COS 126 General Computer Science Fall 2021 Programming Exam 2

Download the project zip now. Download the project zip from TigerFile which contains the files you will need for this exam. Open it in IntelliJ, the same way as you do for your assignments. *Do not examine any files yet.*

Submission & Grading. As with the COS 126 assignments, you can and should submit your code multiple times, but only the last version will be graded. You are responsible for uploading the correct version of your solution. Your program will be graded on correctness, clarity (including comments), design, and reasonable efficiency. You will lose a substantial number of points if your program does not compile. The total number of points possible on this exam is 35.

Before the exam. Read this page of instructions before the exam begins, but do not start (by reading the next page) until instructed. You have fifty (50) minutes.

Resources. You may only use: your textbook, the booksite, your notes, your code from programming assignments, the code on the COS 126 course website, materials from lectures, labs and precepts, and existing Ed posts.

No communication is permitted (e.g., talking, texting, Ed, etc.) during the exam. We will be sitting outside the exam room if you have a question.

No electronic devices permitted (e.g., phones, headphones, calculators, etc.) during the exam, except with course staff. You may use your laptops and scratch paper only.

Do not discuss later. Due to multiple exam times, and various conflicts, some of your peers will take the exam at a different time. Do not discuss the exam contents with anyone (not even other students that you know already took the exam!) until after the graded exams are returned.

Do not remove this exam from the exam room. Return it to course staff at the end of the exam.

Print your name and NetID and Exam Room on this page, now:

Name: _____ NetID: _____ Exam Room: _____

Honor Code pledge. You must electronically sign the honor code by retyping the pledge below and then your name in the file `honor-code.txt`, and upload it to TigerFile as part of your solution. Please do this now.

"I pledge my honor that I have not violated the Honor Code during this examination."

Do not examine or start the exam until instructed to do so.

For this exam you implement a roommate matching program. In Part 1, you will implement the `Person` class, which keeps track of a person's personal preferences (e.g. wake up time, music preferences, etc.). In Part 2, you will implement the `Roomie` class, which makes roommate matches based on people's preferences.

We provide starter code that compiles, runs, and includes all method declarations. You will need to define instance variables and modify the body of various methods (including replacing the dummy return values we include so that the starter code compiles) by following the instructions below. However, DO NOT modify any method signatures.

Part 1 (32 points). Write an immutable abstract data type `Person.java` with the following API. Each `Person` object has the following five attributes: a name, the time they wake up, the type of music they like, their personality type (e.g. `"ENFP"`), and whether or not they have a roommate.

```
public class Person
  public Person(String name, int wake,              Creates a Person with the given initial values
                String music, String personality)     / set initial roommate availability to true
  public String toString()                          Return a String representation of a Person
  public void setStatus(boolean available)          Sets the Person's roommate availability
  public boolean isAvailable()                      Returns Person's roommate availability status
  public static boolean musicAffinity(String m1,    Returns true if the two music Strings are
                                      String m2)      equal or if one is equal to "All!"
  public static int personalityAffinity(String p1,  Returns the number of characters the two
                                        String p2)    personality strings have in common
  public int affinity(Person other)                 Compute the overall affinity between
                                                      this Person and the other Person
  public static void main(String[] args)            Tests all instance methods in this class
```

Here is some more information about the required behavior:

1. **Constructor:** Throws an `IllegalArgumentException` if `wake` is less than `4` or greater than `12`. Assume the values of the `String` arguments are always valid. A Person is initially available (i.e., `true`) as a roommate.

