CompSci 201, L16: Queues and Binary Search Trees

Announcements, Coming up

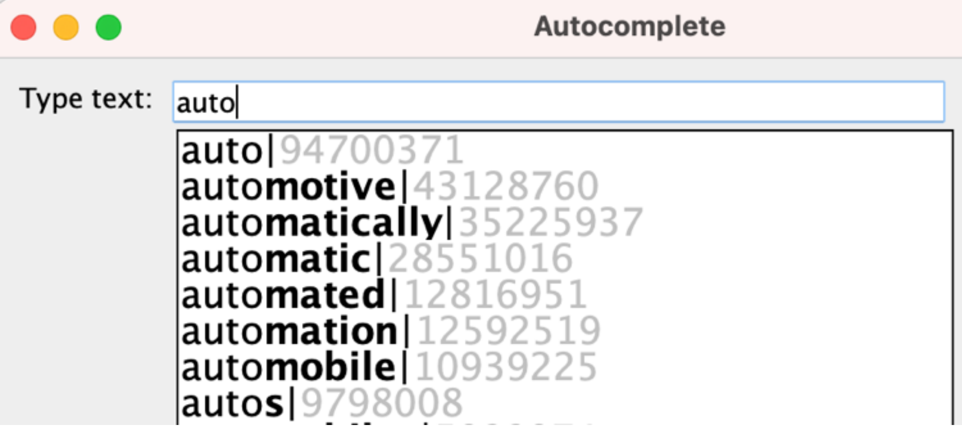
- Today, Wednesday 3/8
  - APT 6 (sorting problems) due
  - Project P4: Autocomplete released

- Friday 3/10
  - Fill out the **midsemester course survey**
  - **No discussion, enjoy spring break!**

- Wednesday 3/22
  - Midterm 2, linked list through Monday's lecture
  - Practice exams available on Sakai resources

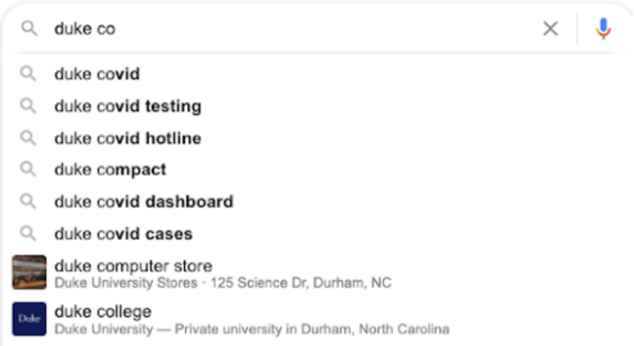
Project 4 Autocomplete

- How to create something like:



Type text: auto

```
auto|94700371
automotive|43128760
automatically|35225937
automatic|28551016
automated|12816951
automation|12592519
automobile|10939225
autos|9798008
```



duke co

- duke covid
- duke covid testing
- duke covid hotline
- duke compact
- duke covid dashboard
- duke covid cases
- duke computer store
Duke University Stores · 125 Science Dr, Durham, NC
- duke college
Duke University — Private university in Durham, North Carolina

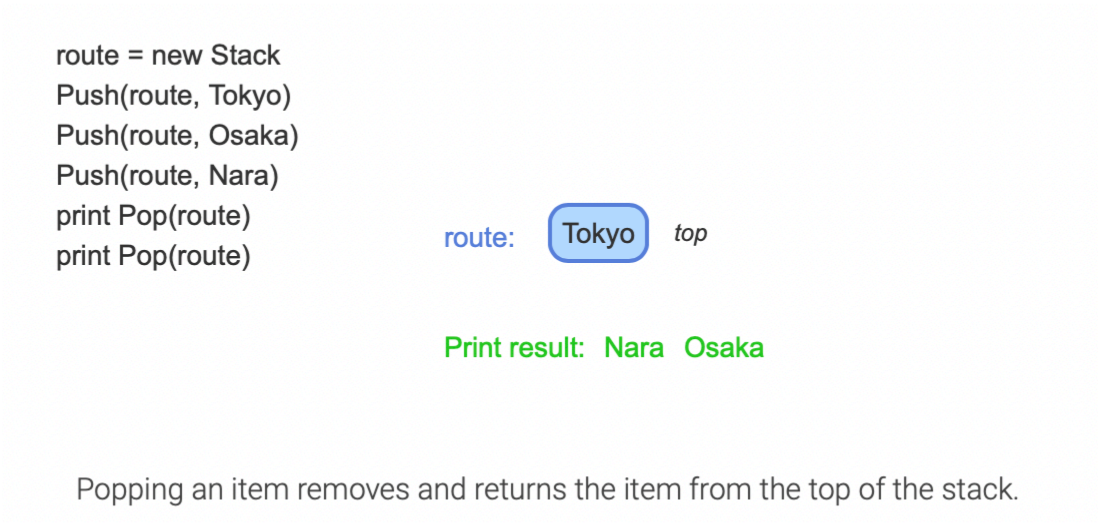
- All about two things:
  - Searching for all words that match on a prefix, and…
  - Sorting them by how common they are,
  - Return these words to show in the GUI above

Today’s Agenda

1. Stack, Queue, PriorityQueue, API perspective
2. Binary Search Tree
3. DIY TreeSet/Map

Stacks, Queues, PriorityQueue: API Perspective

Stack Abstract Data Structure: LIFO List



```
route = new Stack
Push(route, Tokyo)
Push(route, Osaka)
Push(route, Nara)
print Pop(route)
print Pop(route)
```

Print result: Nara Osaka

Popping an item removes and returns the item from the top of the stack.

