Name: Sarvagnya H. Purohit **Registration Number**: 201070056
Date: 02-11-2021 **Branch:** Computer Engineering
Course: DSA **Course Instructor:** Dr. Mahesh Shirole

Lab Assignment 4

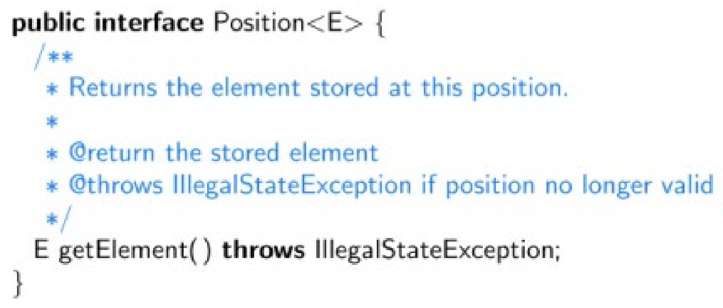
Aim: To implement the positional list using linked list and implement an iterator interface for the same. Use position interface and iterator interface.

Theory:

Position ADT: To provide a general abstraction for the location of an element within a structure, we define a simple position abstract data type. A position interface supports the following single method:

- getElement(): Returns the element stored at this position.

A position acts as a marker or token within a broader positional list. A position p, which is associated with some element e in a list L, does not change, even if the index of e changes in L due to insertions or deletions elsewhere in the list. Nor does position p change if we replace the element e stored at p with another element. The only way in which a position becomes invalid is if that position (and its element) is explicitly removed from the list.



```
public interface Position<E> {
  /**
   * Returns the element stored at this position.
   *
   * @return the stored element
   * @throws IllegalStateException if position no longer valid
   */
  E getElement( ) throws IllegalStateException;
}
```

Positional List ADT: A positional list as a collection of positions, each of which stores an element of generic type. We implement the positional list interface as a doubly linked list in this assignment. To store the locations within the doubly linked list, we use node references. We declare the nested *Node* class in our doubly linked list so as to implement the *Position* interface. As a result, now we treat a *Node* as a *Position*. The Node class is declared as a private static class to maintain encapsulation of the implementation.

The accessor methods provided by the positional list ADT include the following, for a list L:

- first(): Returns the position of the first element of L (or null if empty).
- last(): Returns the position of the last element of L (or null if empty)
- before(p): Returns the position of L immediately before position p (or null if p is the first
- position)
- after(p): Returns the position of L immediately after position p (or null if p is the last position)

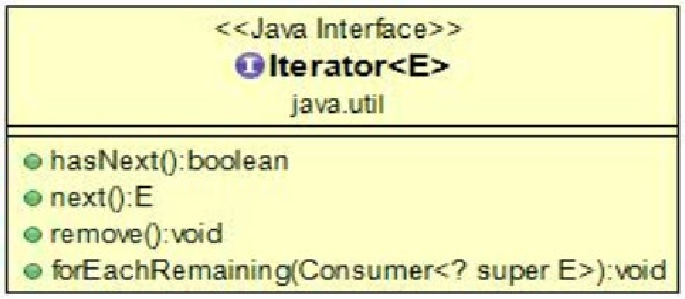
- isEmpty(): Returns true if list L does not contain any elements.
- size(): Returns the number of elements in list L

The positional list ADT also includes the following update methods:

- addFirst(e): Inserts a new element e at the front of the list, returning the position of the new element
- addLast(e): Inserts a new element e at the back of the list, returning the position of the new element
- addBefore(p, e): Inserts a new element e in the list, just before position p, returning the position of the new element
- addAfter(p, e): Inserts a new element e in the list, just after position p, returning the position of the new element
- set(p, e): Replaces the element at position p with element e, returning the element formerly at position p
- remove(p): Removes and returns the element at position p in the list, invalidating the position

Iterators: An iterator is a software design pattern that abstracts the process of scanning through a sequence of elements, one element at a time. The underlying elements might be stored in a container class, streaming through a network, or generated by a series of computations. Java defines the *java.util.Iterator* interface with the following three methods:

- hasNext(): Returns true if there is at least one additional element in the sequence, and false otherwise
- next(): Returns the next element in the sequence
- remove(): Removes from the collection the element returned by the most recent call to next(). Throws an IllegalStateException if next has not yet been called, or if remove() was already called since the most recent call to next.