2. **`toString`:** Returns a String representation of the Person of the form

   `<name> <wake> <music> <personality> <availability>`

   where each variable is separated by a space. For example, a person named **Shawn** who wakes up at **9**, likes **`Jazz`**, has the personality type **`INTP`** and **is available** as a roommate would be represented as the `String`:

   `"Shawn 9 Jazz INTP true"`

3. **`musicAffinity`:** Returns `true` if either music genre is the same, or if either or both of them are equal to **`"All!"`**, which indicates a person likes all types of music. For example,

   ```
   musicAffinity("Jazz", "Jazz") // return true, because they are the same
   musicAffinity("Rock", "Jazz") // return false, because they are different
   musicAffinity("Rock", "All!") // return true, because one of them is "All!"
   ```

4. **`personalityAffinity`:** A Myers-Brigg personality type is a 4-character string, where each character can be one of two options: 1st ch: 'E' or 'I', 2nd ch: 'S' or 'N', 3rd ch: 'T' or 'F', 4th ch: 'J' or 'P'. Thus, there are 16 (2^4) unique personality types. To compute the affinity between two personality types, count the number of characters that are equal across each personality string. Assume that the strings are valid personality types. For example,

   ```
   personalityAffinity("ESTJ", "ESTJ") // return 4, because they share all 4 characters
   personalityAffinity("ESTJ", "INFP") // return 0, because no characters are shared
   personalityAffinity("ENTP", "INTJ") // return 2, because they share 2 characters (N, T)
   ```

5. **`affinity`:** The overall affinity measure considers only three components: wake time ($\mathbf{w_i}$), music compatibility ($\mathbf{a_m}$), and personality type ($\mathbf{a_p}$). The formula you need to compute for affinity is:

$$\mathbf{8 - |w_1 - w_2| + a_m + a_p}$$

where w_1 and w_2 represent the respective wake times of two persons, a_m is 2 if their music is compatible and 0 if it is not, and a_p is the affinity of the two personalities types (between 0 and 4). For example, the affinity between the `Person` objects with the strings **`"Angel 5 All! INTP"`** and **`"Casey 5 Funk ESTJ"`** is **11 (= 8 - |5 - 5| + 2 + 1)**.

6. **`main`:** This method serves as the test client and should not only call each instance method directly, but it must also help verify that each method works as prescribed (e.g., by printing results to standard output).

7. **Hint:** See the `String` API for checking `String` equality and accessing individual characters in a `String`.

Part 2 (3 points). Congratulations if you made it this far! This part may be difficult during a timed exam. It is meant to provide extra challenges only for those of you who have sufficient time. You will receive credit for these components only if your previous steps are solved correctly.

The client program `Roomies.java` contains code that reads a list of persons into a `Person[]` array. Assume an even number of persons. We also provide several data files in the `data` directory and their corresponding matches in the `output` directory. You must write three functions:

```
public class Roomies
  public static Person findBestMatch(Person p,          Returns the best available match for
                                     Person[] list)       for a given Person
  public static String makeMatch(Person p1,             Sets the availability of both Persons to
                                 Person p2)               false; returns a String containing the
                                                          matched persons and their affinity
  public static String makeBestMatches(Person[] list)   Makes best in-order matches and returns a
                                                          String containing all the matched
                                                          persons
```

Here is some more information about the required behavior:

1. **`findBestMatch`:** Returns -- out of all *other available* people -- the `Person` with the highest affinity to `Person p`. In the case of ties, assume the first pair encountered in the better match. Return `null` if there's no match available. For example, for the people listed in `data/prefs04.txt`:

   ```
   Angel 5 All! INTP     Briar 8 Rock ESTJ     Casey 5 Funk ESTJ     Devin 11 Funk ENFP
   ```

   Angel's best match among the 3 other (assuming they are all available) persons is Casey with an affinity of 11.

2. **`makeMatch`:** Creates a match between two `Person` objects by setting each of their availabilities to `false`. It then returns a `String` containing representing `Person` objects and their affinity score, using the format `"<p1> and <p2> : <affinity>"`, for example:

   `Angel 5 All! INTP false and Casey 5 Funk ESTJ false : 11`

3. **`makeBestMatches`:** Iterate through a list of people *in order* and make the best available match for each available person, until all people are matched; return a `String` of all matched persons, with each matched pair on a new line. Hint: Call `findBestMatch` and `makeMatch`. For example, executing `Roomies`:

   ```
   > java-introcs Roomies < data/prefs04.txt
   Angel 5 All! INTP false and Casey 5 Funk ESTJ false : 11
   Briar 8 Rock ESTJ false and Devin 11 Funk ENFP false : 6
   ```