Zybook

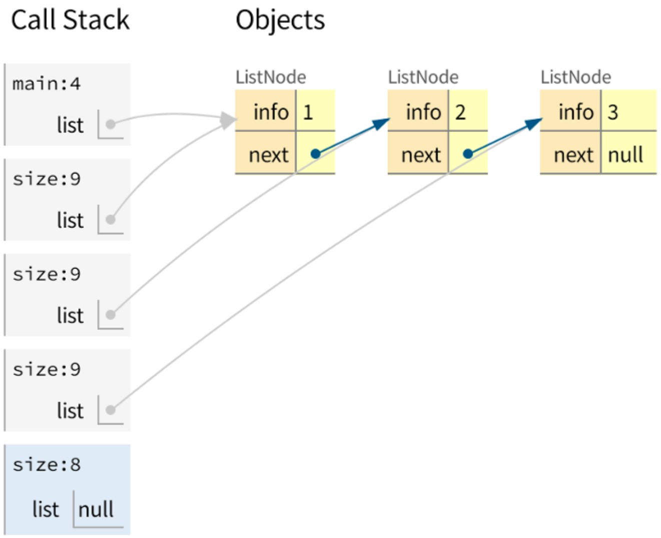
LIFO = Last In First Out

Push: Add element to stack

Pop: Get last element in

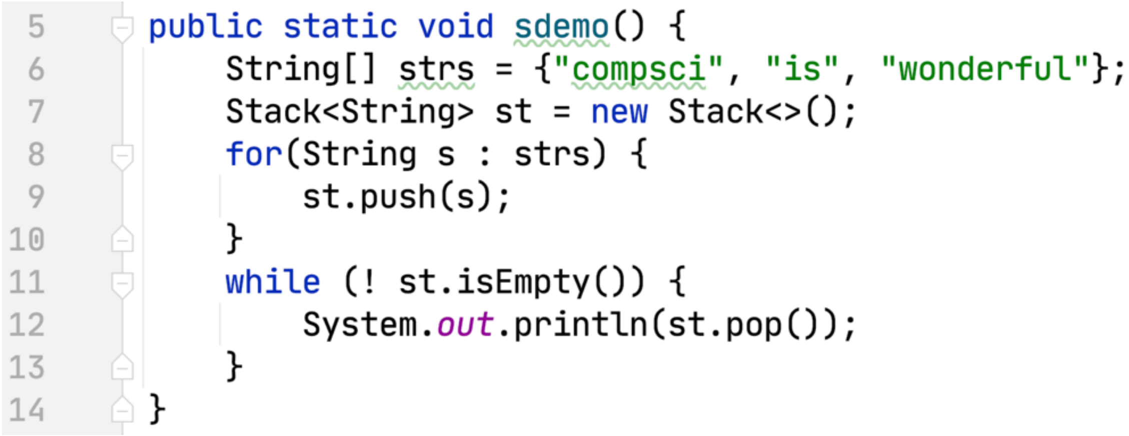
Applications? Stack in the real world?

- Remember the call stack?
- History on your web browser / back button?
- Depth-first search in a graph (more coming soon!)



`java.util.Stack` class

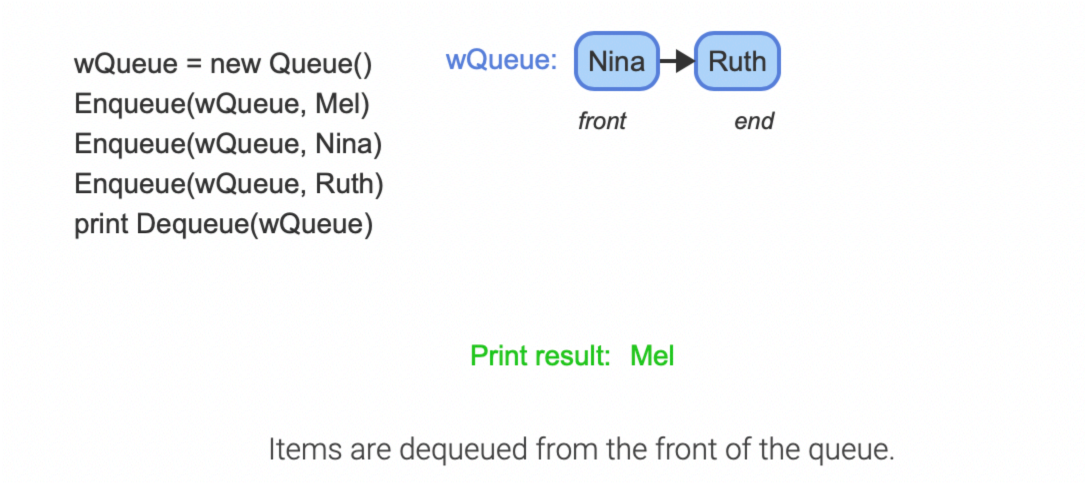
- both push and pop are O(1)
  - Adds and removes from end of `ArrayList`
  - Could also use `LinkedList`



```
public static void sdemo() {
    String[] strs = {"compsci", "is", "wonderful"};
    Stack<String> st = new Stack<>();
    for(String s : strs) {
        st.push(s);
    }
    while (! st.isEmpty()) {
        System.out.println(st.pop());
    }
}
```

```
wonderful
is
compsci
```

