the rating of the color bands. If there are only three bands, then the tolerance is `±20%`.

Write a Java program that represents a resistor as a class. Provide a single constructor. The class should provide public getter and setter methods as well. Create an additional method that returns the description of the color bands for resistance and tolerance. Output should be the color bands used on the resistor and the calculated resistance and tolerance for the resistor. Write a test program to test your class. Place all classes into one file. Output should be user friendly.

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

3. Write a Java program that ranks weightlifters and outputs a final scoreboard in an Olympic Weightlifting category. A category consists of a single phase which is made of two parts: the Snatch and the Clean and Jerk. Each lifter has three attempts in both the Snatch and Clean and Jerk. These attempts should be with different weights unless repeating after a no lift. The highest score for each lift (not including a no lift) is the one that gets used as the official value for the category. When a lifter fails to complete one good lift (using all three attempts) in either the Snatch or the Clean and Jerk, this is considered a "Bomb Out " and the lifter receives a zero result in total. Once the highest value has been collected for each lift, the results in both parts are combined to produce a total result for each competitor, which determines their ranking (total). In case of a tie, factors to decide the ranking of athletes in total (first three tiebreaks) are:

- best result – highest first; if identical, then:
- bodyweight – lowest first; if identical, then:
- best Clean and Jerk result – lowest first;

Assume no other ties otherwise. The top three total lifters would win the medals. Input will be from a data file that looks like the following:

```
MEN'S 69KG
6
FINN 68.59 145 -151 -151 175 181 -184
STOVER 68.92 156 160 162 188 190 -198
JOHNSON 68.60 -142 142 -147 175 -182
HOUSE 68.79 130 -136 136 -160 -160 160
ROWE 68.68 147 -150 -150 -177 -177 -177
PITTS 68.67 -145 -145 145
```

where:

- The first line will contain the name of the event
- The second line will contain a single integer `m` that indicates the number of lifters in the event (max 20)
- Each of the following `m` lines will contain:
    - The lifter's last name followed by a space (max 15)
    - The weight in kilograms of the lifter followed by a space
    - Up to six lifts each separated by a space. The first three represent the three Snatch lifts the last three the Clean and Jerk lifts. The dash in front of the value denotes a no lift; otherwise it is a valid lift. No numbers (blanks) denote not completing a lift at all (not the same as a no lift). Not completing an entire category of lifts earns a `DNF` (did not finish).

Do not assume the lifters in the file are in any order. Output the scoreboard and rank the lifters in the particular event. For each event output:

- Output the name of the name of the event
- Print the headers and the lifters results in the order where:
    - `RANK-` order is based on the total calculated result. Note a lifter who Bombed Out (three no lifts) or has a `DNF` are not ranked and place at the end of the ranking (in that order).
    - `NAME`
    - The three Snatch scores (`S1, S2, S3`), where a – denotes a no lift
    - `BEST-` Snatch score
    - The three Clean and Jerk scores (`CJ1, CJ2, CJ3`), where a – denotes a no lift
    - `BEST-` Clean and Jerk score
    - `TOTAL-` the sum of the best two events. Output a two dashes (`--`) for zero or missing scores and a `DNF` for not finishing- missing an entire category of lifts.

```
MEN'S 69KG RESULTS
RANK NAME   WEIGHT   S1   S2   S3  BEST   CJ1  CJ2  CJ3  BEST  TOTAL
--------------------------------------------------------------------
1   STOVER  68.92   156  160  162   162   188  190 -198   190    352
2   FINN    68.59   145 -151 -151   145   175  181 -184   181    326
3   JOHNSON 68.60  -142  142 -147   142   175 -182   --   175    317
4   HOUSE   68.79   130 -136  136   136  -160 -160  160   160    296
    ROWE    68.68   147 -150 -150   147  -177 -177 -177    --     --
    PITTS   68.67  -145 -145  145   145    --   --   --    --    DNF
```

In the scoreboard above, `ROWE` bombed out since he failed to clear the 177 Clean and Jerk weight; thus, earned a zero or dash total score. `JOHNSON` did not complete the final lift, however, a valid lift was earned in the Clean and Jerk category. Thus a total could be calculated. A `DNF` was awarded to `PITTS` since he did not complete the Clean and Jerk part of the category.

Format the scoreboard with appropriate spacing so the scoreboard is readable. Let the user input the filename from the keyboard. Use any sort mentioned in the textbook.

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

138.905
+2
+3
La
57
2-8-18-18-9-2
rry
12.011
C
6
2-4
oleman

4 (**). On the television show *Breaking Bad*, the opening credits for each episode showed chemical abbreviations in each actor's name. Write a Java program to 'Breaking Bad'-ify your name. For example, if someone's name was "`Breaking Bad`", you would output "`B_reaking B_ad`", with an underscore following each chemical abbreviation and the element name Boron. The "best" element to show is the one that occurs first alphabetically in each name. Again, the word "`Breaking`" contains seven possible element abbreviations – `B, Re, I, N, K, Br,` and `In`. Only the first in alphabetical order, "`B`", is used. In the word "`Bad`", "`B`" is used instead of "`Ba`" since it is first alphabetically.

The `elements.dat` file contains all 118 chemical elements, one on each line. Input the people's name first and last (checking case) from the keyboard. For each name, output the person's first and last name, as shown below, with the "best" chemical abbreviation inserted in each one, using an underscore _, along with the chemical element information, according to the specifications listed above. If no abbreviation exists for any name, just output the name in its original form and the string "`No chemical elements found`". Enter the file name from the keyboard. Check for valid files, and ask the user to run the program again. Output should look similar to below.

Sample File (partial):

```
1 - H - Hydrogen
2 - He - Helium
3 - Li - Lithium
4 - Be - Beryllium
5 - B – Boron
…
118 - Uuo - Ununoctium
```

Sample Runs:

```
Enter the file name: elements.txt
Enter a name: James Chegwidden

JAm_es C_hegwidden

Elements:

95 - Am – Americium
6- C – Carbon

Run again(Y/N): y


Enter a name: Dae Xaz

Dae Xaz

No chemical elements found

Run again(Y/N): N
```

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