Bryce Dunlap

Final

CS5004

Final Report

My approach to this project was to first use the backround knowledge of linked lists applied in Lab 4. Admittedly this was a weak starting point in terms of how the overall structure of my project would pan out, as I never wrote a GUI application before. However, as time grew on I reshaped my project, starting from just wanting to input a simple csv file that would encrypt and decrypt the messages inside to now what is a program that allows the user to enter whatever message they want, multiple times and encrypt/ decrypt them all at once while also using a singly linked list structure in the backround. So, my singly linked list is fairly simple to understand. I have a “main” interface that would be my ListUse, which holds the function methods I can use on my linked list. In the sub classes that extend ListUse I have a class labeled EmptyList which can be viewed as my the final node in the linked list, as if I were pointing to null (in C). The other sub class which extends ListUse is NotEmptyList, which will signify the node or nodes that will hold the data if stored in the list. Also, important to note that this linked list takes in generic data type, so conveniently I can easily customize a linked list for just about anything fairly easy, (example: I changed it from storing todo tasks (datatype) in a linked list to now holding a message(string datatype). I would say this is the “main” concept in my program in terms of what is asked for within the rubric. However, if you were to ask me personally what my “main” accomplishment within this project is, I’d label the CaesarsCipherGUI.java file as the main operator. This GUI takes in quite a few attributes such as buttons, text field, text box etc. Although not entirely ideal to integrate my singly linked list with the GUI, it was a fun demonstration of applying the knowledge learned. The reality is, a linked list would become quite inefficient for this application, it worked well to be able to add to the front of the linked list each time the user adds to the text but deleting a word is where the issue of inefficiency arose. If you look closely at my code you will see that I am not deleting a word at a specific location within the linked list itself, I am simply re-writing it using a for loop and copying over what the text display has stored. I’d also like to note that I made my casears cipher methods within a sub class of “CryptographyWork” interface in order to leave room for future growth in applying different encryption algorithms.

|  |  |
| --- | --- |
| Concept 1 : Recursion | I used this concept in my linked list, Look in file NotEmptyList.java on line 15.  I demonstrated this concept by repeatedly calling the method getListSize() while pointing towards the tail. This will go down the whole list until it reaches the EmptyList “node”. Another place where recursion can be found is also in NotEmptyList.java on line 49, this is my forEach method. The forEach method came in handy when needing to print each element in the linked list structure, the recursive approach points towards the tail to recursively navigate through the entire linked list. There are more methods in which I use recursion but since they fall under other Concepts, I’d much rather explain how they were there then here, I think this is enough for recursion. |
| Concept 2: Logical Structure and Design, interfaces | My two interfaces are ListUse and CryptographyWork. The purpose of making my CryptographyWork an interface was to have the idea of expanding what type of ciphers I could implement in the future. This could in theory give me the option to implement multiple ciphers that can be used to encrypt and decrypt. |
| Concept 3:  Useful and Logical Abstraction using Generics and Lambda Expressions | Take a look at how my singly linked list is implemented. You will see it applies to generic types, meaning I can make multiple linked lists that have different data types because of this generic design of my linked list signified by the <T>. There are plenty of places where I use lambda expressions here are where you can find a couple: CaesarsCipherGUI.java Lines: 208, 210, 248, 306, etc. I will talk about the one at line 210 here because the others fall under the next concept where I will explain. At line 210, in CaesarsCipherGUI.java you will see use of a lambda expression, what is happening here is since I used my linked list and map to encrypt the String in ever node, I needed to copy it back over to “words” which is apart of the DefaultListModel which lets me display the text, with the help of my forEach ofcourse to iterate though each element. |
| Concept 4: Higher Order Functions Map, Filter, and Fold | Take a look at my ListUse interface, you will see where I define all of these functions, map, filter and fold. To see where it is applied in the subclass look in NotEmptyList.java. To see where I actually use these functions look at CaesarsCipherGUI.java, on line 208, you will see where I use map, this helped me encrypt each node within my linked list in a convenient way. On line 297 you can see where I used filter and it kinda ties in to where I use fold on line 306. Basically I filter the list to look for a specific sequence of characters, creating a new linked list of these filtered words. I then use fold (or I like to see it as reduce) to basically count the amount of occurrences this word (or sequence of characters) is found. I go into why this is important partially in the comments above this method but to sum it up, if you have a sequence of the same letter multiple times or the same word multiple times, it will make caesars cipher considerably more vulnerable due to being able to detect a pattern. |
| Concept 5: Hierarchical Data Representation as an ADT or a Linked List ADT (Whatever makes the most sense for your application) | The linked list ADT is represented through my ListUse<T> interface. This design allows for flexible and type generic approach to managing the hierarchy data structures, specifically my singly linked list. I’ll list specifically, generic data types, encapsulation appropriate (seen on lines 11 and 12 in NotEmptyList), the recursive approach in traversing nodes allows for future addition of a hierarchy of lists if wanted. Also, I mentioned it before but my CryptographyWork interface also allows for future growth of adding additional ciphers. |
| Concept 6: Architectures and Design Patterns MVC Design and a design pattern | My CaesarsCipherGUI demonstrates MVC design. Messagelist which is instantiated on line 24 holds my data in a singly linked list while “words” on line 22 holds the text that is being displayed in my text display, essentially giving me flexibility to implement different data structures if wanted. The viewing component of my program is made up of Swings components, JPanel, Jlabel, JTextField, JButton and JScrollPane which displays the view of what the user can interact with within the GUI. The controlers can be found towards the bottom, these are noted in methods called ActionListeners, these give the functionality to all the buttons and actions performed in the GUI, such as encrypting, decrypting, search button and the user being able to type and add a message as well as being able to click and delete a message. |