Queue Abstract Data Structure: FIFO List



```
wQueue = new Queue()
Enqueue(wQueue, Mel)
Enqueue(wQueue, Nina)
Enqueue(wQueue, Ruth)
print Dequeue(wQueue)
```

Print result: Mel

Items are dequeued from the front of the queue.

Zybook

FIFO = **First** In First Out

Enqueue: Add element to queue

Dequeue: Remove first in element

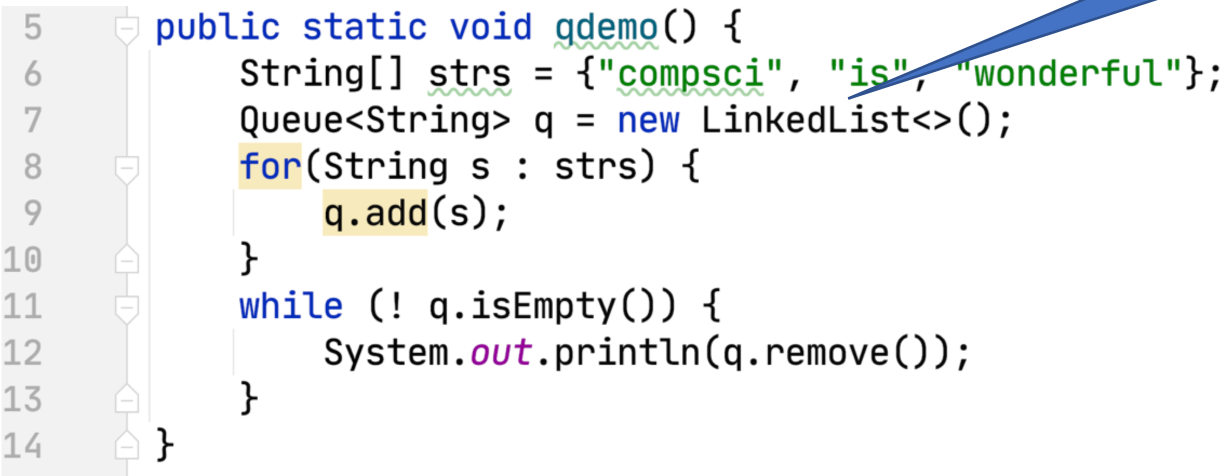
Applications? Queue in the real world?

- Operating system keeps track of which program should get processor time next.
- Waitlist for class registration on Dukehub?
- Many “shortest way to get from X to Y” problems, e.g., breadth-first search in a graph (more coming soon!)

`java.util.Queue` interface

- Both add and remove are O(1)
  - Add at end of LinkedList
  - Remove from front of LinkedList

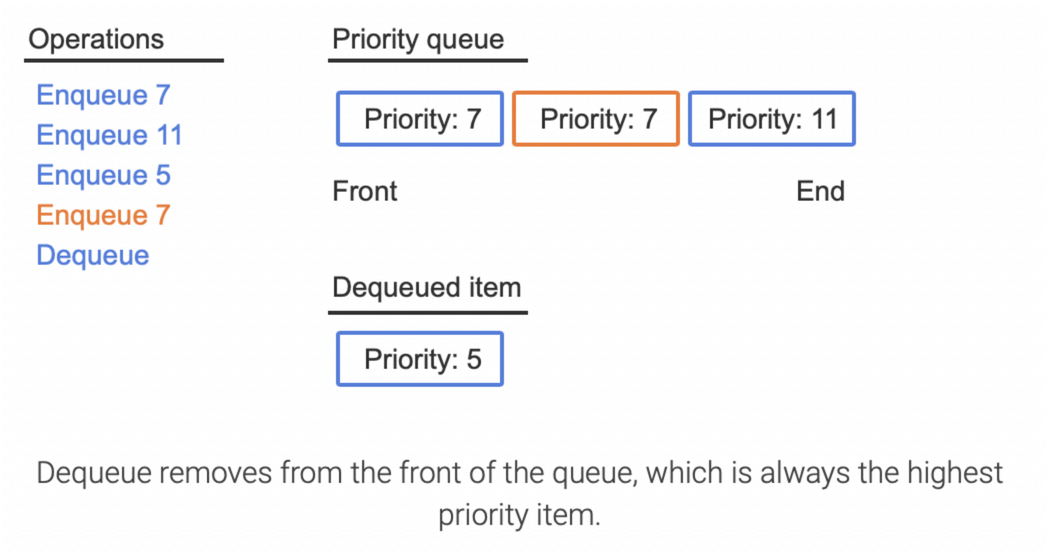
LinkedList implements the Queue interface.



```java
public static void qdemo() {
    String[] strs = {"compsci", "is", "wonderful"};
    Queue<String> q = new LinkedList<>();
    for(String s : strs) {
        q.add(s);
    }
    while (! q.isEmpty()) {
        System.out.println(q.remove());
    }
}
```

```
compsci
is
wonderful
```

