Program Set #2
Total Points: 30

Choose 3 of the 5 problems below for full credit. See Java Grading Guide Sheet for grading/submission information. Partial credit will be given. (10 points each)

1. Implement Programming Exercise #7- Chapter 13 (Liang) pp. 536. Provide all the data/members as mentioned in the text. A UML class diagram is not needed. Place all classes into one file. Output should be user friendly.

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

2. A word is said to be "abecedarian" if the letters in the word appear in alphabetical order. For example, the following are all 6-letter English abecedarian words:

```
abdest, acknow, acorsy, adempt, adipsy, agnosy, befist, behint, beknow,
bijoux, biopsy, cestuy, chintz, deflux, dehors, dehort, deinos, diluvy,
dimpsy
```

Write a Java program and method called `IsAbecedarian` that check to see if a word of any size entered from the keyboard is abecedarian or not. Assume the word entered is a legal word. Output should be user friendly.

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

3. Write a Java program that will allow a person to play the game of "Skunk" against three computer players. Skunk is a dice game that uses two 6-sided dice. Use random numbers to simulate the dice rolls. The goal of the game is to roll the dice for points and the first person who gets 100 points wins. The player determines how many times they wish to roll the dice during their turn provided they don't get "skunked". Once a player gets skunked, their turn is over and may lose some or all their points as a result. There are two ways to get skunked:

- One of the two dice can roll a one. This is a "single skunk". In this case the person loses all points that were earned during this turn.
- Both dice roll a one. This is a "double skunk". In this case the person loses all their points and return to zero points.

The player can end their turn at any time after they have made at least one roll. A roll consists of rolling both dice. If a "one" does not appear, the value on both dice are added to the players score. For example, assume player 1 currently has 27 points. The following table will show how the player's turn went:

Comments	Dice	Roll	Score
Start of turn			27
Roll 1	3	4	34
Roll 2	5	4	43
Roll 3	2	2	47
Roll 4	6	3	56
End of turn			56

Player 1 ends their turn with a score of 56. Assume player 2 currently has 52 points. The following table will show how the player's turn went:

Comments	Dice	Roll	Score
Start of turn			52
Roll 1	2	5	59
Roll 2	6	1	**Single Skunk**
End of turn			52

Player 2 ends their turn with a score of 52. They lost all points earned this turn because of the single skunk. Assume player 3 currently has 70 points. The following table will show how the player's turn went:

Comments	Dice	Roll	Score
Start of turn			70
Roll 1	5	4	79
Roll 2	2	3	84
Roll 3	1	1	**Double Skunk**
End turn			0

Player 3 ends their turn with a score of 0, because of the double skunk. The program must have three computer players to play against.

- The first computer player always ends its turn after 3 rolls (assuming the turn is not ended earlier due to a skunk).
- The second computer players always ends its turn after it has earned at least 20 points (assuming the turn is not ended earlier due to a skunk).
- The third computer player always tries to earn 100 points. Its turn is ended only when it gets a skunk.

The program must show the points of each of the computer players. It must also describe what happened during their last turn. The user player always goes first. When their turn is over the three computer players get turns (one turn per computer player). When a player wins, output who the winner of the game was. Finally, ask the user if he/she wishes to play the game again.

Implementation:

This program will be less structured than previous assignments. Some classes that will be needed are specified below and additional classes can be used. It is up to the student to figure out what the responsibilities (methods) of each class will be. The following class must be used:

- `Skunk` (object) - to represent the entire game. Creating a `Skunk` object should start the game.

Also, no object should directly examine or alter the variables of any other object; all access should be done by talking to (calling methods of) the other object. The student should figure out what the various classes and objects should be and what their instance variables and their methods should be. There is no single "correct" design; any reasonable design will do. The better the design, the easier the programming will be. Use the objects defined to play a game of Skunk. Place all classes into one file. Output should be user friendly.

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

4. Write a Java program that generates portmanteaus. A portmanteau is two words that can be merged together to form a new word. Use the website below to give you some idea on how to do this.

- https://www.namerobot.com/namerobot/name-generators/name-factory/merger.html

Let the user enter two words from the keyboard. Convert all input into uppercase. Output the merged word in uppercase letters. Output should be user friendly.

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

5. The Library of Congress Number (LCN) system are used by libraries to place books on shelves. Every book has a unique LCN. Each LCN has three parts: call letters, a call number, and cutters. The call letters are two uppercase letters. The call number is a number between 1and 10,000, inclusive, possibly including a decimal part with at most 3 digits. The cutters are a letter followed by 0 to 9 digits. Books are placed in order of their call letters (AA precedes AB, etc.); books with the same call letters appear in increasing value of their call numbers; books with the same call letters and call number are sorted by the letter of their cutters followed by the value of the number that follows the letter. For example, `DE23.12X73` appears before `DE23.12X123` because 73 is numerically less than 123. Write a Java program that given a set of LCN place them in the correct sequence. Input from a data file will be four sets of three LCNs. Read each LCN as a single string. Print each set of three LCNs in the correct sequence. On the final line, print the correct sequence for all 12 LCNs. Let the user input the file name from the keyboard. Refer to the sample output below.

Sample File

```
AB3D4 ZZ52A3.95 EF9.12X
DD555A123 BB9A3 XX123H
```

```
BB515A5 BB9A21 BB9A123
BB9.12X3 BB9.12A543 BB9.12A98
```

Sample Run:

```
Enter the file name: books.txt

Set 1: AB3D4 EF9.12X ZZ52A3.95
Set 2: BB9A3 DD555A123 XX123H
Set 3: BB9A21 BB9A123 BB515A5
Set 4: BB9.12A98 BB9.12A543 BB9.12X3

All 12 LCNs in order:

AB3D4 BB9A3 BB9A21 BB9A123 BB9.12A98 BB9.12A543 BB9.12X3 BB515A5 DD555A123 EF9.12X
XX123H ZZ52A3.95
```

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