**Computer Programming II Lessons Learned Reflection**

Kyle Petersen

CSC327-1 Computer Programming II

Dr. Bindu George

July 7th, 2024

Inheritance  
 The first lesson taught to us in the programming II course was primarily about the idea of inheritance in object-oriented programming languages such as Java. According to Pankaj (2022), "Inheritance in Java is the method to create a hierarchy between classes by inheriting from other classes." Put plainly, inheritance is the idea that one class (the subclass) can inherit properties or methods from another class (superclass), which allows for extensibility and reduces code duplication. It also helps define a template that all inherited classes must follow, potentially enhancing code readability and maintainability. Pankaj (2022) uses the example of “if Sedan extends Car and Car extends Vehicle, then Sedan is also inherited from the Vehicle class. The Vehicle becomes the superclass of both Car and Sedan.”. Because of this hierarchy established car now has access to properties and methods which are given from both the car and vehicle class in this instance which is powerful in helping us organizing code and reducing duplication.

### Java Swing

The next concept taught to us was how to create graphical user interfaces through the medium of Java Swing. To start out we made a basic user interface using JFrames and JPanels to create the foundation for our applications. From there we learned about additional important UI concepts such as events where code is run when a certain event occurs such as the press of a button.

JavaFX  
 After the basics of Java swing were learned, we went into more depth so we could make more feature rich applications using JavaFX. This included more control with the application including JavaFX events, and most importantly, layouts. Layouts we learned were very important when organizing any kind of elements in a UI application since without them our GUIs would look very unorganized and cluttered. We talked about the importance of picking the correct layout depending on the situation in our discussion form.

### Abstract Classes and Interfaces

After we had gotten our UI chops up to snuff with Java Swing, we than delved deeper into OOP with the idea of abstract classes and interfaces. According to W3 Schools, “Data abstraction is the process of hiding certain details and showing only essential information to the user.” in reference to a Java abstract class we are referring to how we can restrict a class from being instantiated and only allow methods which do not have a body. Meaning that the subclass will be doing most of the work and the abstract class we are inheriting from is essentially just a template. In that same vein we learned about interfaces which also provide a template for classes to inherit from, the main difference being that a class can inherit multiple interfaces but only a single abstract class. Interfaces are also expected to have all attributes and methods as public.

### Recursion

Next up was the idea of recursion which is basically when a method calls itself. When a method calls itself, it can setup a loop which we can stop by creating an appropriate path to return. Whenever a method is called it is also added to something called the “call stack” which is basically just a list of methods which need to be called in order, this is important for us to know when working with recursion because if we never exit the loop we have created, the call stack will become full and create a “stack overflow” error.

### Sorting/Searching

The next programming concept tackled was the idea of sorting and searching a list. There are many sorting algorithms we can use when trying to sort a list in some way, with this in mind it is important that we pick the correct algorithm for our circumstance when we need to sort a list. For example, according to Geek for Geeks (2023) it is best to use a selection sort “When the list is small. As the time complexity of the selection sort is O(N^2) which makes it inefficient for a large list.”, this is due to its time complexity being O(N^2) inefficient which means for each element in a list the amount of time it takes to sort goes up exponentially. In contrast a quick sort should be used for datasets that fit in memory specifically. According to Geeks for Geeks (2023) again, “Quick Sort is an in-place sort (i.e. it doesn’t require any extra storage) so it is appropriate to use it for arrays.” meaning data structures with direct memory access such as arrays are very efficient with the quicksort specifically. We had also touched on searching algorithms which is meant to find a specific element in a list and follows similar rules where we need to find the appropriate algorithm depending on the list in question and our time/memory complexity needs.

### Data Structures (Lists, Sets, Maps, LinkedLists, Stacks, and Queues)

Next a myriad of data structures was introduced to us which we can now use in our programming tool arsenal depending on the situation, primarily stuff found in the Java Collections and Map frameworks. Although there are a ton of data structures available to us, most of these new data structures are similar in that they usually include some way of adding, removing, searching, sorting, and iteration, but are more efficient in certain situations than others depending on the situation. For example, when we are talking about a stack, a scenario that comes to mind that is useful for them is the back button on a web browser where every time we navigate, we add a new URL to the navigation stack. Then when we need to go back, we can just pop the top of the stack in O(1) time making it very efficient. In contrast, using a queue for this purpose would be inefficient, as removing the most recently visited URL would require dequeuing all the elements until we reach the last one, resulting in O(n) time complexity. This same principal of using a specific data structure for efficiency in involved in all the rest of the data structures mentioned above such as trees, maps, and lists.

References

GeeksforGeeks. (2023, November 1). *When to use each sorting algorithm*. <https://www.geeksforgeeks.org/when-to-use-each-sorting-algorithms/>

Pankaj. (2022, August 3). *Inheritance in Java example*. DigitalOcean. <https://www.digitalocean.com/community/tutorials/inheritance-java-example>

*W3schools.com*. W3Schools Online Web Tutorials. (n.d.). <https://www.w3schools.com/java/java_abstract.asp#:~:text=Abstract%20class%3A%20is%20a%20restricted,the%20subclass%20(inherited%20from>).