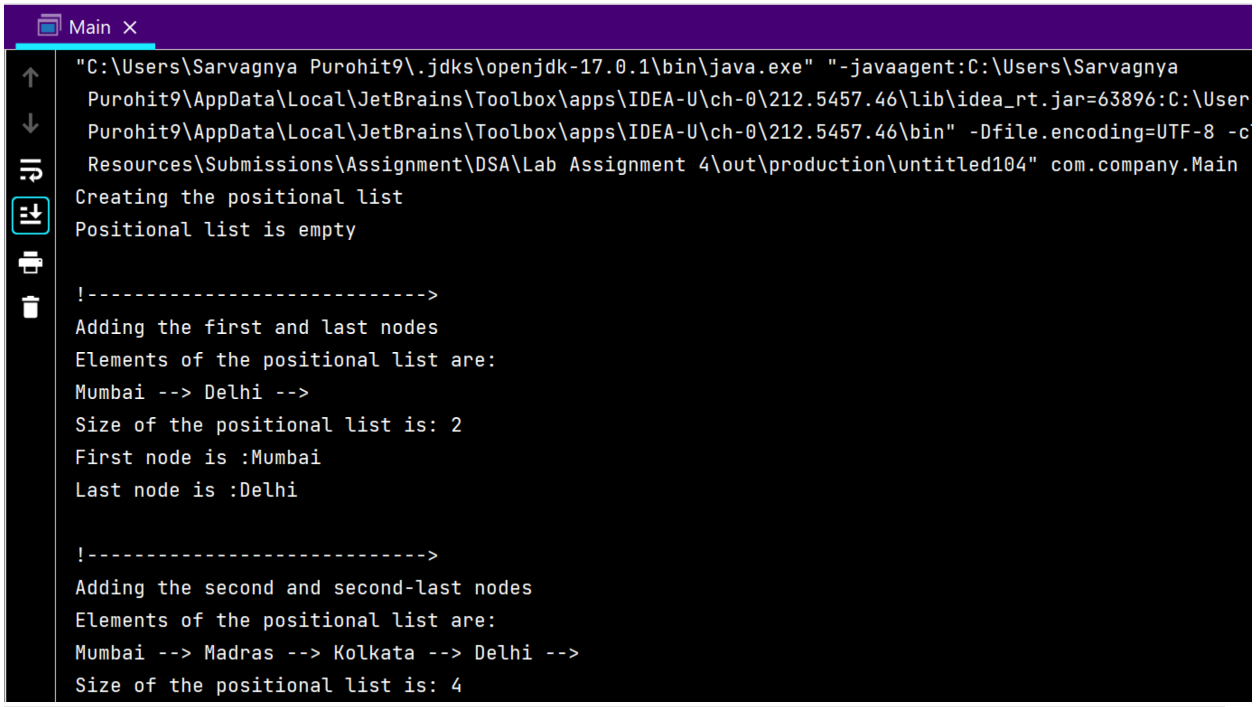
Iterable: A single iterator instance supports only one pass through a collection; calls to next can be made until all elements have been reported, but there is no way to “reset” the iterator back to the beginning of the sequence. However, a data structure that wishes to allow repeated iterations can support a method that returns a new iterator, each time it is called. To provide greater standardization, Java defines another parameterized interface, named *Iterable*, that includes the following single method:

