COMP250: Project 1 – Java Recap

**DUE DATE: September 18th. Standard late policy applies (-10% per day late, no later than end of 21st).**

# Summary

In this project, you will be demonstrating that you understand the fundamental concepts from COMP151 and COMP152. Refer to the review slides or ask me for help if needed. You will be building an object-oriented program that uses 2 classes to support the main method. In theory, none of this should be new material for you, so I’m fine with having a heavier hand in helping for this project to ensure we’re all on the same page.

# Specifications

You will create a program that has a main method (with a few methods to support it), as well as 2 classes which use inheritance to represent objects. Below is a list of what you need to create:

**Side Classes**  
**Person,** which should contain the following:  
 \* **name**, which should be a **String** field.  
 \* **age**, which should be an **int** field.  
 \* a **constructor** methodthat creates a Person when given a name and age.  
**Patient**, which should inherit from **Person** and contain the following:  
 \* **doctor**, which should be a **String** field.  
 \* a **constructor** method that creates a Patient when given a name, age, and doctor.  
 \* a **toString()** method that returns the patient’s name, age, and doctor as a **String**.  
  
**Main Class**  
The **main** class is where your main method and other helper methods will go. Refer to the step-by-step guide for more detail. In short, you will create Patient objects out of the information in **patients.txt**. After, you will run a loop that allows the user to choose from 4 options.

# Step 1: Creating the Person, and Patient classes (30 points)

Begin by creating **Person.java** and declare the **Person** class in it. Next, add the 2 fields (**name** and **age**). Lastly, create the constructor which accepts a name and age. This constructor should set the value of **name** and **age** to be the values passed in.

Next, create **Patient.java** and have it inherit from the Person class (hint: use **extends**). Next, add the **doctor** field. Now you can create the constructor for Patient**.** Because it inherits from Person, the constructor should accept a name, age, and doctor. You also need to initialize the parent constructor (hint: use **super** and send it **name** and **age**).

Lastly, create the **toString()** method in **Patient**. This should **return** a String representation of their name and age, followed by the patient’s doctor’s name. For example, it should return something like **“Benjamin Thomas, age 13. Treated by Dr. Leslie Park.”** if that’s the information for that patient.

# Step 2: Starting the Main Method (25 points)

Now you should have the **Person** and **Patient** classes set up. I suggest making a few Patient objects to re-familiarize yourself with using objects and verify they work.

The next step is to create **Main.java**, which should contain your **main method**. The first thing you should do in it is create an empty **ArrayList** to hold the patients you’ll create in step 3. Also, initialize 2 **Scanner** objects to represent **patients.txt** and keyboard input. In this instance, initializing means to create the Scanners but don’t give them values yet since that will be done inside a try-catch next.

Once that’s done, you should set up the open **patients.txt** (provided in this download) and set up the keyboard input. You can create Scanner objects out of a file with **new Scanner(new File(“patients.txt”)),** though you may need to change the exact string used for the filename depending on how your IntelliJ project is structured. Note that you will need to surround this in a try-catch that handles the possible **FileNotFoundException** that Scanner throws.

# Step 3: Processing the Patient Info (20 points)

Now it’s time to process **patients.txt** and create objects. Your **Scanner** object for patients.txt has a useful **hasNextLine()** method that you can use to continue a **while** loop until there’s no more lines. As it progresses through the file, you can store each line in a String variable to manipulate. Each line contains 3 comma-separated values, so you can use **split(“,”)** on each line to get an array of values that you can use to create **Patient** objects. Each of these new patients should be appended to the ArrayList made in step 2.

**NOTE:** You’ll need to use **Integer.parseInt()** to convert the patient’s ages from text to integers. You should be putting the object creation into a try-catch to ensure the program doesn’t crash if there’s an invalid age in a line. You can skip entries with invalid ages by using **break** (I intentionally included an invalid one to ensure your try-catch works).

# Step 4: Creating the Main User Input Loop (20 points)

The final step is to add an infinite loop (**while(true)**) that asks the user to choose from 4 options. I suggest using a switch expression to decide which option to do based on the user’s input, but a standard if-else is fine. The 4 options should be:

**A – List All Patients**. This should call a method that loops through the list of patients and prints their information using their **toString** methods.

**B – List All Patients (under age 40)**. This should call a method that loops through the list of patients and only prints out their information if their age is under 40. I suggest designing a single method to accept a lower and upper bound for age and sending in 0 and 40 for the bounds here.

**C – List All Patients (age 40+)**. This should use the same method as option B, but with different bounds (41 and 100). Make sure you design it so that 40 is not excluded (for example, don’t accidentally have option B print 1-39 and C print 41+).

**D – Exit.** This should simply **break** out of the loop or use **System.exit()** to end execution of the program.

# Submission & Grading

The blackboard download for the project includes the grading scheme I’ll be using, so that you can see exactly what is worth what points. The remaining 5 points missing from above is a general “does it all work?” which allows me to still give credit for good code even if there’s a separate issue with getting it to run correctly.

You should submit all relevant files (preferably in a .zip format with name like **FirstnameLastinitial\_Project1,** for example **SeanS\_Project1**) to the project’s page on Blackboard. As noted in the syllabus, all projects must be submitted to Blackboard by the due date (or up to 3 days late for -10% each extra day) to be graded. When you come to demo the project to me, I will be downloading and running the code that was submitted on Blackboard, to ensure that no changes were made to the project after the due date.

You may submit as many times as you want prior to the due date but note that the late penalty will still apply accordingly. For example, if a second submission is a day late, it will still lose 10%. This is to encourage people to finish by the initial due date, and to only utilize the late period for fixes that are worth more than 10%. I’m generally flexible with what counts as a “day” late (12:30am is technically the next day, but I’m not picky) if it’s not a recurring issue.