Design

The program is composed of several classes from a bottom up design the classes are composed of a basic application class which holds generic data items of an application such as name, address and etc. further the application contains a next pointer to work as a list to add to the job class. The job class encapsulates an application such that a job can be composed of a list of applicants that have submitted an application to the job. Dynamic binding is used by making the application class an abstract class with a virtual display function. Further the derived classes contain data members of jobs and questions to make a linear linked list in those classes. Basic memeber functions for the data structures (particularly related to a linked list) were made, such as insert, remove, retrieve, addition and etc. Dynamic binding is used when adding data to the particular application. The manager class particularly uses the dynamic binding and run time type identification when instantiating the classes. Both technical and work application clases have many of the same functions and although they should have been pushed up the heirarchy this was not done (although it should have been) due to implementing the operation in a different manner. The technical application class copies a linear linked list using the copy function provided in the questions class. A full copy is made for exploring the differences between Java and C. Functions like copy and remove and others are implemented recursively while others such as insert and display are implemented iteratively due to finding a middle ground between efficiency and programming tasks.

1

How well did the data structures perform for the assigned application?
The data strucutres performed the operations of the assigned application mostly well with regards to efficieny and object oriented design. The design of the classes were by the objects that are instrinsic to the assignment for example, storing the applicants data in two different classes as a heirarchy design was essential to the program. Further dynamic binding was used to downcast to the derived classes to display functions that were not in the base class through casting. In this design most of the classes had very similar linked structures but each differed in some form of design fundamentally. Although the complexity is reduced somewhat using OOP the modularity of the program is decreased.

2

Would a different data structure work better? If so, which one and why.
In the case of the program problem the data structures chosen fit the task well. A alternative data structure that can be used in place of a linear linked list is a hash table since the amount of jobs are known but the number of applicants can be hashed with a key. Each of these has its own advantages with their respective running times. However the linearity of the linear linked list is very inefficient. For example the length of the linked list requires a polynomial time display due to the interconnected linkedlist, this might be reduced due to chaining of the hash table. Since the program does implement a linear linked list with the increasing number of applicants, the program may run slower. In conclusion the data structure that fits the program design well is either a linear list or a hash table.

3

What was efficient about this program design?
The efficiency of the design lies in the object oriented design of the program. The responsibilities are characterized as encapsulation or responsibility of each object. Each class working with data structures do implement the remove, insert and other data structures related functions. Although the code reusage is extensive throughout the program the data structures may not have much in common to form a heirarchy. The relationships as encapsulation served to simplify the code and reduce the complexity in design which contributed towards the efficiency of the design. Further using the tables both reduced the linear searching and increased run time efficiency.

4

What was not efficient about this program design?
The inefficiency of the program lies in the code repeating. For example both the application, job and questions have a next pointer. Further the application heirarchy repeats a lot of code that may be condensed into the base class. In this case all three can be treated as a linear linked list node. Even further a generic linked list can be derived into all three structures so that the heirarchy is reduced even further.

5

What would you do differently?
Given more time, I would modify the heirarchy and push more functionality into the base of the heirarchy (application class) which would reduce the code and help make the design leaner. Further I would also change the hard coded constants for some of the data strcutures to better represent the user input into the design.