CSE12 Discussion 6

Pooja Bhat

10am Midterm Question

Let st1 and st2 be two stacks of size 5 and let the elements {6, 7, 4, 3, 8} be added to st1 and elements {1, 5, 6, 4, 2} be added to st2 in a given order, with no other operations in between. Let q, be an empty queue. What is the value of the variable 'val' and the output of the print method at the end of the following SecretMethod:

A) val = 5 q = [2, 1, 3]
B) val = 1 q = [null]
C) val =1 q = [3, 4]
D) val = 2 q = [3, 4]
E) val = 2 q = [2, 4]

Answer: D

```
public static Integer SecretMethod(Stack<Integer> st1,
Stack<Integer> st2, Queue<Integer> q) {

    Integer top1, top2;
    Integer val=st1.peek();

    for (int i=0; i<st1.size(); i++) {
      top1 = st1.pop();
      top2 = st2.pop();

      if (top2 >= top1) {
        q.add(top1);
      }
      else {
        val = top2;
      }
    }
        System.out.println(q);
        return val;
  }
```

Midterm Question

```
public void complexity(int param) {
        int c;
        if (param >= 0)
            c = 1;
        else
            c = 1000;
        for(int i = n; i>=1; i=i/2){
            for (int j=0; j<=c*n; j++} {
                arr[ i ] = arr[j]+c;
            }
        }
}
```

A) Best - O(nlogn) Worst - O(n^2)
B) Best - O(nlogn) Worst - O(nlogn)
C) Best - O(nlogn) Worst - O(1000nlogn)
D) Best - O(n^2) Worst - O(n^2)

Answer: B

Midterm Coding Question

public int removeDuplicates() : Removes any duplicate items in the list. Returns the number of items removed

- *Example 1: If the list had items {2,2,3,7,9,9,15} then the resulting list will be {2,3,7,9,15} where duplicates of items 3 and 15 have been removed.*

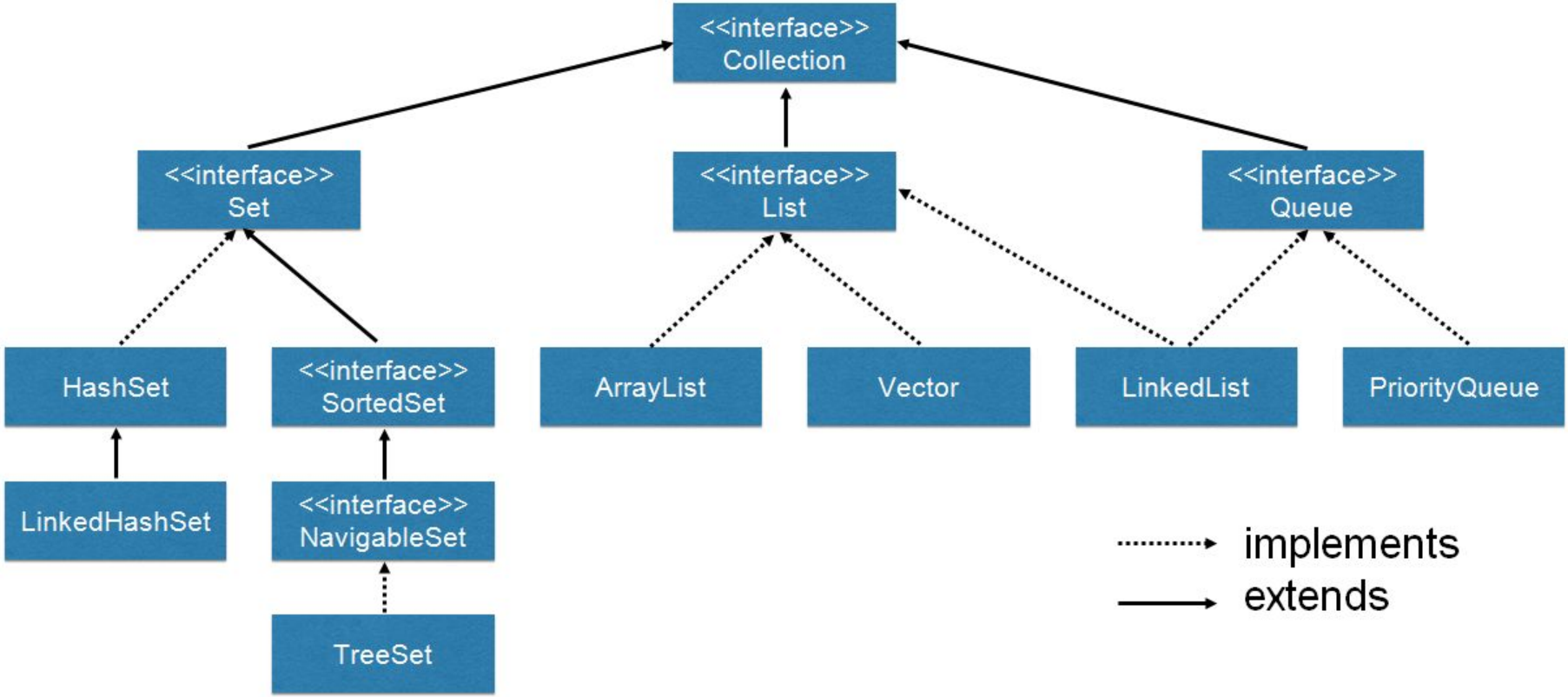
In this case the method returns 2.