Priority Queue in the Abstract



Operations

- Enqueue 7
- Enqueue 11
- Enqueue 5
- Enqueue 7
- Dequeue

Priority queue

Dequeued item

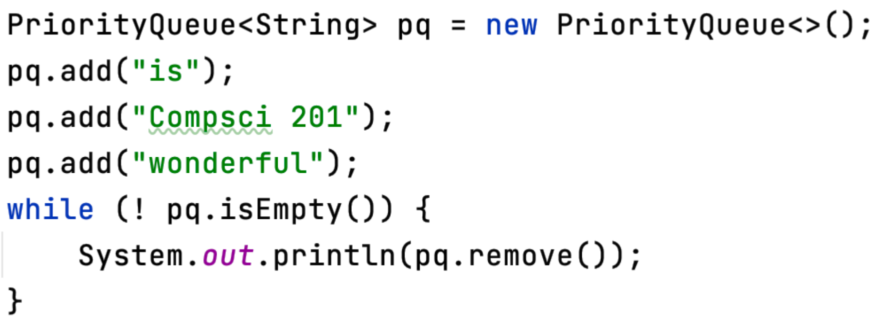
Dequeue removes from the front of the queue, which is always the highest priority item.

Queue sorted by priority instead of insertion order.

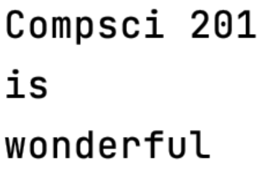
Zybook

java.util.PriorityQueue Class

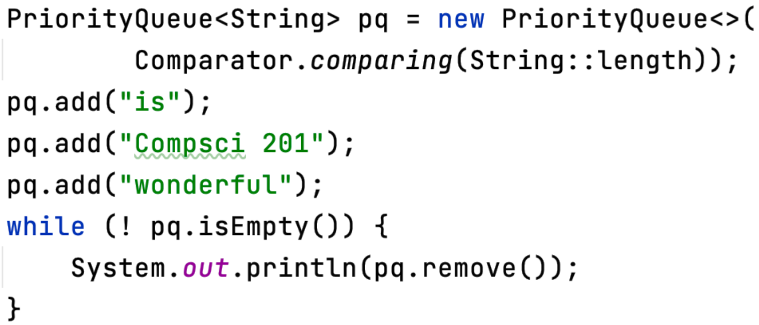
- Kept in sorted order, smallest out first
  - Objects must be Comparable OR provide Comparator to priority queue



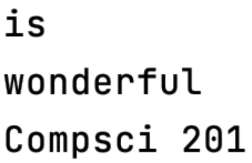
```java
PriorityQueue<String> pq = new PriorityQueue<>();
pq.add("is");
pq.add("Compsci 201");
pq.add("wonderful");
while (! pq.isEmpty()) {
    System.out.println(pq.remove());
}
```



```
Compsci 201
is
wonderful
```



```java
PriorityQueue<String> pq = new PriorityQueue<>(
        Comparator.comparing(String::length));
pq.add("is");
pq.add("Compsci 201");
pq.add("wonderful");
while (! pq.isEmpty()) {
    System.out.println(pq.remove());
}
```



```
is
wonderful
Compsci 201
```

Complexity of java Priority Queue

Method	Behavior	Runtime Complexity
`add(element)`	Add an element to the priority queue	O(log(N)) comparisons
`remove()`	Remove and return the minimal element	O(log(N)) comparisons
`peek()`	Return (do *not* remove) the minimal element	O(1)
`size()`	Return number of elements	O(1)

