Midterm Review

worksheet can be found on kevin-miao.com

Today
+ Linked Lists / Deques

Deque

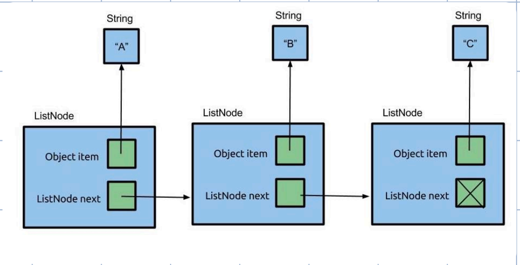
Double-Ended Queue :
can add items to front and back.

Linked Lists

Vanilla

['A', 'B', 'C']

x IntList $\longrightarrow$



- Enhancements
  1 - Encapsulation : Abstraction
  2 - Sentinel : No null Handling
  3 - Doubly-Linked : efficiency
  4 - Generic Lists

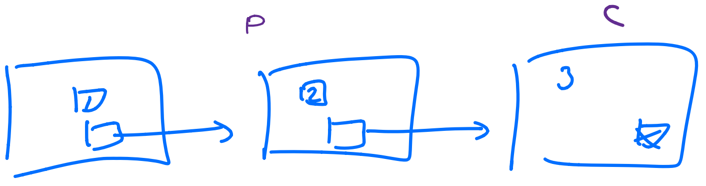
- Destructive vs Non-Destructive
  - Destructive ↓ Mutative (void)
  - Non-Destructive ↓ Returns new List

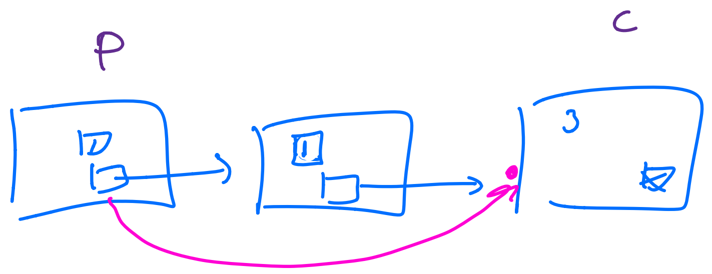
4 IntLists

Spring 2017 MT 1

Fill in the blanks below to correctly implement removeDuplicates.

```
public class IntList {
    public int item;
    public IntList next;
    public IntList (int f, IntList r) {
        this.item = f;
        this.next = r;
    }
    /**
     * Given a sorted linked list of items - remove duplicates.
     * For example given 1 -> 2 -> 2 -> 2 -> 3,
     * Mutate it to become 1 -> 2 -> 3 (destructively)
     */
    public static void removeDuplicates(IntList p) {
        if (p == null) {
            return;
        }
        IntList current = p.next;
        IntList previous = p;
        while (current != null) {
            if (current.item == previous.item) {
                previous.next = current.next;
            } else {
                previous = current;
            }
            current = current.next;
        }
    }
}
```





5 DLLists

Summer 2018 MT 1

Implement DLList.duplicate, which destructively copies each node in the list, resulting in a list that's twice as long. For example, if the list initially contained [1, 2, 3], it should contain [1, 1, 2, 2, 3, 3] after the call to duplicate.

```
public class DLList<BleepBlorp> {
    private class Node {
        public Node prev;
        public BleepBlorp item;
        public node next;
        public Node(BleepBlorp i, Node p, Node n) {
            item = i;
            prev = p;
            next = n;
        }
    }
    private Node sentinel;
    private int size;
    public void duplicate() {
        Node p = sentinel.next;
        while (                        ) {
            Node copy = new Node(         );
            ___________________________________;
            ___________________________________;
            ___________________________________;
        }
        size *= 2;
    }
}
```

6 Deques

Spring 2017 MT 1

On project 1, you implemented the Deque interface as ArrayDeque and LinkedListDeque. In this problem, you'll improve your Deque interface by adding a new default method. Your job: Add a default method remove(Item x) that removes all items equal to x. This method should return true if anything was removed. It should remove false if nothing was removed. You may not use the new keyword. Only write one statement per line. You may not need all the lines. If you need to determine that two Items are equal, use the .equals method, i.e. don't use == to compare Items.

```
public interface Deque<Item> {
    void addFirst(Item x); void addLast(Item x);
    boolean isEmpty(); int size();                      [null, 1, 2, 3]
    void printDeque(); Item get(int index);
    Item removeFirst(); Item removeLast();
    default boolean remove(Item x) {
        if (isEmpty()) { return False ;}

        int OGSize = size();
        for (int i=0; i < OGSize; i++) {
              item temp = remove First();
              if ((temp!=null && temp.equals(x)) || temp == x)
        {           addLast(temp);              }

                    }

        return OGSize != size();

    } /* Answers using the new keyword will not be given credit. */
}
For example, the test below should pass:
Deque<Dog> dd = new ArrayDeque<Dog>();                  ~~null. equals~~ ?
dd.addLast(new Dog(\Ljilja"));
dd.addLast(new Dog(\Rikhard"));
dd.addLast(new Dog(\Spartacus");
dd.addLast(new Dog(\Rikhard"));
assertEquals(4, dd.size());
boolean rikhardRemoved = dd.remove(\Rikhard");
assertTrue(rikhardRemoved);
assertEquals(2, dd.size());
```

Reminder: You may not use the new keyword! This is not just an arbitrary restriction, but ensures that we don't use any unnecessary space.