- *Note: You may use the next() method you implemented before in this method.*

- *The iterator must be left back in the same position it was before the method was called.*

Generics Review

Collection Interface



Compiles or not? A: Yes B: No

1. LinkedList<List> myL = new LinkedList<List>();
//Good. same type

2. Collection<List> c = new LinkedList<List>();
//Good because LinkedList implements Collection

3. LinkedList<List> myL;
   myL = new LinkedList<ArrayList>();
//Error! Type mismatch: cannot convert from LinkedList<ArrayList> to LinkedList<List>

4. LinkedList<? extends List> myL = new LinkedList<ArrayList>();
//Good: Holds any subtype of list

A: Yes B: No

5. LinkedList<? super List> myL = new LinkedList<List>();
//Good. List or any superclass of List.

6. LinkedList<? super List> myL = new LinkedList<Collection>();
Good. List extends Collection.

7. LinkedList<Collection> myL = new LinkedList<Collection>();
Good.

8. myL.add(new ArrayList());
Also good

Constrains on T

- dHeap<T>
- Sometmes the type T can't be "just any old Object", it must satisfy some conditions.
- What is the condition for T in a heap?
- The elements in a heap must be **Comparable,** and the heap must be able to call the compareTo method on every element
- For example,
  - heap.add(new Integer(5)) //OK because integers are comparable
  - heap.add(new Node()) //Not OK because Objects are not Comparable
- Hence we need to enforce the condition that T must extend Comparable interface.

Generics

- So instead of class dHeap<T>, we would have
  - class dHeap < T extends Comparable>
- Now we cannot add Node objects to the heap.
- But this still allows us to compare an Integer and a String to each other! How to fix this?
- The compareTo method takes a parameter of type T to indicate that items must be compared to the passed parameter of type T
  - Example, Integer.compareTo(Integer)
- Hence we can specify the type to compare as follows:
  - class dHeap< T extends Comparable<T>>
- This says compare Integers ONLY to Integers, Strings ONLY with Strings etc.

Relaxing our Constraints

- class dHeap < T extends Comparable<T>>
- Let's say we had a parent class Shape that extends Comparable and overrides the compareTo method to compare different Shapes.
  - Shape.compareTo(Shape)
- Now consider a child class Circle that inherits the compareTo method from Shape, but doesn't implement any compareTo method of its own
  - There is no Circle.compareTo(Circle)
- Since it inherits the compareTo method from its parent, a Circle is comparable to other Shape objects. How do we relax our constraint to allow this?
  - class dHeap <T extends Comparable < ? super T>>
- This says that T should be comparable to any object of type T, or a type that is a super class of T.
- So this allows a Circle to be compared to other Shape objects

Heaps Review

Binary Heaps

- A heap is a specialized tree-based data structure that satisfies the ‘heap property’. It can be classified further as either a "max heap" or a "min heap".

- In a max heap, the keys of parent nodes are always greater than or equal to those of the children.

- In a min heap, the keys of parent nodes are less than or equal to those of the children.

- What happens if the elements 1,2,3,4,5 are inserted in that order to a max-heap?

Where will you find the largest key in a min heap?

A. At the root
B. At the leaf level
C. At a non-leaf node
D. At the root's leftmost child

For a node at array index j, what is the indices of its children?

A. 2*j - 1 and 2*j - 2
B. 2*j +1 and 2*j + 2
C. 2*j and 2*j + 1
D. 2*j +1 and 2*j - 1

Binary Heaps

- Is the heap a sorted data structure? Is it an ordered data structure?
  - It is a partially ordered data structure, as there is no particular relationship among nodes on any given level, even among the siblings or cousins.
- For a node at index j, what is the index of its parent? Assuming it isn't a root, of course.
  - floor ((j-1)/2)
- Create a binary max heap by inserting 1, 5, 3, 9, 7, 2. Perform a remove operation
- The heap represented by the array 1, 2, 5, 6, 4, 3, 9 represents a binary min-heap. True/False?

dHeaps

- d-ary heaps are a generalization of binary heap(d=2) in which each node has d children instead of 2.
- Like a binary heap, a dHeap is also a nearly complete binary tree, with all levels having maximum number of nodes except the last, which is filled in left to right manner.
- For a node at index j, what are the indices of its children?
  - d*j+1 to d*j + d
- For a node at index j, what is the index of its parent?
  - floor((j-1)/d)

Does the array 20, 15, 9, 18, 6, 11, 8, 4, 7, 1, 14 represent a 3-ary max heap?

A. Yes
B. No

BubbleUp and TrickleDown

- Create a 4-ary min heap by inserting 5, 9, 4, 11, 3, 7. 1, 6 in that order.
- Perform a remove operation on the heap.
- Perform another remove operation.
- Restoring the heap property during insertion: BubbleUp
- Restoring heap property during deletion: trickleDown

Implementing dHeap and Priority Queue

- You will be implementing a dHeap, with the value of d passed to the constructor.
- You will implement both min and max heap. A boolean variable will be passed to the constructor to indicate whether the heap is a min or a max heap.
- Once you have your dHeap implementation ready, you will implement a Priority Queue that uses the methods of dHeap for its add( ) and poll( ) methods
- Adapter Design Pattern - Remember HW4?
- Stack used the methods of DoubleEndedLL - Similarly our Priority Queue here uses the methods of the dHeap. Make a note of exceptions thrown by each of them, if any!

What is a Priority Queue?

- A priority queue is an abstract data type which is like a regular queue data structure, but where additionally each element has a "priority" associated with it.
- In a priority queue, an element with high priority is served before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.
- Note that the value we use to compare in our heap is the priority for the Priority Queue. Hence an element of highest priority is always at the root.
- Most common application of a priority queue - a scheduler