List ADT

Wednesday, 24 January 2018 9:52 AM

During compilation: checked exception

Implementations

- Arrays vs. Linked Lists

Arrays

- Retrieval is fast, O(1)
- Dynamic operations (insertion/ deletion) are slow, O(n)
    - Splicing and compacting - have to update size
- Size limited to MAXSIZE (need to know beforehand)
    - Can create new array when run out of space
    - Good for fixed-size lists
- When removing elements, not necessary to delete last element (size given)
- Elements are contiguous > runs out of memory as computer memory is fragmented

Generic Array

- Use <E> to indicate type of objects
- Primitive types can be wrapped to be treated as objects

`ArrayList<E>`

- Dynamic size (similar to Vector)
- Underlying data structure same as Array

Linked Lists

- Allow elements to be non-contiguous in memory
- Associate each element with its neighbour (contains reference to neighbour)
- Create using generic Java
- Basic, Extended, Circle, Tail and Double Linked List

Insertion:

- Must have reference pointing to object, else Garbage Collector will remove object

Linked List Node

- Contains element and next ListNode
- Needs a head pointer to indicate first node
- Linked list is created backwards (use head to point to current head)

BasicLinkedList

- Interface tells what functions are available
- Can only add/ remove from front
- Good for stacks

ExtendedLinkedList

- `extends BasicLinkedList`
- Can insert/ delete after

Tailed Linked List

- More efficient process to access end of list

- `extends ExtendedLinkedList`
- Add a tail pointer to the last element
    - Create node, extend list, change tail pointer
- Queues (FIFO)

Double Linked List:

- Traverse sequence in both directions
- `extends TailedLinkedList`
- Casting to access subclass fields and methods
- `current.next`
    - Points to a list node as it is a list node method

Circular Linked List

- Last node points to first