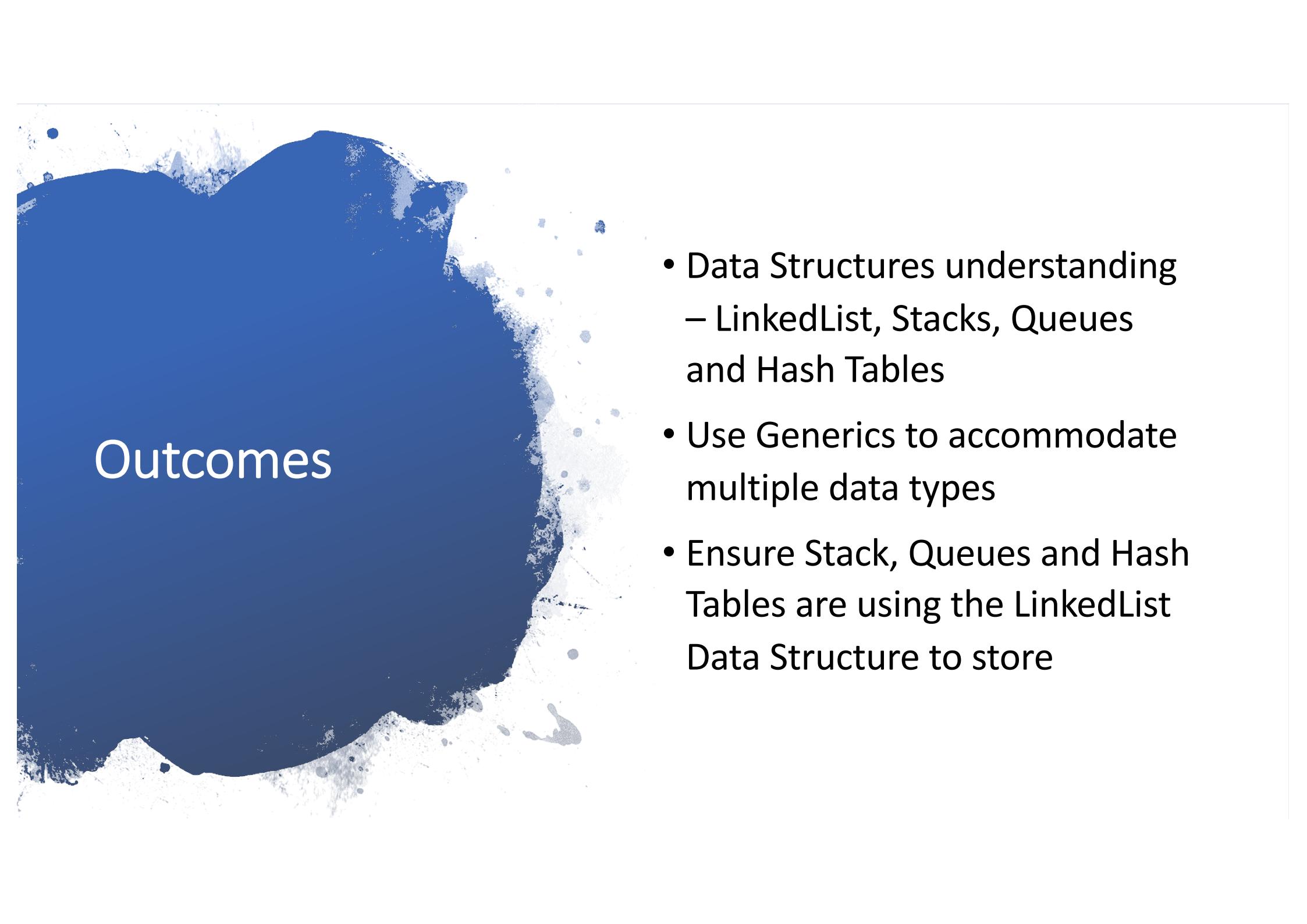




BridgeLabz

Employability Delivered

Data Structure Problems using Java Generics

A large, abstract graphic in the background features a dark blue shape on the left and a white, textured shape on the right, separated by a jagged white line. The overall effect is organic and modern.

Outcomes

- Data Structures understanding
 - LinkedList, Stacks, Queues and Hash Tables
- Use Generics to accommodate multiple data types
- Ensure Stack, Queues and Hash Tables are using the LinkedList Data Structure to store

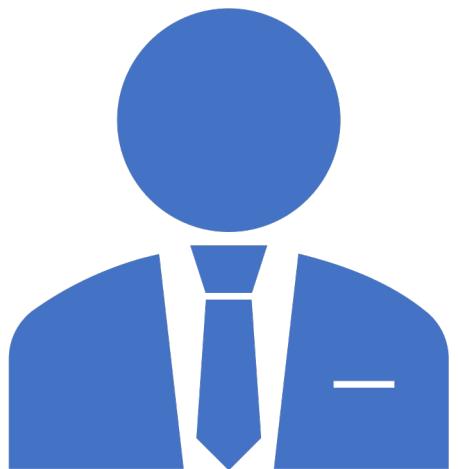
Section 1: LinkedList

The Unordered List Abstract Data Type

The structure of an unordered list, as described above, is a collection of items where each item holds a relative position with respect to the others. Some possible unordered list operations are given below.