- iterator(): Returns an iterator of the elements in the collection

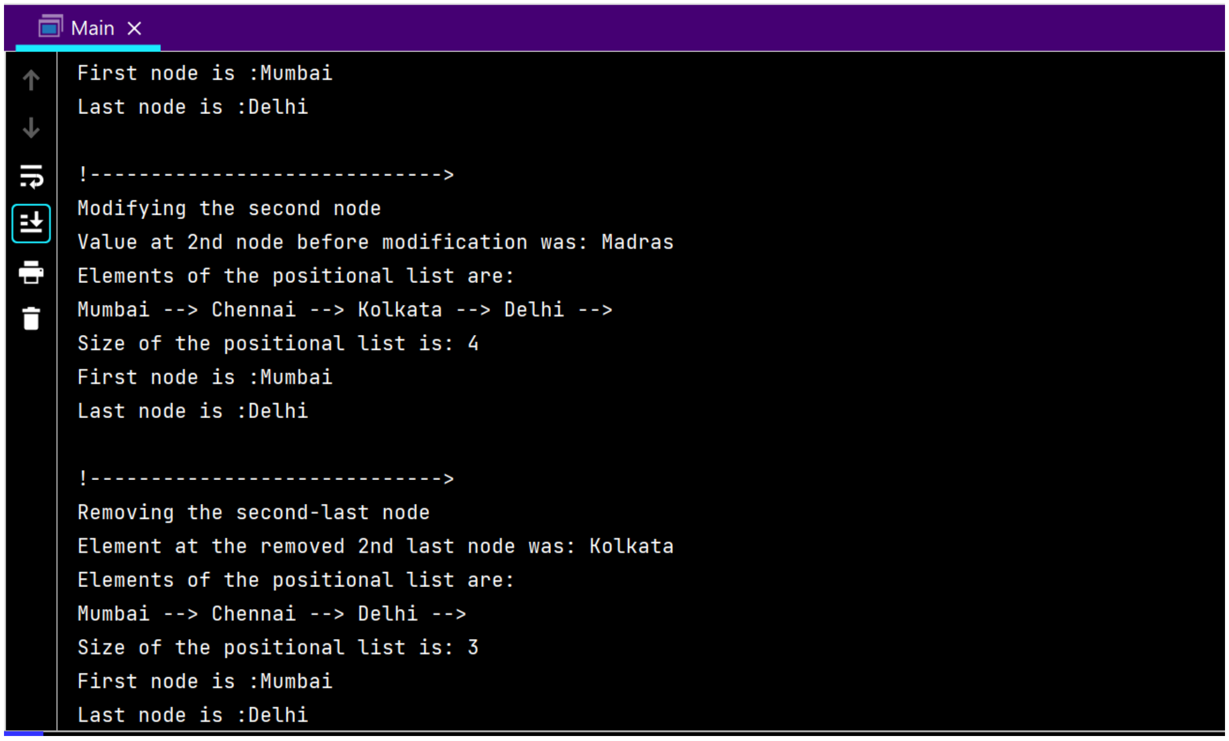
Each call to iterator() returns a new iterator instance, thereby allowing multiple (even simultaneous) traversals of a collection.

In the LinkedPositionalList class, we will allow both forms of iterations i.e., we will have an iterator for both - the elements of the list and the positions involved in the list.

