



# Problems for Coding Practice (not for grading)

Each of the following from 1 to 6 is a Java program you may write to practice things you learned from week 1 to 4 of this course. From 7 to 9, each is based on one or some of the programs of 1 to 6 and re-structure their code (e.g., using multiple methods, or multiple classes in the same file, or multiple classes each in its own file.) Sample solution code will be posted at the end of each problem as soon as it is available. Note that none of these programs implies it is going to appear in the midterm exam.

You may work with someone as a group to solve one or some programs below. No submission is needed.

1. (**P1.java**) Ask and let user enter three names, one a time, like 'John Smith', 'John Peter Doe', and 'Jane K. Smith'. Then the program checks and displays names with same first name, which should be case insensitive. In this case, they should be '**John**Smith' and '**John** Peter Doe'. (Here is a sample solution for your reference: [P1.java](https://usflearn.instructure.com/courses/1340430/files/79136900/download?wrap=1)  
(<https://usflearn.instructure.com/courses/1340430/files/79136900/download?wrap=1>).\_   
(<https://usflearn.instructure.com/courses/1340430/files/79136900/download?wrap=1>).\_)
2. (**P2.java**) Modify P1.java such that it displays names with same last name, which should be case insensitive. (Here is a sample solution for your reference: [P2.java](https://usflearn.instructure.com/courses/1340430/files/79136996/download?wrap=1)  
(<https://usflearn.instructure.com/courses/1340430/files/79136996/download?wrap=1>).\_   
(<https://usflearn.instructure.com/courses/1340430/files/79136996/download?wrap=1>).\_)
3. (**P3.java**) Generate and display 20 random integers from numbers in three different ranges: [-15, 3], [20, 55], and [101, 111]. These numbers may be drawn

all from one of the three ranges, or any two, or one of each range. That is, which one or two or all three ranges to use is also in a random manner. When a number is generated, display it with an count index, which begins with 1 and ends with 20. In addition, if the number is a multiple of 3 (e.g., -6, 3, 54, 111, etc.), not including zero, print 'v' with the number like (14) 54 v, meaning 54 is the 54th random integer generated and it is a multiple of 3 so 'v' is attached. Finally, print the total count of numbers that is marked with 'v'.

4. (**P4.java**) Let user enter four words as the input arguments of the main() method of this program followed by the 5th argument, which must be either 'az' (meaning a-to-z or ascending) or 'za' (meaning z-to-a or descending). The program prints these three words, each on its own line, based on their length in the order specified by the 5th argument. For instance, if the five arguments entered at the command line are Lime, Banana, Strawberry, Pear, and za, then, the output would be as shown below. (Because Pear and Lime have equal length, they can be printed in either order.)

```
Strawberry
Banana
Pear
Lime
```

5. (**P5.java**) Same as P4.java except that, instead of four words but four floating-point numbers like -22.9, 0, 77.051, and 4.88 are entered. After they are printed in the order as specified by the 5th argument, this program also prints four more data about the input numbers: MAX, MIN, SUM, and AVG. Use the above four inputs, the last four lines of the output should be

```
MAX = 77.051
MIN = -22.9
SUM = 59.031
AVG = 14.75775
```

6. (**P6.java**) It generates a random integer between 1~100 without printing it out, then, it let user enter one number a time between 1 and 100 to guess it. Each time a number is entered, the program tells user 'higher' (if the one entered is smaller than the one generated) or 'lower' (if the one entered is larger than the

one generated) or 'you get it' (when they match). When matched, the program also prints the total number of guesses made. This process is repeated until user enter 'x' instead of a number. In this case, the program prints the *minimum* of total number of guesses of all games played since the program execution begins.

7. Unless required, the above programs, P1.java to P6.java, can be simply solved by implementing everything needed in just the main() method of each program. However, it would be a good practice to create a new version of each or some, e.g., P1\_1.java or P5\_1.java, then, move statements from main() to a **new method** (or more methods, if needed) in the same class and let main() call the method(s) to complete the job. With or without input arguments, depending on your design and need of each such method. This is exactly how your Assignment4.java is structured, where the main() calls triangles\_0(), triangles\_1(), etc.
8. You may go further by creating P1\_2.java, P2\_2.java, etc., for each or some of the above program to use a second class and implement a method to complete the same job. You would save the classes in the same .java file until you try next problem. The programs of twoClasses.java, threeClasses.java, and calculation.java that we discussed in class provide good examples of how multi-class approach is applied to solve a problem.
9. If time allowed, try one or some of the programs finished above with two classes and save each class in its own file in same or different packages. You may use NetBeans or simply JDK to practice this type of works--*re-structuring a Java program*.