Data Structure

Recitation IX
Topic

- List ADT
- Lab 4
List

- A list is a collection of elements, with a linear relationship existing among its elements.

- Each element on the list except the first one has a unique predecessor and each element except the last one has a unique successor.
Assumptions for Our Lists

- Unbounded, if it implemented by array, then add to a full list, the capacity of the underlying array is increased.
- Allow duplicate, find can return any one of duplicates
- No null element
- Sorted lists are sorted in increasing order, compareTo()
- x.equals(y) is equivalent to x.compareTo(y)==0
- Indexes of the list are contiguous, starting at 0. Any invalid range of index raise an exception
- Lists maintain their current position, the position of the next element accessed by getNext(). This position is incremented by getNext(), zeroed by reset(), and unchanged otherwise.
List Interface

Public interface ListInterface<T>
{
    int size();
    void add(T element);
    boolean contains(T element);
    boolean remove(T element);
    T get(T element);
    String toString();
    void reset();
    T getNext();
}
reset() and getNext

- **Iteration**: a mechanism to process the entire list, element by element, from the first element to the last element.

- At any point in time there is a current position within the list

- reset() sets the current position to the first element

- getNext() returns the element at the current position and advances to the next position.
Iteration

list.reset()

for (int i=0; i<list.size(); i++) {
    element = list.getNext();
    // play with element
}

Iteration

- What happen if we insert or remove an element in the middle of iterating through the structure?
- Change the size of list, making the termination condition of the iteration counting loop invalid (repeating or skipping elements)
- Preconditions whenever getNext() is called:
  - list is not empty.
  - list.reset() has been called.
  - list has not been modified since the most recent list.reset().
public interface IndexedListInterface<T> extends ListInterface<T>
{
    void add(int index, T element);
    T set(int index, T element);
    T get(int index);
    int indexOf(T element);  //index of the first element
    T remove(int index);
}

add(): inserts the given element at the specified index, shifting other elements down to make room. All current elements at positions higher than that position have 1 added to their index

remove(): deletes and returns the element at the specified index, the resulting gap is filled. All current elements at positions higher than that position have 1 subtracted from their index
ListInterface<String> list1 =
    new ArrayUnsortedList<String>(3);
list1.add("Wirth");
list1.add("Dykstra");
list1.add("DePasquale");
list1.add("Dahl");
list1.add("Nygaard");
list1.remove("DePasquale");
System.out.print("Unsorted ");
System.out.println(list1);
Example

ListInterface<String> list2 =
    new ArraySortedList<String>(3);

list2.add("Wirth");
list2.add("Dykstra");
list2.add("DePasquale");
list2.add("Dahl");
list2.add("Nygaard");
list2.remove("DePasquale");
System.out.print("Sorted ");
System.out.println(list2);

Sorted List:
Dahl
Dykstra
Nygaard
Wirth
Example

```java
IndexedListInterface<String> list3 = new ArrayIndexedList<String>(3);
list3.add("Wirth");
list3.add("Dykstra");
list3.add("DePasquale");
list3.add("Dahl");
list3.add("Nygaard");
list3.remove("DePasquale");
System.out.print("Indexed ");
System.out.println(list3);
```

Indexed List:
[0] Wirth
[1] Dykstra
[2] Nygaard
main()
Loop N times
flip a coin
if heads
produce()
else
consume()

produce()
if queue is not full
generate an item
enqueue the item

consume()
if queue is not empty
dequeue an item
numberSuccessfulConsumers++

public static boolean coinFlipIsHeads() {
    return Math.random() < HEAD_PROBABILITY;
}
For each HEAD_PROBABILITY(0 - 1)

Calculate numberSuccessfulConsumers/
(number_iterations/2)
• Method used in book: one stack and one queue
  • public static boolean isPalindromeFancy(String s)

• Method used in 101
  • public static boolean isPalindromePlain(String s)
  • A better program is to attack the string with charAt() from both ends. This accesses each character only once.
public static boolean isPalindromePlain(String s)
{
    if (s.length() <= 1) // base case 1
        return true;
    if (s.charAt(0) == s.charAt(s.length()-1)) // base case 2
        return false;
    return isPalindrome(s.substring(1,s.length()-1));
}

Two base cases are that an empty string or a string of length one is always a palindrome.

Another base case is that if the first and last characters are not equal, then it is not a palindrome.

If neither base case applies than the original string is a palindrome if and only if the substring omitting the first and last characters (a smaller problem closer to the first two base cases) is a palindrome.
public static String genRandomPalindrome(int N)

public static String genRandomNonPalindrome(int N)

For example, assume N=5 and the 5 random integers are 0, 23, 9, 25, 2. Then the random string is axjzc.

genRandomNonPalindrome() returns axjzcaxjzc

genRandomPalindrome() returns axjzcczjxa.
public static String genRandomPalindrome(int N) {
    String s = getOriginalString(N);
    return s+s;
}

public static String getOriginalString(int N) {
    String result = "";
    Loop(N times) {
        random choose form 26-character
        append to result
    }
    return result;
}
Lab 4 2B

How to get the elapsed time (Date class in java.util):
Before the loop:

   startTime = (new Date()).getTime();

End of the loop:

   elapsedTime = (new Date()).getTime() - startTime;
Lab 4 2B

String s;

long startTime = (new Date()).getTime();

Loop(M times){
  If(isPalindromeFancy(s) != true){break; //error}
}

elapsedTime = (new Date()).getTime() - startTime;
Java Queue

- [http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Queue.html](http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Queue.html)

- E element()
  - Retrieves, but does not remove, the head of this queue.

- boolean offer(E o)
  - Inserts the specified element into this queue, if possible.

- E peek()
  - Retrieves, but does not remove, the head of this queue, returning null if this queue is empty.

- E poll()
  - Retrieves and removes the head of this queue, or null if this queue is empty.

- E remove()
  - Retrieves and removes the head of this queue.
import java.util.Queue;
import java.util.LinkedList;

public class TestQueue {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<String>();
        queue.offer("Hello");
        queue.offer("World");
        queue.offer("?");
        queue.offer("?");
        System.out.println(queue.size());
        String str;
        while((str=queue.poll())!=null){
            System.out.print(str);
        }
        System.out.println(queue.size());
    }
}