Lipisha Chaudhary
Tejas Raval
lc2919@rit.edu,
tr7550@rit.edu

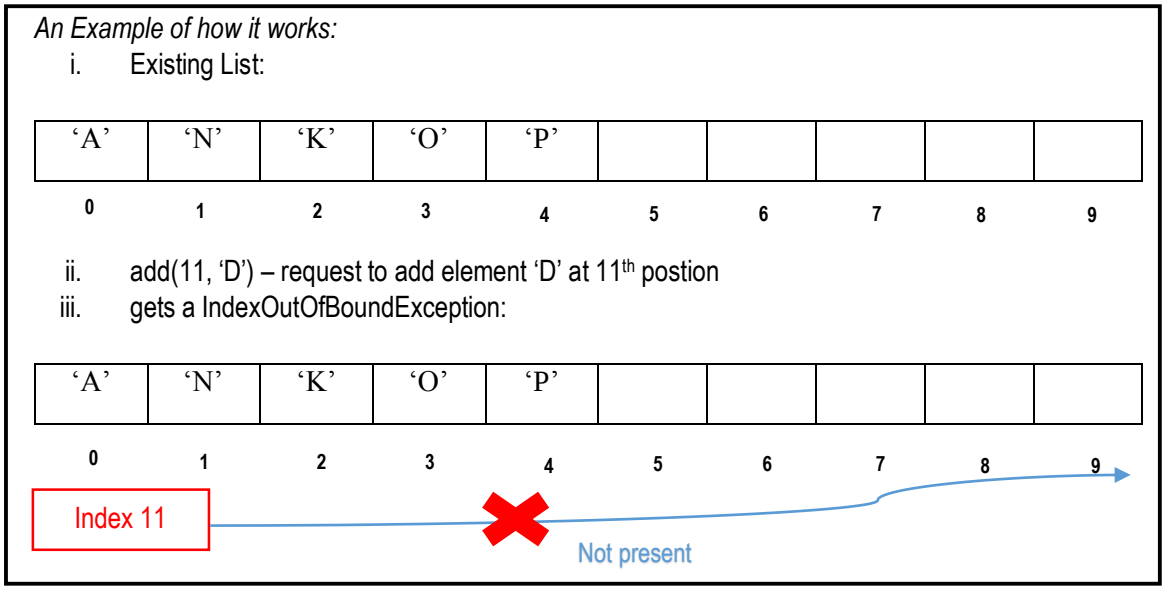
Assignment 7: Exceptions & File Handling

Question 1) Generics – ArrayList & LinkedList - Exceptions

The current assignment problem has asked us to modify the code in such a way that there must be exceptions thrown whenever deviation from the normal procedure is detected. The problem specifically requires us to implement an IndexOutOfBoundException and a customized exception called a NonEvenException, which checks the Integers for even numbers and adds them and checks the Strong for even length and adds them.

Following methods were modified to accommodate the IndexOutOfBoundException:

- **add(index, element):** This add method takes an index as the parameter, this index acts as a pointer which points towards the place where our element is to be inserted. If for example we have a list of capacity 10 which has been only filled up to the 5th index, and if the user while inserting gives (11, *element*), then this will result into an IndexOutOfBoundException since the list doesn't have an index = 11.



- **get(index):** Similar case can be considered for the get method, if a user requests for the program to provide them with the element present at the 11th element the program will throw a IndexOutOfBoundException.
- **remove(index):** For the remove method the program cannot remove an element at an index which doesn't exist in the list indices.
- **set(index):** The set method can be considered alike case as the get method, we cannot explicitly set an element at an index which is not present in the list.

Why was add() method not checked for IndexOutOfBoundException?

While inserting in the lists – ArrayList & LinkedList – a counter will always be present to keep track of the number of actual elements which were inserted in them. The add() method takes help of this counter to check whether the list has reached its maximum capacity, if yes then the list capacity is dynamically increased to accommodate any new element which has to be added. In such case there are very less chances of this resulting into an IndexOutOfBoundException.

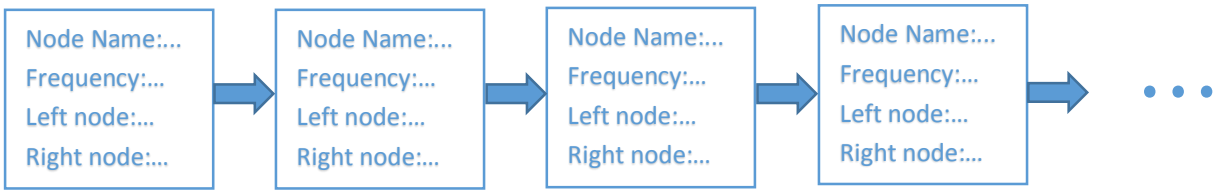
NonEvenException: Since the requirement was specific to the Integer and String type elements, for checking the elements datatype we have use instanceof operator. After checking the data type, we check if the number is even, is yes then insert the element or else throw an NonEvenException, similar with Strings but with the constraint of the strings being of even length. This exemption is a custom made exception, therefor we have created a different class and a common function which checks for these constraints and throws an exception whenever required. Additionally, this is a checked exception, meaning an exception where is caught needs to have remedy steps for it to handle it. In this problem we were asked to rollback all the operations committed before an exception occurred. For this we have based our decisions on the Boolean values which are returned by the add function if even one value is false we call the removeAll() to delete all the elements which were inserted previously.

Question 2) Huffman Code [1][2][3]

This problem asked us to implement a code for Huffman encoding. The main objective of this problem was to give us an idea how file handling works. The algorithm which we referred to was from different sources

Our approach to this problem was pretty straightforward:

- Read the input file.
- Find unique letters and compute their frequencies.
- Sort the letters according to their frequencies in ascending order of occurrence.
- Add the frequencies of the left and right child nodes, and again sort the entire list, and repeat the process.
- Once the Huffman tree is built, we assign the value of '0' & '1' to each left and right edge, respectively,
- Traverse the tree, starting from the child node while recording the values of each edge sequentially – '0' or '1'- reaching up till the root node, reverse the order of the code obtained.
- Compress the output file by appending the codes of each letter as appeared in the input file.



The one distinct point in our code is that we have used the property of code reusability, i.e. the implementation of Huffman code is done using heaps, mainly priority queues, but the requirement clearly stated that we are not supposed to use any java classes from the collections framework, so we thought of using generic linked list from our previous assignment. In the list we are creating a TreeNode object and storing each object along with its information in the linked node. This method proved to advantageous since we were able to easily access the data as a whole entity, and were able to skip using multiple number of data structures for storing.

REFERENCES:

[1] https://www.cs.ucsb.edu/~franklin/20/assigns/prog6files/HuffmanEncoding.htm
[2] https://www.siggraph.org/education/materials/HyperGraph/video/mpeg/mpegfaq/huffman_tutorial.html
[3] https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1126/handouts/220%20Huffman%20Encoding.pdf