WOTO

Go to duke.is/g8smv

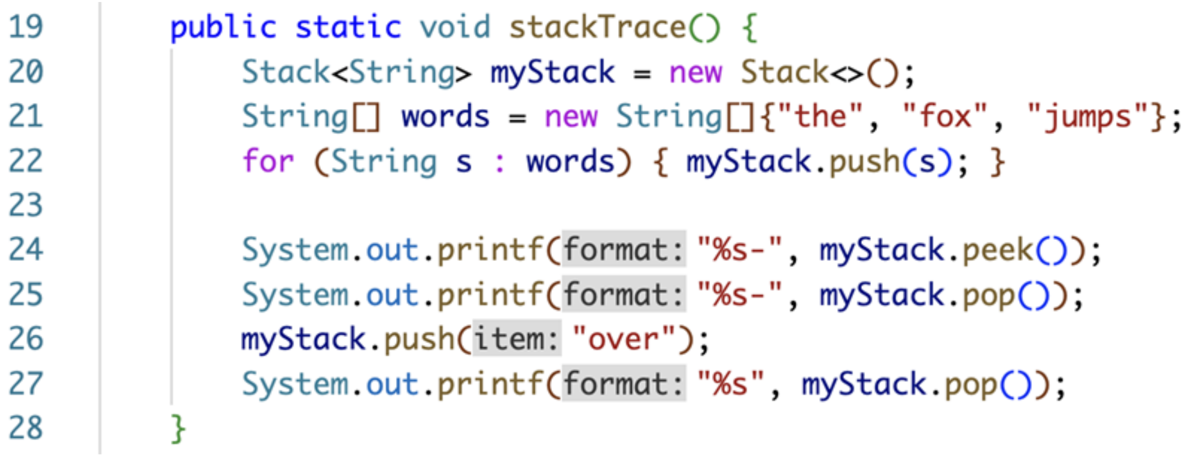
Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

2

What will be printed by the stackTrace method? Write your answer with no quotes and hyphens between words (as they would appear if printed as below). For example, you might write (though it would not be correct): the-fox-jumps. *

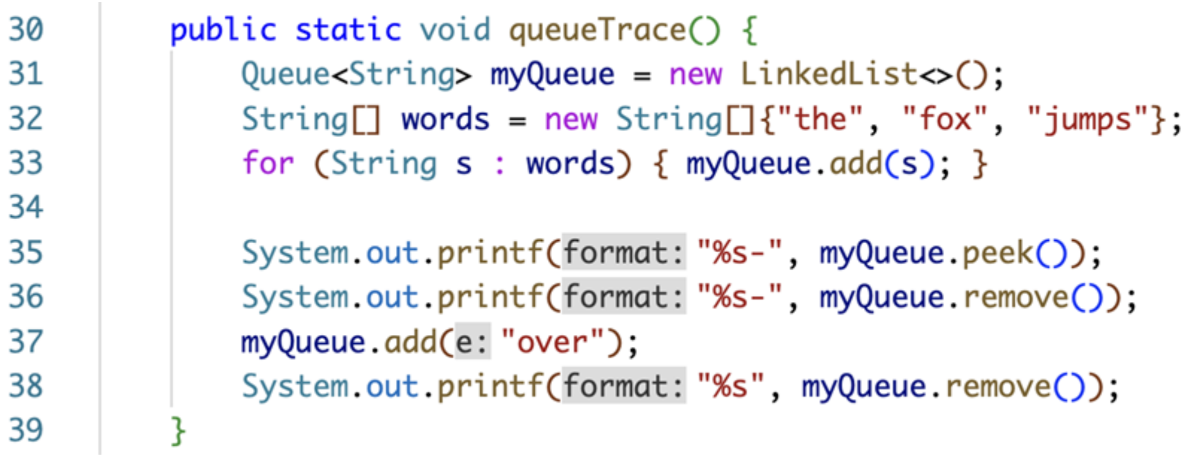


```
    public static void stackTrace() {
        Stack<String> myStack = new Stack<>();
        String[] words = new String[]{"the", "fox", "jumps"};
        for (String s : words) { myStack.push(s); }

        System.out.printf(format: "%s-", myStack.peek());
        System.out.printf(format: "%s-", myStack.pop());
        myStack.push(item: "over");
        System.out.printf(format: "%s", myStack.pop());
    }
```

3

What will be printed by the queueTrace method? Write your answer with no quotes and hyphens between words (as they would appear if printed as below). For example, you might write (though it would not be correct): the-fox-jumps. *

