**Programming 2 Portfolio Project: Lessons Learned**

Andrew Barnes

Colorado State University Global

CSC372-1

Dr. Reginald Haseltine

2/11/2024

**Programming 2 Portfolio Project: Lessons Learned**

My journey throughout the Programming 2 course was a mixture of reinforcement learning, extending my software development foundation, and enhancing my Java vocabulary. I knew about the four basic principles of object-oriented design but got to implement these mechanisms throughout the course. I was also aware of recursion, but similarly, I got some actual practice. Having done a few years of hobbyist programming, I’ve been able to complete my project goals, but this course enabled me to understand more of what I’ve been doing. Coming from Python and a small amount of C++, I was a step ahead and ready for Java. Understanding new semantics, although similar to C++, I became much more comfortable writing Java programs independently. I could better use the internet to find answers to the subject matter I was previously ignorant of. The following paragraphs further dissect my experience with this course.

**Inheritance & Polymorphism**

Inheritance and polymorphism are two of the four basic principles of object-oriented programming. These two principles, in particular, allow code to be effectively organized and reused. Inheritance allows the programmer to create new classes as a subclass of a preexisting class, mitigating redundancy and enabling reusability in the programmatic workflow. A dog, cat, bird, cow, and sheep class could all be a subclass of an animal class. In this light, you wouldn’t have to define all the standard data and operations amongst all the animals. For example, they all make sounds. The sound method of the animal class would be overwritten with each animal subclass instance. The dog’s sound method would be overwritten to produce “Bark!”, while the cat's would be “Meow”. The others would be “Chirp”, “Moo”, and “Baah”.

When the animal class is inherited, polymorphism comes into play as a single interface representing various methods with the same name. Polymorphism can be programmed as compile or runtime. When the programmer chooses what parameters to satisfy a method’s arguments, the method expects those parameter’s data types to be worked out at compile time, and when the user inputs information that decides on the method, that method is executed at runtime (Nishad, n.d.). A real-life scenario would be programmatically deciding whether to use a local SQLite database or a remote server hosting a Postgres database. This would be a compile-time decision unless the programmer creates a form for the user to populate with information for the program to interpret which to use, such as URL, user, and password, instead of just the local file path. That would be a telltail sign for the program that it needs to use the method that will instantiate a Postgres connection. Since both databases use SQL, none of the other methods in a database manager class would have to be specific to one or the other. Both can insert tables and records just the same.

**Java GUIs**

Learning to build GUIs in Java utilizing frame windows and event handling mechanisms was great because how often are console applications used? It’s much more satisfying to start programming a GUI because of the developed understanding of how graphic-based software is created. This module showed me some of the steps needed for a user-friendly interface when developing modern software. I liked Swing because of its Model-View-Controller architecture, where the model is state information per component, the view decides how to display a component on the screen, and the controller is the arbiter for how components react to what the user does. The Swing MVC architecture is a great way to visualize component relationships between the program and the user. Reading through this framework’s documentation was enough to get me going and make some primary forms with a submit button tied to an event handler that ran a method when the button was clicked and could grab all the data from the form.

**JavaFX**

JavaFX is a Java framework for developing GUIs. In week 3, I understood the workflow of using a layout manager. JavaFX and its layout managers play a crucial role in GUI development, at least for me, who currently doesn’t know how to make an interactive graphic, such as a button, spawn to an instructed location on a computer monitor. Think about all the different monitor sizes users could have and have to write an algorithm to consider the user’s screen size as a parameterized input. Sounds daunting. Thankfully, layout managers take care of that complexity. JavaFX has quickly earned its way into becoming a primary framework in my software development toolkit.

**Object Oriented Design**

Module 4 will always be one of my favorite programming weeks ever. I’m always trying to enhance my object-oriented prowess. Learning about interfaces alone had a ripple effect on what I know about computer programming. A goal of mine is to become proficient in C++, but was always confused by headerfiles and their existence in folders. So, including those ‘.h’ files into ‘.cpp’ files was always weird. The fantastic thing about learning a Java interface is that I could correlate that with what I want to hone as my primary programming language, C++. A Java interface is an abstract datatype with empty body methods, allowing the programmer to implement those methods into a Java class (*Java Interface*, n.d.). Interfaces can be used to show how different classes will converse, preventing the classes from being linked to actual implementations. Because of this, a class can implement various interfaces, unlike how extending a superclass is strictly held to one. An example of an interface used in sorting is the Comparable interface. It selects all the classes that implement comparison, allowing the user to override methods like compareTo to sort elements of a given class by their data.