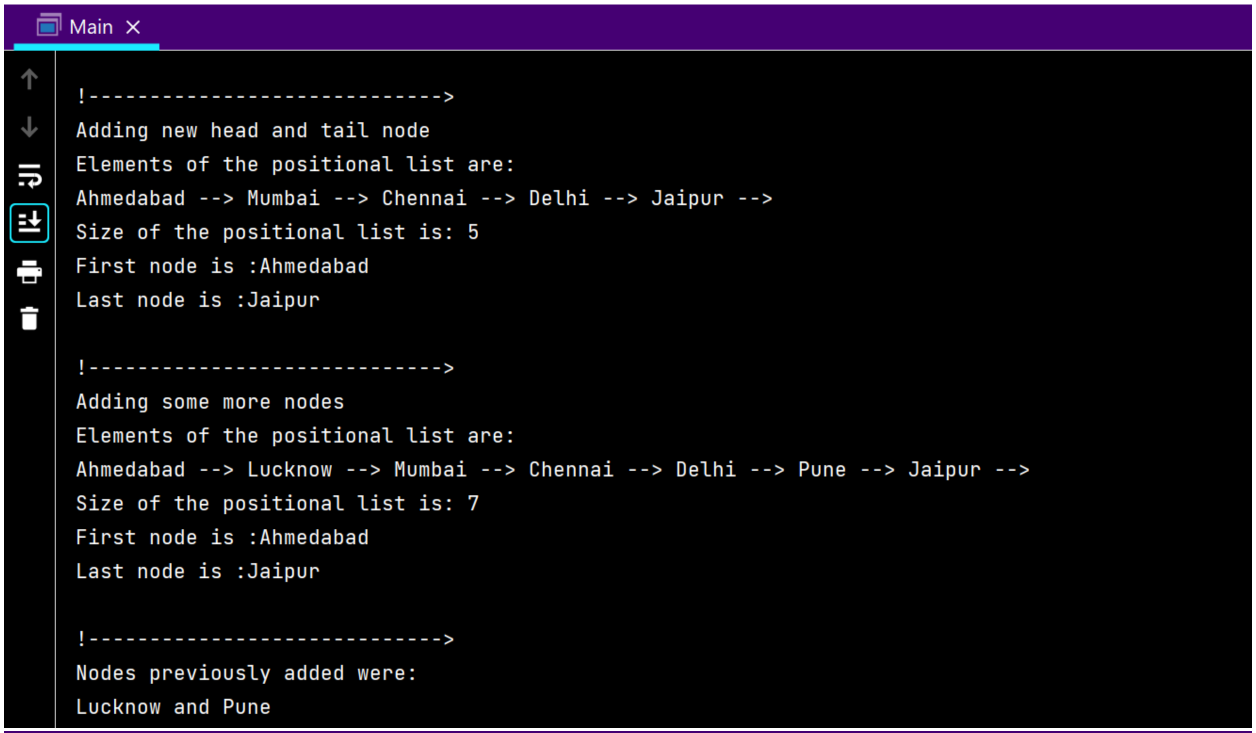
Test Data and Output of the program:



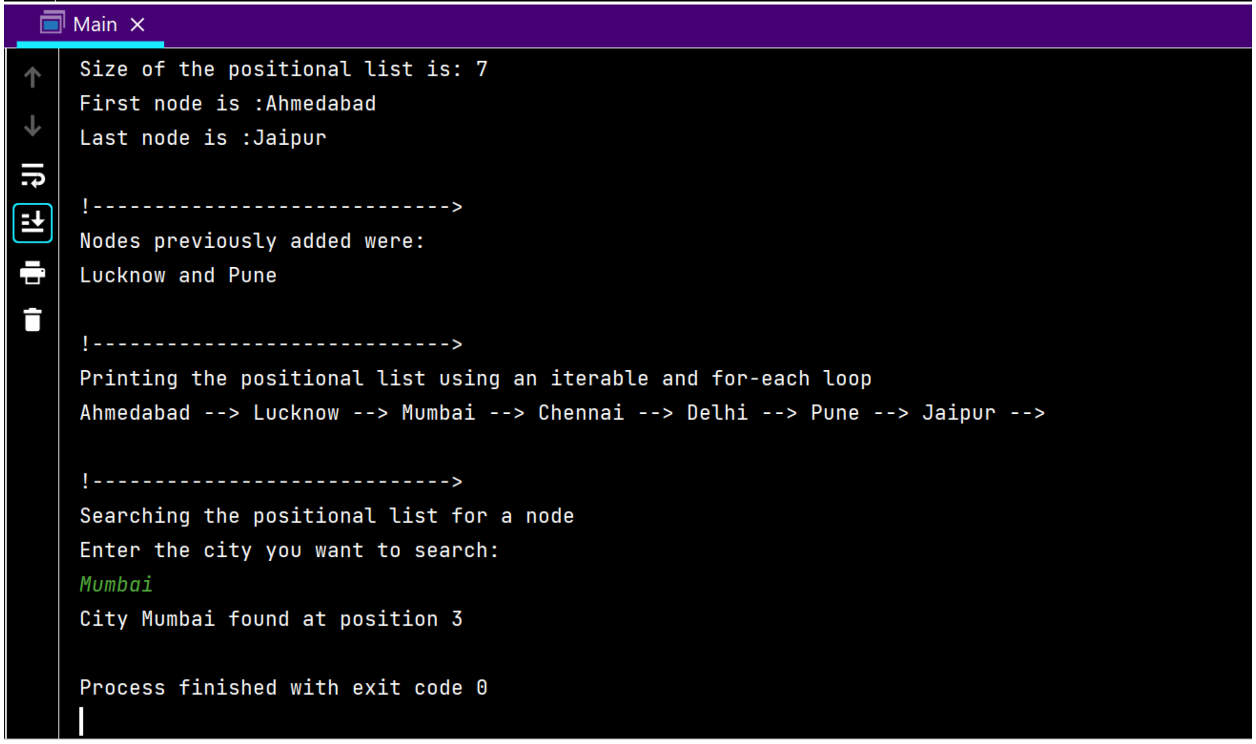
```
"C:\Users\Sarvagnya Purohit9\.jdks\openjdk-17.0.1\bin\java.exe" "-javaagent:C:\Users\Sarvagnya
 Purohit9\AppData\Local\JetBrains\Toolbox\apps\IDEA-U\ch-0\212.5457.46\lib\idea_rt.jar=63896:C:\User
 Purohit9\AppData\Local\JetBrains\Toolbox\apps\IDEA-U\ch-0\212.5457.46\bin" -Dfile.encoding=UTF-8 -c
 Resources\Submissions\Assignment\DSA\Lab Assignment 4\out\production\untitled104" com.company.Main
Creating the positional list
Positional list is empty

!---------------------------->
Adding the first and last nodes
Elements of the positional list are:
Mumbai --> Delhi -->
Size of the positional list is: 2
First node is :Mumbai
Last node is :Delhi

!---------------------------->
Adding the second and second-last nodes
Elements of the positional list are:
Mumbai --> Madras --> Kolkata --> Delhi -->
Size of the positional list is: 4
```



```
First node is :Mumbai
Last node is :Delhi

!---------------------------->
Modifying the second node
Value at 2nd node before modification was: Madras
Elements of the positional list are:
Mumbai --> Chennai --> Kolkata --> Delhi -->
Size of the positional list is: 4
First node is :Mumbai
Last node is :Delhi

!---------------------------->
Removing the second-last node
Element at the removed 2nd last node was: Kolkata
Elements of the positional list are:
Mumbai --> Chennai --> Delhi -->
Size of the positional list is: 3
First node is :Mumbai
Last node is :Delhi
```



```
!---------------------------->
Adding new head and tail node
Elements of the positional list are:
Ahmedabad --> Mumbai --> Chennai --> Delhi --> Jaipur -->
Size of the positional list is: 5
First node is :Ahmedabad
Last node is :Jaipur

!---------------------------->
Adding some more nodes
Elements of the positional list are:
Ahmedabad --> Lucknow --> Mumbai --> Chennai --> Delhi --> Pune --> Jaipur -->
Size of the positional list is: 7
First node is :Ahmedabad
Last node is :Jaipur

!---------------------------->
Nodes previously added were:
Lucknow and Pune
```



```
Size of the positional list is: 7
First node is :Ahmedabad
Last node is :Jaipur

!---------------------------->
Nodes previously added were:
Lucknow and Pune

!---------------------------->
Printing the positional list using an iterable and for-each loop
Ahmedabad --> Lucknow --> Mumbai --> Chennai --> Delhi --> Pune --> Jaipur -->

!---------------------------->
Searching the positional list for a node
Enter the city you want to search:
Mumbai
City Mumbai found at position 3

Process finished with exit code 0
```

Conclusion:

In this lab assignment, we learnt about positional lists. While implementing linked lists, searching through the linked list wasn't efficient and random access wasn't possible in the linked list since it only had to traverse incrementally starting from the header or the trailer. This way, the linked list wasn't cache-friendly.

By using positional lists, we can now perform arbitrary insertions and deletions anywhere in the list by introducing a more formal ADT referring to a location called "*Position*". This is much like a cursor within a text document where we refer to locations relative to the cursor and not based on any kind of indexing.
Given a position in the positional list, we can now append/prepend another position to the given position, remove the given position, get the element at that position, modify the element at that position all in O(1) time complexity.
Additionally, we also learnt about *Iterator* and *Iterable* in Java which makes iterations through any iterable collection more convenient and standard.