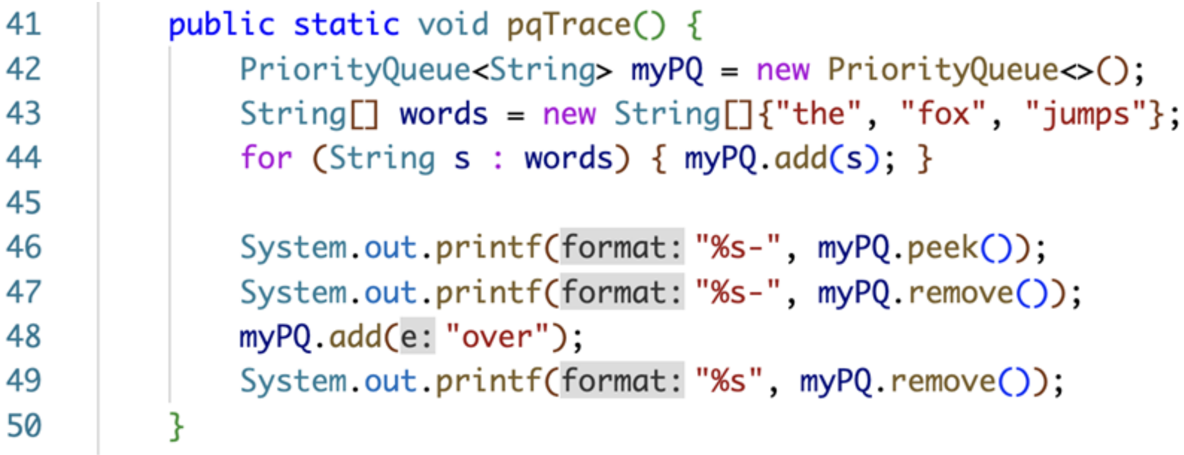


```
public static void queueTrace() {
    Queue<String> myQueue = new LinkedList<>();
    String[] words = new String[]{"the", "fox", "jumps"};
    for (String s : words) { myQueue.add(s); }

    System.out.printf(format: "%s-", myQueue.peek());
    System.out.printf(format: "%s-", myQueue.remove());
    myQueue.add(e: "over");
    System.out.printf(format: "%s", myQueue.remove());
}
```

4

What will be printed by the pqTrace method? Write your answer with no quotes and hyphens between words (as they would appear if printed as below). For example, you might write (though it would not be correct): the-fox-jumps. *

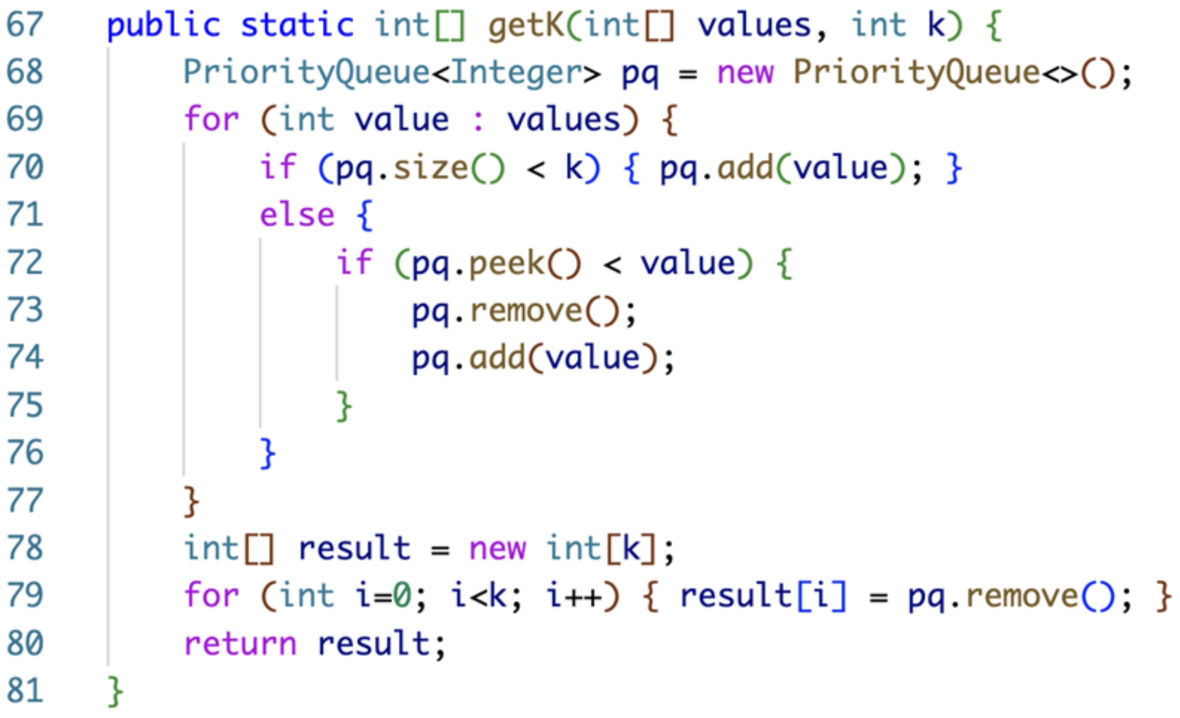


```java
    public static void pqTrace() {
        PriorityQueue<String> myPQ = new PriorityQueue<>();
        String[] words = new String[]{"the", "fox", "jumps"};
        for (String s : words) { myPQ.add(s); }

        System.out.printf(format: "%s-", myPQ.peek());
        System.out.printf(format: "%s-", myPQ.remove());
        myPQ.add(e: "over");
        System.out.printf(format: "%s", myPQ.remove());
    }
```

