CICS-160 Fall 2024 Assignment 2: Abstract Data Types, S and D S.O.L.I.D. principles Due on Sunday October 6 2024.

Learning Goals:

This assignment is designed to give you an opportunity to practice with the design of a couple of classes that work together in order to solve a particular problem, and how to state the specifications of such a solution via Abstract Data Types (ADTs). This is an assignment centered on design, not implementation. Your design should pay close attention to using the S and D principles of the SOLID OOP design approaches (the Single Responsibility principle and the Dependency Inversion Principle)

This assignment will be 100% manually graded, so, different from other assignments this semester, you will not get an autograded score when you submit your work.

Overview

For this project, you will be designing two classes that, together, will implement a system to assign employees to companies. You will present your design by submitting the ADT for these two classes. You do not have to implement the classes you design. If you submit code but no ADTs, your assignment will not be graded and you will not receive credit for it. You can, of course, implement the classes as a way of practicing, but we will not be looking at any code while grading, and we, unfortunately, will not be able to give you feedback on your code at this time. Later this semester we will implement aspects of these two classes, but right now we want to place our attention to the design of the classes, not to their implementation. You can use one of several different ADT styles we have seen in class, but you need to make sure that the required information is included, and included correctly. **Pay attention, using object oriented design principles we have ben seeing in class, to correctly use object encapsulation and abstraction, as well as coming up with a design that follows the Single Responsibility Principle and the Dependency Inversion Principle.**

In our system, we will have candidates and companies. There will be an equal number of each, but we do not know the exact number. That is, if there are four candidates, there will also be four companies. If there are sixteen candidates, there will be sixteen companies, and so forth. Each candidate will rank the companies they want to work with, by assigning them a number.

Here is an example with three candidates and three companies. Just for the sake of more easily talking about them through the example, the candidates are called Alice, Bob, and Charles, and the companies are called Acme, BMT, and CarsRUs. But, in reality, these candidates and companies are represented by numbers: for candidates, 0→ Alice, 1 → Bob, and 2→ Charles. For companies, 0→ Acme, 1→ BMT, and 2→ CarsRUs.

Each candidate lists the companies in order of preference. For this example, Alice prefers Acme best, then BMT, then CarsRUs. Bob's top choice is also Acme (just like Alice), but its second choice is CarsRUs, and its third choice is BMT. Charles ranks the companies as CarsRUs first, then BMT, then Acme. This is represented in the following matrix:

Alice	0 (Acme)	1 (BMT)	2 (CarsRUs)
Bob	0 (Acme)	2 (CarsRUs)	1 (BMT)
Charles	2 (CarsRUs)	1 (BMT)	0 (Acme)

The preference of each of the candidates is represented by a list. For Alice, in this example, that list would be [0, 1, 2]. For Bob, the list would be [0, 2, 1]. For Charles it would be [2, 1, 0].

Now, as an added element, imagine that there is some outside process that assigns candidates to companies. Those assignments are represented as a list of candidate-company pairs, in some arbitrary order. For example, [(2,1), (0,0), (1,2)] represents that Charles (candidate #2) is assigned to BMT (company #1), Alice (candidate #0) is assigned to Acme (company #0), and Bob (candidate #1) is assigned to CarsRUs (company #2). In each pair, the candidate is always listed first, and the company second. Let us call this list a Matching. In our scenario, a matching will always have as many elements as there are candidates.

Your task is to design two classes that, when working together, will create a system that can perform a certain number of tasks for the system described above:

- store the preferences for a candidate.
- Given a Matching and a list of candidate preferences, determine if every candidate is matched with their favorite company (depending on the candidates' preferences, achieving that type of matching might be impossible).
- Given a matching and a list of candidate preferences, determine if every candidate is matched with *some* company.
- Given a matching and a list of candidate preferences, determine if every candidate is matched with a company from among its top N choices. N is number greater than or equal to 1.

- Any operation that involves two or more candidates should be performed via the ListOfCandidates objects, which can then make calls to objects of type Candidate.

Your tasks

Based on the operations described above, submit ADTs for a class called Candidate and a class called ListOfCandidates. Your ADTs should indicate the signature of each of the methods, including both what inputs they will take and what outputs they will return, if any. If a method uses another method (of its class or of its own class), you must document how that other method is used.

Upload three files to gradescope:

- Candidate.pdf, which includes the ADT for class Candidate.
- ListOfCandidates.pdf, which includes the ADT for class ListOfCandidates.
- Optionally, Ii you want, your design can include more than two classes. If it does, you need to upload a PDF with the ADT of those other classes.
- Main.pdf, which explains how each of the operations required of the system is achieved. This document should make explicit mention of the methods class Candidate, class ListOfCandidates, and any other class you have decided to use.