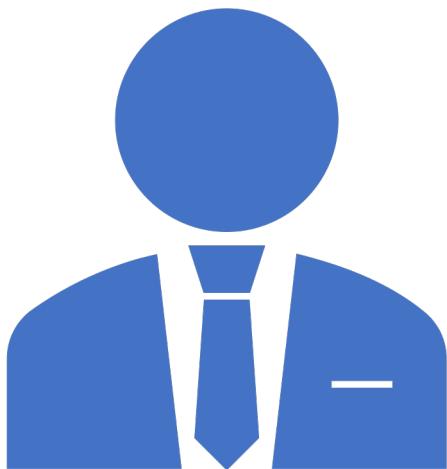
- `List()` creates a new list that is empty. It needs no parameters and returns an empty list.
- `add(item)` adds a new item to the list. It needs the item and returns nothing. Assume the item is not already in the list.
- `remove(item)` removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.
- `search(item)` searches for the item in the list. It needs the item and returns a boolean value.
- `isEmpty()` tests to see whether the list is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items in the list. It needs no parameters and returns an integer.
- `append(item)` adds a new item to the end of the list making it the last item in the collection. It needs the item and returns nothing. Assume the item is not already in the list.
- `index(item)` returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list.
- `insert(pos,item)` adds a new item to the list at position pos. It needs the item and returns nothing. Assume the item is not already in the list and there are enough existing items to have position pos.
- `pop()` removes and returns the last item in the list. It needs nothing and returns an item. Assume the list has at least one item.

UnOrdered
LinkedList
API



UC 1

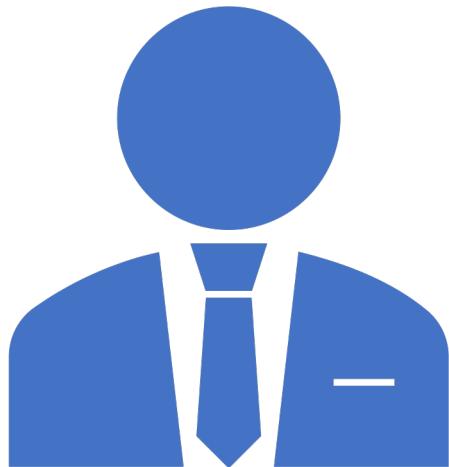
Lets create a
simple Linked List
of 56, 30 and 70



UC 2

Ability to create Linked List by adding 30 and 56 to 70

- Node with data 70 is First Created
- Next 30 is added to 70
- Finally 56 is added to 30
- LinkedList Sequence: 56->30->70



UC 3

Ability to create Linked List by appending 30 and 70 to 56

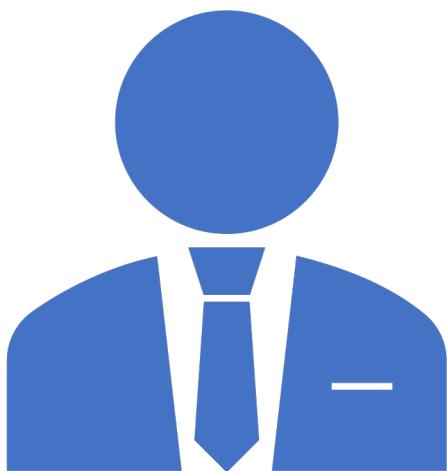
- Node with data 56 is First Created
- Next Append 30 to 56
- Finally Append 70 to 30
- LinkedList Sequence: 56->30->70