5

The getK method will return... *



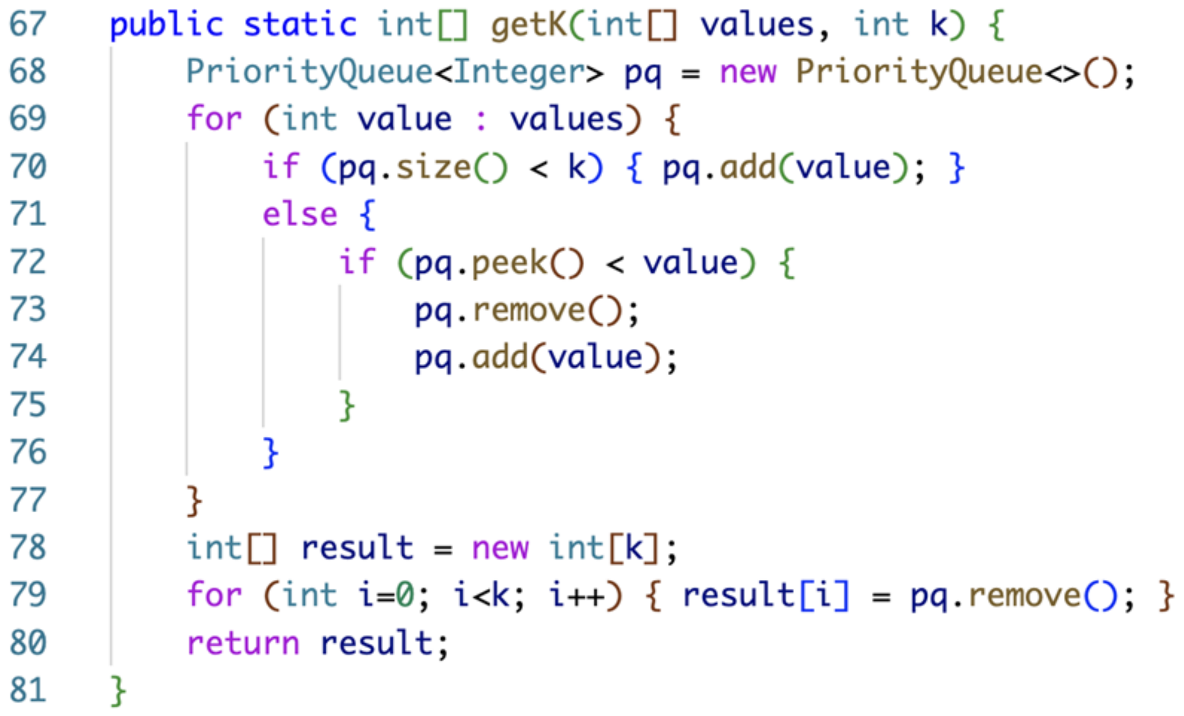
```
public static int[] getK(int[] values, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int value : values) {
        if (pq.size() < k) { pq.add(value); }
        else {
            if (pq.peek() < value) {
                pq.remove();
                pq.add(value);
            }
        }
    }
    int[] result = new int[k];
    for (int i=0; i<k; i++) { result[i] = pq.remove(); }
    return result;
}
```

○ The k **smallest** elements of values

○ The k **largest** elements of values

6

What is the asymptotic runtime complexity of the getK method as a function of N = values.length and k? *



```java
public static int[] getK(int[] values, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int value : values) {
        if (pq.size() < k) { pq.add(value); }
        else {
            if (pq.peek() < value) {
                pq.remove();
                pq.add(value);
            }
        }
    }
    int[] result = new int[k];
    for (int i=0; i<k; i++) { result[i] = pq.remove(); }
    return result;
}
```

○ O(k log(k))

○ O(k log(N))

○ O(N log(k))

○ O(N log(N))

Binary Trees

Comparing TreeSet/Map with HashSet/Map

TreeSet/Map

- O(log(N)) add, contains, put, get *not amortized*.
- Stored in sorted order
- Can get range of values in sorted order efficiently

HashSet/Map

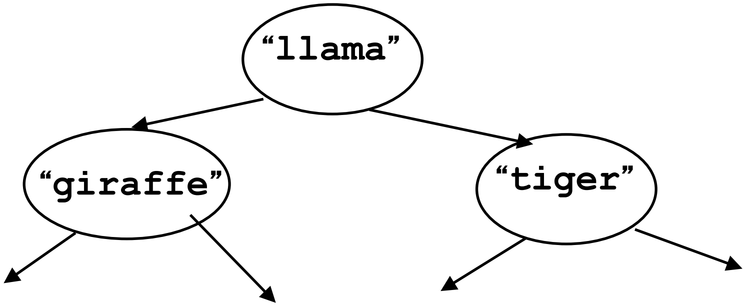
- O(1) add, contains, put, get, *amortized*.
- Unordered data structures
- Cannot get range efficiently, stored unordered

