# C212 Lab 10

Intro to Software Systems

## Instructions:

- Review the requirements given below and complete your work. Please submit all files through Canvas (including all input and output files).
- Download the Lab10.zip file from Canvas.
- Submit a zip file that contains your finished code to Canvas by Wednesday, April 13.

## Lab: Input / Output and Exception Handling

## Part 1: I/O reader

Given a file of patient records provide a menu-based system to perform the following tasks:

1. Add a patient to the file (We don't consider patients have duplicate names)
2. Delete a patient record
3. Count the total number of sick and recovered patients
4. Find average age for all patients
5. Sort patients by age
6. Sort patients by first name or last name (We don't consider patients have middle name)
7. Shuffle patients
8. Exit

(The menu is text and **not GUI based**)

The patients.txt file would look like this:

**Tom Jordan 10/25/1987 sick**
**Jakayla Shelton 11/11/1977 recover**
**Alisha Mejia 02/25/1969 sick**
**Aaliyah Navarro 06/27/2000 sick**
**Johnny Espinoza 01/02/1960 sick**

Where **order by first name, last name, date of Birth, current status** and are separated by whitespace

The skeleton code for the methods is given below:

```
1.  Public static String isBirthVaild (int day, int month, int year){
    //check if the given birth is valid.
    }
2.  public static void addPatient (String name, String Birth, String fileName)
    throws IOException {
    // Add a new patient record to the file.
    // if given birth is not valid, then patient will not be added into the file.
    // Birth must save in a format of Month/Day/Year, in total length of 10, such
    as “02/03/2022”, “11/12/2001”, “01/24/1998”, “12/01/1980” and so on.
    }
3.  public static void deletePatient (String name, String fileName) throws
    IOException {
    // Delete an existing patient record from the file
    // can’t delete if the patient still sick.
    // if there are two patients occur with the same name, and they are both not
    sick, only delete the first one. Otherwise, delete the first patient who is not
    sick.
    }
4.  public static int countPatients(String status, String fileName) throws
    FileNotFoundException {
    // return numbers of sick patients or recovery patients in the file.
    // if client given “” (empty string), then return the total number of patients.
    }
5.  public static void averageAge(String fileName) throws FileNotFoundException {
    // find the average age for all patients in the file.
    }

6.  public static void sortPatientsByAge(String fileName)throws
    FileNotFoundException {
    // modify file by sorting patients by age for all patients from young to old
    }
7.  public static void sortPatientsByName(String firstOrLast, String
    fileName)throws FileNotFoundException {
    // modify file by sorting patients by first name or last name for all patients from
a~z
    }
8.  public static void shufflePatients(String fileName)throws FileNotFoundException
    {
    // modify file by shuffle all patients, so they are not in any order
    // Using random in this method is required.
    }
```

Hint:

- Make sure after each modify, you have the same numbers of patients, and nothing is mess up.
- Following the instruction carefully from the skeleton code.
- You should apply while loop for your menu in main.

## Part 2: Exception Exercises

In the zip folder for this week’s lab there is a class entitled *ExceptionExercises.java* that contains two static methods and one main method that simply calls both of the methods. Each of the methods will throw *at least* one exception and cause the program to end early. Your task is to add exception handling into these methods. **Do not edit the main() method, and do not simply fix the methods so that the exceptions are not thrown.** Your job is to handle the exceptions when they’re thrown, not stop them from occurring.

## Problem1()

This method iterates through an integer array and changes the values inside of the array. Your task is to handle any exceptions that are thrown such that:

1. The *for* loop does not end when an exception is thrown.
2. Any time an exception is thrown, your method will:
    a. Print the following statement: "Exception encountered: " + (the message contained in the exception).
    b. Set the value of intArray[i] to 100 (if *i* is within the bounds of the array).

    **Note:** You can get the message of an exception using the .getMessage() method on the exception.

3. The method returns the intArray variable no matter what. (You shouldn't add any return statements or change the current return statement).

**Reminder: You should not be changing any of the code that is already in this method.** Rather, you need to add code to handle any exceptions that are thrown.

## Problem2()

This method simply returns an ArrayList of Integers. Your task is the handle exceptions in this method such that:

1. The for loop always runs to completion and does not exit prematurely.
2. In the case of a NullPointerException, the method will:
    a. Print the following statement:
        i. "Exception encountered: " + (the message contained in the exception).
    b. Do something to prevent this exception from occurring in future iterations of the for loop.
    c. Still add *i* into the ArrayList.
3. In the case of an ArithmeticException, the method will:
    a. Print the following statement:
        i. "Exception encountered: " + (the message contained in the exception) + " at index " + (the index that the exception occurred at).

**Note:** You can get the message of an exception using the .getMessage() method on the exception.

**Reminder: You should not be changing any of the code that is already in this method.** Rather, you need to add code to handle any exceptions that are thrown.

Keep in mind that this method will throw multiple exceptions and you will need to handle each type of exception differently. **If your code is correct, the return value of the method will be an ArrayList containing [0, 1, 2, 1, 4, 1, 6, 1, 8, 1]** (And the method will print some stuff as well).

**Hints for Part 2:**

1. Use try/catch statements to handle exceptions. We will link some information on try/catch statements below
    a. Exception Handling in Java | Java Exceptions - javatpoint
    b. Java try-catch - javatpoint

2. Remember that a single try statement may have multiple catch statements attached to it. This information just might come in handy on problem 2, but who's to say? (wink wink)

3. If you're having issues in which your for loops are ending whenever one exception is thrown, consider the placement of your try/catch statement and whether any other placements of the try/catch could continue the loop while still handling the thrown exceptions correctly.
    a. If you're still stuck, try googling "java exception handling inside loop" and see what the internet has to say.