UC 4

**Ability to insert
30 between 56
and 70**

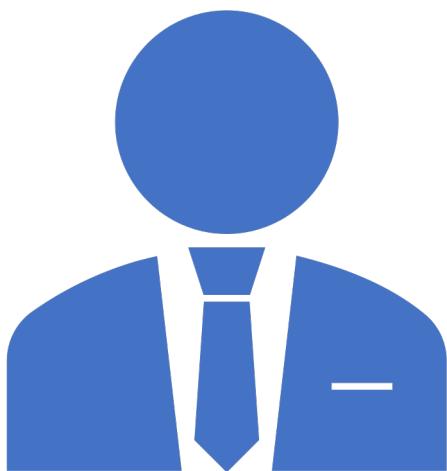
- Final Sequence: 56->30->70



UC 5

Ability to delete the first element in the LinkedList of sequence 56->30->70

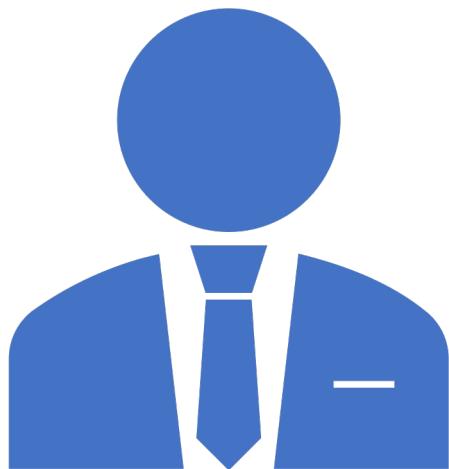
- Write pop method
- Note there is new head
- Final Sequence: 30->70



UC 6

Ability to delete the last element in the LinkedList of sequence 56->30->70

- Write popLast method
- Note there is new tail
- Final Sequence: 56->**30**



**Ability to search
LinkedList to find Node
with value 30**

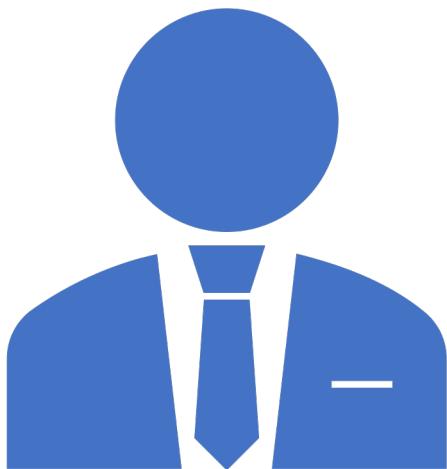
- Write Junit Test Case as demonstrated in class
- Loop through LinkedList to find node with key 30



UC 8

Ability to insert 40 after 30 to
the Linked List sequence of
56->30->70

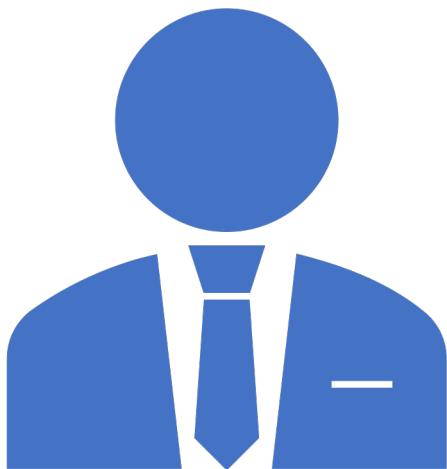
- Write Junit Test Case as demonstrated in class
- Search LinkedList to get Node with key value 30
- Then Insert 40 to 30
- Final Sequence: 56->30->40->70



UC 9

Ability to delete 40 from the Linked List sequence of 56->30->40->70 and show the size of LinkedList is 3

- Write Junit Test Case as demonstrated in class
- Search LinkedList to find node with key value 40
- Delete the node
- Implement size() and show the Linked List size is 3
- Final Sequence: 56->30->70



UC 10

Ability to create Ordered Linked List in ascending order of data entered in following sequence 56, 30, 40, and 70

- Refactor the code to create SortedLinkedList Class
- Create Node that takes data that is Comparable
- Perform Sorting during the add method call
- Final Sequence: 30->40->56->70

Ordered LinkedList API

The Ordered List Abstract Data Type

We will now consider a type of list known as an ordered list. For example, if the list of integers shown above were an ordered list (ascending order), then it could be written as 17, 26, 31, 54, 77, and 93. Since 17 is the smallest item, it occupies the first position in the list. Likewise, since 93 is the largest, it occupies the last position.

The structure of an ordered list is a collection of items where each item holds a relative position that is based upon some underlying characteristic of the item. The ordering is typically either ascending or descending and we assume that list items have a meaningful comparison operation that is already defined. Many of the ordered list operations are the same as those of the unordered list.

- `OrderedList()` creates a new ordered list that is empty. It needs no parameters and returns an empty list.
- `add(item)` adds a new item to the list making sure that the order is preserved. It needs the item and returns nothing. Assume the item is not already in the list.
- `remove(item)` removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.
- `search(item)` searches for the item in the list. It needs the item and returns a boolean value.
- `isEmpty()` tests to see whether the list is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items in the list. It needs no parameters and returns an integer.
- `index(item)` returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list.
- `pop()` removes and returns the last item in the list. It needs nothing and returns an item. Assume the list has at least one item.



BridgeLabz

Employability Delivered

Thank
You