Maximum Wilder-Smith
013134095

2400 Project Report

Task 1:

The Task 1 code works by implementing the ADT Bag as an array Bag and as a linked bag. This Bag is used as the data structure to store a collection of Students from the Student class in a Bag called Roster.

Student:

The student class is fairly simple, it allows for the creation of a Student object that has an ID integer, and Strings for last name, first name, and an academic level. There are some predefined Strings for the academic level to help avoid typos when doing comparison's for class level.

task1():

Next there is the task1() method which is where all of the actions needed for the class are performed. Upon running task1(), the user will see the following:

```
Beginning Task 1 part 1....

Choose an option: 1-add student 2-drop student 3-search student 4-get class info 5-exit
```

The "Task 1 part 1" signifies that this is running a resizable array bag. The options for user input are also shown here. To handle improper input, a message will show up if the user enters a number not 1 through 5 telling them to use one of the shown commands. If they enter text, a message will display telling them to enter a number. They main problem to deal with was improper user input. To catch this, there is a try catch that catches a InputMismatchException, which is thrown if the user inputs text instead of a number. If this happens, the program will tell them to instead enter a number and return them to the main menu.

Upon entering the number 1, the user will see the following prompts:

```
Choose an option: 1-add student 2-drop student 3-search student 4-get class info 5-exit
1
Enter name with a comma separating first and last name: First, Last
Enter the student's class level [1-Freshman 2-Sophomore 3-Junior 4-Senior]: 1
Enter student ID: 123456
Added First
```

These prompts ask for a first and last name separated by a comma. This operation is done by splitting the nextLine() with the delimiter ",". If there are less than 2 entries in this array, the program will ask the user to follow the format of "First, Last". After getting the Student's name, the program asks for a class level. If the user inputs anything other than 1 2 3 or 4, the program will save the Student's academic level as "UNKNOWN". This was how user input was checked in this case. Finally the program asks for an ID number for the student. If the user entered valid inputs, then the program will confirm that the student was added by saying "Added" and the first name of the student and returning the user to the main menu.

Upon entering the number 2, the user gets the following prompt:

```
Choose an option: 1-add student 2-drop student 3-search student 4-get class info 5-exit
2
Enter student ID to remove:
123456
Removed student
```

The program will then ask the user for the ID of the student to remove. If the student exists in the bag then the program will say the student was successfully removed as shown above. If there is no student with the given ID then the program will output the following:

```
Choose an option: 1-add student 2-drop student 3-search student 4-get class info 5-exit
2
Enter student ID to remove:
1
Student not found!
```

Upon entering the number 3, the user gets the following prompt:

```
Choose an option: 1-add student 2-drop student 3-search student 4-get class info 5-exit
3
Enter ID to search for:
123456
student Id=123456, first name='First', last name='Last', academic level='Freshman'
```

If a student with the ID exists, the program will display their information using the Student toString() method as shown above. If the ID doesn't exist, the program has the following output:

```
Choose an option: 1-add student 2-drop student 3-search student 4-get class info 5-exit
3
Enter ID to search for:
1
Student not found!
```

Upon entering 4, the User gets the following information:

```
Choose an option: 1-add student 2-drop student 3-search student 4-get class info 5-exit
4
Class size: 1
Class is empty: false
Class is full: false
Freshman: 1
Sophomore: 0
Junior: 0
Senior: 0
```

Which will use the Bag's methods to get the class size, whether the bag is empty or full, and the amount of students in each class. This last one is done using an adapted getFrequencyOf() method that checks for the same class level instead of object equality. The method is passed a Student object whose only relevant data is class level.

Upon entering 5, the program will quit and begin Task 1 part 2, which is the exact same as task 1 except that a linked bag is used to store the data instead of a resizable array bag. This was easily changed by just changing the constructor of the roster to LinkedBag().

Task 2:

Task 2 involves converting infix expressions into postfix and prefix expression by using Stacks to process the characters of the expression. This Task was much more difficult than the first as priority of operations had to be preserved. This required a helper method priorityOfOP() returns a numerical value for the priority of each operator. There is also the checkBalance() helper method. This method uses the provided algorithm in which it has a stack that keeps track of parentheses, and brackets. One problem faced was changing the type of Stack without duplicating postfix and prefix as was done with task 1. This was solved by passing a Java class as a parameter to the task 2 method. Within the methods a new instance of this class is created

and is used as the constructor for the StackInterfaces. Originally an anonymous instance of the desired Stack type was passed into the methods, but two were required for each Stack in prefix. This lead to the use of creating new instances of the class as this only requires only argument. A sample run of the program is shown below:

```
Enter infix expression or type 'q' to exit:
a+b*c+d
Prefix: ++a*bcd
Postfix: abc*+d+
```

Postfix:

Postfix takes in an infix String expression and breaks this into an array of characters that are looped over. There is one Stack that implement StackInterface to hold all the operators and a String for the output. If the active character is an operator, and the priority of this operator is less than the operator at the topic of the stack, that operator is popped and appended to the output String. If the priority is greater than or equal to that of top operator, the operator is pushed onto the stack. If the character is an close parentheses or bracket, the operator stack is pushed and appended to output until an open parentheses is popped. This check was implemented by checking when the priority of the top is the same as the priority of "(". This whole operation is the following switch case:

```
case ')':
case '}':
case ']':
    while(!ops.isEmpty()&&priorityOfOp(ops.peek())!=priorityOfOp('(')) {
        String two = vals.pop();
        vals.push( entry: ""+ops.pop() + vals.pop() + two);
    }
    ops.pop();
    break;
```

If the active character is not an operator then it is immediately popped and appended output. Upon looping through all characters, the stack is popped and appended to output until it is empty.

Prefix:

This method also takes an infix String, but uses two Stacks. One for the operators and one for the sets of output. Similar to the postfix method, this one breaks up the infix String into characters that are iterated through. Because of the two Stacks and the fact this program needs to put the symbols in front of the operands, this portion of the task was more difficult to implement. If the active character is an operand, it is pushed to the character stack. If it is an operator, the priority of the operators are compared. If the active character has a priority less than or equal to the operator at the top of the, then the top operator is popped and appended to top two operands. The top operand needs to be added as the second operand when appended. This is shown below:

```
while(!ops.isEmpty()&&priorityOfOp(c)<=priorityOfOp(ops.peek())) {
    String two=vals.pop();
    vals.push( entry: ""+ops.pop()+vals.pop()+two);
}
ops.push(c);
```

As with postFix, the if an end parentheses of similar character is reached, all operators and operands are processed until the matching opening character is reached. After going through the entire expression, the remaining operators are popped evaluated before pushing them to the operands. One problem occured when the user just inputs a string of characters without any operators. If this happened the program would return the only the last character of the inputs String. To fix this problem a while loop was added that will add any strings left in the operand stack to the output.