And finally, Concept 7: SOLID design principles. (This needed to be outside the columns)

Single-Responsibility principle – My ceasars cipher was subclassed from the interface CryptographyWork is an example of using S in solid design principles, the class CaesarsCipher has one use, and that is to encrypt a string. Another demonstration of this is in my mainGUI, the only purpose of my main is to run the gui, it holds no applicable methods.

Open-Closed Principle – my ListUse interface with its two subclasses is a good example of this, its open for new data types of linked lists to be used but closed for the need to modify the existing code for it to work. Thanks to generics. The same goes for my CryptographyWork interface, its open to extend more ciphers and doesn’t need further modifications to get it to work.

Liskov Substitution Principle – By implementing my ListUse interface with two subclasses “NotEmptyList” and “EmptyList”, I adhere to Liskovs substitution as any method that uses ListUse<T> should be able to use an instance of either subclass without affecting the execution of the program.

Interface Segregation Principle – This one is a bit iffy as my program does rely on the ListUse interface to work. However, this could easily be changed up so that when I encrypt/decrypt my messages it doesn’t use map in my linked list. I just used it so I can efficiently use my higher order functions.

Dependency Inversion Principle – Like stated in the previous principle, I kind of relied on the dependency of my ListUse interface, with the idea of being able to use my higher order functions efficiently.

-Overall my knowledge of the last two principles need some for in depth understanding. However, I put a lot of hard work into this and I believe the instructions weren’t 100% clear for this portion of the final, with that being said I hope there is some leeway here.

**Extensions:** I made a fully operational GUI. I remember Dr. G stating we did not need to make a fully operational program but to hit all concepts, so I am using my case of having a fully operational program as an extension. I also added some fun additions to my GUI that I’d count as an exntention. First, I changed the icon, lol. Secondly, since encrypting these messages in a small data set like this happens almost instantly, I wanted to add a fun GIF as if it were loading during encrypting, so I had to set a timer that would turn off after 1000ms giving the user the thought that the program is actually loading but the reality is I just wanted my GIF in there. Just take a look at my program and you will see the extensions, there are many additional implementations that were not listed on the rubric as a requirement. Hint: look at my readme too.

I understand that my learning is dependent on individual effort and struggle, and I acknowledge that this assignment is a 100% original work and that I received no other assistance other than what is listed here.

Acknowledgements and assistance received:

I did not use generative AI in any form to create this content and the final content was not adapted from generative AI created content.

I did not view content from any one else’s submission including submissions from previous semesters nor am I submitting someone else’s previous work in part or in whole.

I am the only creator for this content. All sections are my work and no one else’s with the exception being any starter content provided by the instructor. If asked to explain any part of this content, I will be able to.

***By putting your name and date here you acknowledge that all of the above is true and you acknowledge that lying on this form is a violation of academic integrity and will result in no credit on this assignment and possible further repercussions as determined by the Khoury Academic Integrity Committee.***

|  |  |
| --- | --- |
| Name: Bryce Dunlap | Date: 04/10/2024 |