Project 1

Donald Turner

10/7/19

1. Problem Statement

For this project, I was tasked to create a class roster to hold a group of students. To go about this, I implemented a bag data type which allows for duplicates of entries and does not have an order to it. However, this bag could not have duplicate student ID’s. To create this bag, I used two different data types being the linked list and a resizable array. This roster also had to be accessible in the fact that the user has to be able to see its contents including the current size of the class, if the class is full or empty, how many students are in each academic level and just general access to the students themselves.

Secondly, I had to create an expression converter in order to turn infix expression into their postfix and prefix equivalents. To go about this, I used a stack which has the method of last in first out. This mean if I put 5 objects into this stack, the first one I can take out was the fifth object put in. To create this stack, I used three different methods which were using a resizable array, a vector, and a linked list. Postfix and prefix expression are hugely helpful when it comes to computations using the computers. They take less time to evaluate than their infix equivalents.

2. Implementation

I will divide the planning portion into two sections as there were effectively two separate programs to be created in this project. I will first focus on the creation of the student roster. This was going to be a relatively big program just because of how many parts it had. I started off by creating a bagInterface so that I would know what methods I would need to add in the two separate implementations. I knew I would have to be able to allow the user to add students and also let the user remove, access, and view class size and academic levels. I created static functions for all of these. After creating this interface, I went on to first create the array implementation. The most important thing in the array implementation was the actual array itself and the fact that it had to be resizable. I had to make sure there was enough room in the computer to be able to store new array elements which meant there had to be a max capacity. Whenever the program would try to extend the array to add more students, it would need to check that its new capacity did not exceed the max capacity. If it does, the program will throw an error rather than give a not enough memory error which would then allow the user to access all of our data. The array would then implement all the other methods seen in the interface by modifying the array as needed and changing the amount of space it had to an extent as to not go over the maximum. The array would also have two other arrays inside of itself containing all of the current ID’s of the students and containing how many people were in each academic level. Since, there could be no duplicate ID’s, every time a new student was to be added, I had to check whether or not the new student’s ID already existed thus simply used a for loop to cycle through the current ID’s and check if it already existed.

To implement the Linkedlist, I did everything the same except for the usage of an array. Instead I created a Node subclass which would contain the student of the data and have a pointer which would direct us to the next Node in the roster. This way, we could effectively just add new nodes as needed and not worry about the rest of the nodes being an issue if there was not enough space as they were completely independent from one another.

After creating the linked list and array, I needed some sort of tester class to ensure my two data type worked. To go about this, I created an infinite loop that would constantly take input from the user until the user told the program to ‘stop’. I used a scanner to read data in and then made sure it matched on the possible commands such as ‘addStudent’, ‘getClassSize’, ‘academicQuanitties’, and more. After the scanner reads the command it makes sure it is valid and if not asks the user to enter a new command. Once a valid command is entered, the program will go about its process of executing it. For example, if ‘addStudent’ was selected, the program would then ask the user for a first and last name, an academic level, and a student ID. It would then create a new student and throw it into the roster. To be able to test this myself, I found a list of 4500 or so names online and threw them into a text file. I then created random students and executed commands until I knew it worked well. On top of this, the program asks the user what kind of implementation they want to be used, whether it be array or linkedList. To go about this, I initialized the roster as a Baginterface as either the linkedlist or array are also bagInterfaces as they implement it. The user would enter array or linkedList effectively and the roster would then be created as the array or linkedlist accordingly. Interfacing is so powerful.

For the second part of the project, the stacks had to be created. Just as I did with the bag, I created a stack interface with the necessary functions I wanted. This interface only had five methods as that is all a stack needs. It had push which would add a new object to the stack, pop which would remove the top object and return it to the user, peek which would return the top entry without removing it, isEmpty which returns whether or not there are entries in the stack, and clear which would remove every single object in the stack.

I began by creating the array version of the stack which was resizable. I had to make sure everytime I resized the array, there was enough room left so I once again create a Max capacity and if the maximum capacity was ever exceeded an error would be thrown. To implement push, I simply threw the new entry at the end of the array. For pop, I simply grabbed the last item out of the array and returned it, then set the last item to null effectively removing it. To create peek, I grabbed the last item in the array and returned it without removing it. To create isEmpty, I had a entry counter running that would add 1 to itself every time a new push command happened and would subtract 1 every time a new pop command was hit. If the entry number was ever 0, I knew the array was empty. For clear, I would pop the stack until isEmpty was true.

To create the linked list, all of the methods described above were done in the same way except that I used a linked list. The list had nodes which had their data and a pointer to the next node in the list. If the user wanted to push an object, a node would be created to hold the objects data, pointed toward the current first node, and then would be labeled first node. To pop a node, I would simply return the data at the first node then set the first node to the second node. To peek, I returned the first nodes data. To see if the linkedlist was empty, I used the same counter method as I did for the array. To clear it, I would pop until the entry number was 0.

To create the vector was the easiest. The vector and all of the methods we needed were already built into the java libraries. All I had to make sure was to know if there was enough space or not left so I used the max entry method once again to check for that. The vector already has implementations to push, pop, peek, clear, and check if it isEmpty or not, so I simply called each of those in the necessary functions to make it work.

After creating the three implementations, a main class was needed to test if the array, linked list and vector were working. Using the power of interfacing once again, I created an arbitrary roster stack and then asked the user to determine which type they wanted to use. Once they selected, I would turn the arbitrary roster into the according style. I then used scanner to ask the user for an expression they wanted to convert. After entering the expression, I had to ensure it was a valid expression. To check validity, you need to make sure open and closed delimiters match each other. So using whichever stack the user selected and going left to right in the expression, I would add the open delimiters to it and whenever a closed delimiter cam up, I would check the top of the stack to make sure the according open delimiter was on top. If they did not match, the expression was invalid and thus would not work. It would throw a custom InvalidExpression error made. After doing this the expression would be then turned into its postfix and prefix equivalents using their respective methods to do so.

3.Issues

The major issues I ran into while programming was mostly unrelated to that of the project itself. They were all simple syntax error or the VsCode compiler I was using was not working. Additionally, after creating the entire portion of the project for infix to prefix and postfix, it suddenly stopped working. It kept telling me it could not access the array, linked list, or vector even though they were all inside the same package. To go about fixing this issue, I had to do a lot of research and consult online forums. It ended up being an issue of capital letters. Apparently, my folder had capital letters and when I told my classes to be inside that package folder, they were having trouble dealing with the capital letters. After changing this, my program worked like a charm. I also was having trouble dealing with improper input from the users. If I asked for a string and a number was given, an error would be thrown and kill the program. To fix this, I used try catch blocks which would catch the errors and tell the user what they did wrong and have them try again. All in all, the errors were simple but took a while to solve as I was unaware of what was causing issues to begin with.

4. Results/Output

Here are some sample results of the program execution starting with the student roster.