* List
  + A sequence of elements
  + May contain duplicates
  + Allows for null elements
  + Expandable & Shrinkable
  + Available as an interface java.util.list
  + Boolean add(E e)
  + Void add( int index, E e)
  + E get(int index)
  + Boolean isEmpty()
  + Iterator<E> iterator
    - Returns and Iterator over the elements in the list
* ArrayList
  + Simplest class that implements the List interface
  + Array have a fixed size, ArrayList is dynamic
  + Constant time access to elements
  + Insertion and Removal is difficult
* Linked List
  + Linked lists may grow and shrink
  + Linear Time access
  + Linear time insertion and removal.
    - Except if previous element is supplied, then it is constant.
  + Each element has a reference to the next element
* Single Linked List
  + addFirst(E item)
    - adds element to the front of linked list
  + addAfter(Node<E> node)
  + removeAfter(Node<E> node)
* With a linked list representation, overflow will not happen
* Underflow can happen, and should be handled the same way as for n array implementation.
* When a node is popped from a list and the node references an object the reference does not need to be set to null.
  + Unlike an array implementation, it really is removed – you can no longer get to it from the linked list
* Single Linked list implementation of Queue
  + In a SingleLinkedList you can easily find the successor of a node, but not its predecessor
    - Pointers (references) are one way
  + If you know where the last node in a list it’s hard to remove that node, but it’s easy to ass a node after it.
    - Use the first element in a SingleLinkedList as the front of the queue
    - Use the last element in a SingleLinkedList as the back of the queue
    - Keep pointers to both the front and the rear of the SingleLinkedList.