Dallin Van Mondfrans
u0717113
CS 2420 – Assignment 2

Assignment 2 Analysis

1. I was unaware until doing this analysis that this assignment is typically done as a pair programming assignment. After having completed the assignment, I can more fully see and appreciate the value that having a partner to work with on this assignment would be. Although I did not find this assignment to be extremely difficult technically, I did have trouble understanding and visualizing how all the classes fit together and were organized. I am sure that if I had had a partner to work with on this assignment, I would have been able to more quickly see the underlying structure. Additionally, I made one small error when writing the equals method in the Book class, which took me several hours to debug and resolve. I believe that working with a partner would have reduced that time significantly.

2. Although Java's Comparable and Comparator interfaces are very similar, they have several key differences.

    a. Comparator has one method that must be implemented, with the following method signature: public int compare(Object lhs, Object rhs). We can see that the compare() method takes two parameters, a left-hand-side object(lhs), and a right-hand-side object(rhs). The user defines how to evaluate which one is "less", and then returns an integer(int) value. If the value is less than 0, then lhs is said to be "less than" rhs. The converse is also true, with a return value of 0 indicating that the two objects are "equal".

    b. Comparable also has one method to be implemented: public int compareTo(Object obj). However, this method takes only one parameter. When this method is called by an instantiated object of the same type as obj, the two are compared as specified and once again, an integer(int) value is returned. If the object on which the method is called, or "this", is "less than" obj, then a negative value is returned. The converse is also true, with a return value of 0 once again indicating that the two objects are "equal".

    Apart from the small differences pointed out above, the real difference between the two is that objects that implement Comparable can be passed directly to the sort() method. This is because the object itself defines how it is to be compared to other objects of the same type. In

contrast, objects that have not implemented the Comparable interface can be sorted by defining another class that implements the Comparator interface, which will specify how two objects of the given type are to be compared. This class is then passed to the sort() method as a second parameter along with the list of objects to be sorted.

Comparable should be used to define the natural ordering of a user-defined class. This is often called the natural ordering of an object. Comparator should be used when two different objects need to be compared. For example, Comparator would be used to compare different object classes(i.e. Circle, Triangle, Square, etc.). Because these are all different objects, we would need to implement Comparator and define how the different classes would compare. An equivalent example of when Comparable would be used, then, is if we needed to compare two objects of the same class such as Circle. This class would define how to compare itself to other Cirlce objects, and thus we could simply pass a list of Cirlce objects to the sort() method to sort our list.

3. I feel that I spent most of my time efficiently on this assignment. I worked through the implementation of methods for this assignment systematically, without jumping ahead and leaving large portions unfinished. However, I left most of the testing until the end, which I think ended up costing me time as I had to do all of the debugging at that point. Additionally, I think that I should have sat down and written up the class hierarchy in simplified form on paper before delving into the more complicated classes like Library and LibraryGeneric. In the future, I will focus on seeking to see the big picture and understand more fully the problem to be solved before trying to implement each individual method.

4. Generic code is extremely important not only for this course, but in programming in general. In the context of this class, however, it is paramount that the data structures and algorithms which we create and design be able to take different types of objects. A data structure is simply a specific way of organizing information. Thus, the type of that information will vary widely, and we want to be able to organize whatever information it is efficiently. In the context of algorithms, the pseudo code and implementation of how to deal with different types of information is identical, so we do not want to restrict a data structure to only one type of information, when it can handle a much wider variety. Ultimately, generic code makes programs more versatile, effective, efficient, and less cost intensive.

5. I estimate that I spent about 12 hours on this assignment in total.