**Recursion**

At first, recursion was incredibly simple, and at its core, it is. Mainly, I had some trouble keeping track of the expanded hierarchy and snapping back to the base case in my head when trying to perform a recursive algorithm on my own. Recursion taught me that iteration isn’t the only way to parse through and perform operations on an array. I found recursive factorials to be the most accessible exhibit to understand, then Fibonacci numbers, and finally merge sort. Going through merge sort with some arbitrary input arrays with pen to paper is extremely helpful. Until now, I haven’t tried to solve any coding ideas I have come up with. Recursion was an eye-opener because it forced me to visualize an unknown sequence of events. To become more proficient and effective in my coding, I understand that anything in the real world can be translated into data and operations. I know that’s a bit out of scope from recursion, but creating a function that calls itself is ingeniously creative, and I find inspiration in that.

**Collections: Sorting & Searching Array Lists**

Oh, what power lies in the Java Collections framework. The inherent focus of array lists, sorting, and searching algorithms of pre-built data structures shows how mighty the collections library can be. I found that sorting and searching can have a relationship for this module. I suppose it’s more of a one-way relationship, but a relationship nonetheless. Searching an array could take an O(n) worst-case scenario if you go from beginning to end, but if you sort the array before searching, you can better optimize the search algorithm (Portiankio, 2023). This is something precious I’ve learned, and will keep this knowledge handy on my toolbelt near the awesomeness of JavaFX. Searching through a sorted array could become like finding a name in a phonebook. You open the book to the middle page. If the name you’re looking for isn’t on the page, it must come before or after where you are. You can then dismiss the half you know the name is not and dive into the half where you see the name must be. You can keep dismissing half and dividing the other half until you’re left with one or two pages. A divide and conquer algorithm is a highly efficient logarithmic time complexity. Either that or, in the best case, the name was on the middle page, and it only took one operation.

**Collections: Hash Maps, Tree Maps, & Linked Lists**

Diving deeper into the collections library are hash maps, tree maps, and linked lists. These are alternate formations of data management with their forms of accessibility.   
For example, hash maps offer quick data retrieval using key/value pairs. An array indexes by integer value, but a hash map can index by other data types. You query the hash map with the key, and the value is returned. Treemaps, on the other hand, are slower because they offer order and have the added ability to iterate. Linked lists are a data structure where each element, called a node, is its object that includes a data part and an address part linked to other nodes in the linked list using pointers and addresses. These will all be handy for me in my programming ventures because different programs can be optimized through different mechanisms. The responsibility of implementation will fall on me, the programmer.

**Stacks, Queues, & Exception Handling**

This last week, I was fortunate enough to gain an understanding of stacks, queues, and exception handling. Stacks manage data using Last In, First Out (LIFO) and are helpful for problems like the Tower of Hanoi. Queues exhibit First In, First Out (FIFO) and more or less act as a line would. Stacks and queues are effective data management mechanisms, but exception handling is highly prioritized in programming because of its robustness. Programs shouldn’t crash because of unexpected inputs. Handling exceptions with a logistic approach can yield astounding results. There could be a variety of exceptions, and creating ways to deal with each of them is much more effective than treating all exceptions the same. It’s not the most exciting, but can make or break a program.

**Conclusion**

Reflecting on my Programming 2 voyage, it’s easy to see I experienced further transformation in my Java coding abilities. Each module reiterated programming concepts, built new fundamental skills, and enhanced my Java vocabulary. I solidified meaningful connections with my transformation from Python and some C++ to a more practiced Java developer. I know where I’m at and where I want to be. This course enabled me to map out future software development endeavors, both personal and academic. Onward to what’s next!

**References**

Nishad, Ankit. (n.d.) *Difference between compile-time and runtime polymorphism in Java*. Enjoy Algorithms. https://www.enjoyalgorithms.com/blog/difference-between-compile-time-and-runtime-polymorphism

Portankio, Volodymyr. (2023). *How to sort and array in Java*. CodeGym. https://codegym.cc/groups/posts/how-to-sort-an-array-in-java

W3Schools. (n.d.) *Java Interface*. https://www.w3schools.com/java/java\_interface.asp