TreeNode to store Strings

Nodes for trees

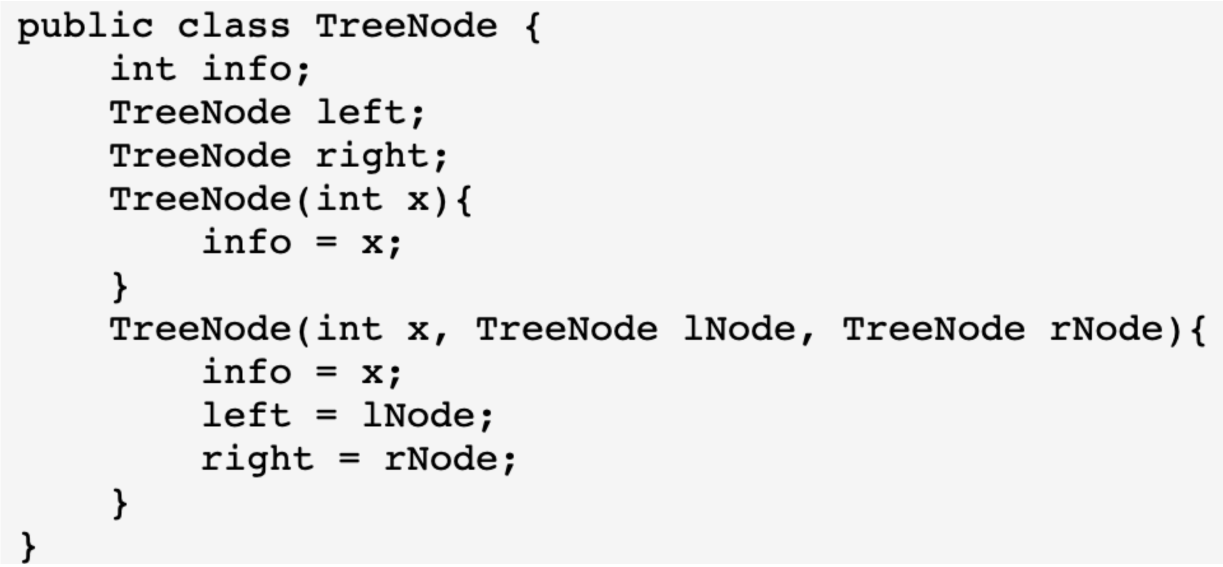
```
public class TreeNode {
    TreeNode left;
    TreeNode right;
    String info;
    TreeNode(String s,
             TreeNode llink, TreeNode rlink){
        info = s;
        left = llink;
        right = rlink;
    }
}
```

Like LinkedList but each node has 2 pointers instead of 1



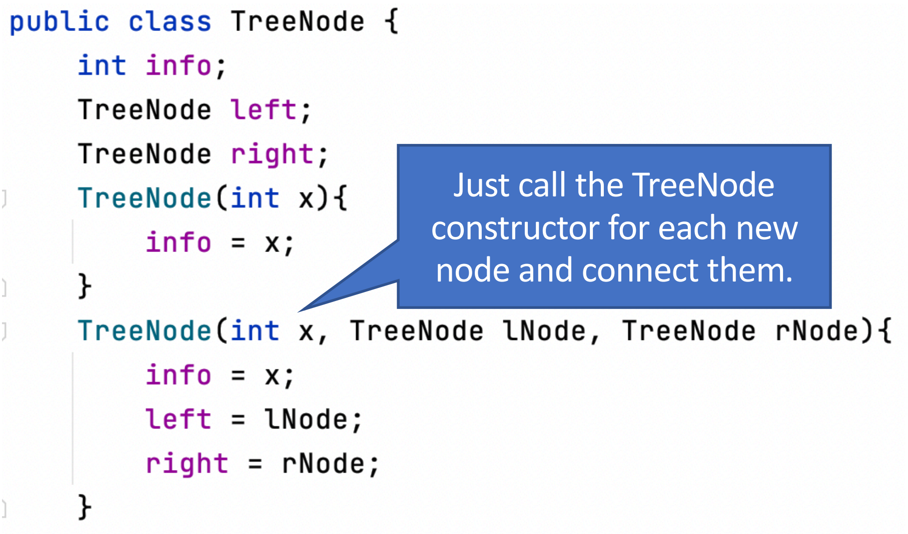
APT TreeNode to store ints

APT TreeNode will only hold integer. Would need to create another class to hold Strings? Another for…?



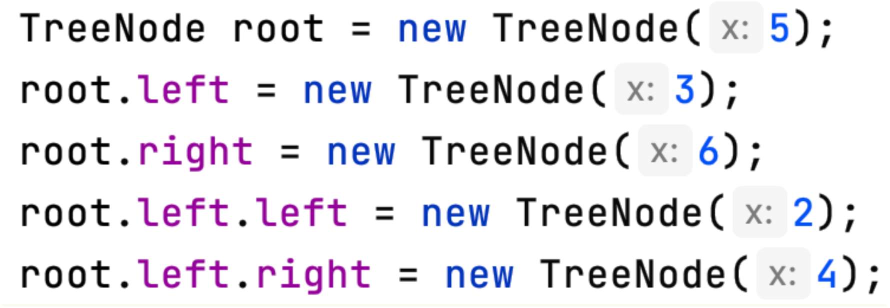
```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){
        info = x;
    }
    TreeNode(int x, TreeNode lNode, TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
}
```

FAQ: Making a tree with nodes?



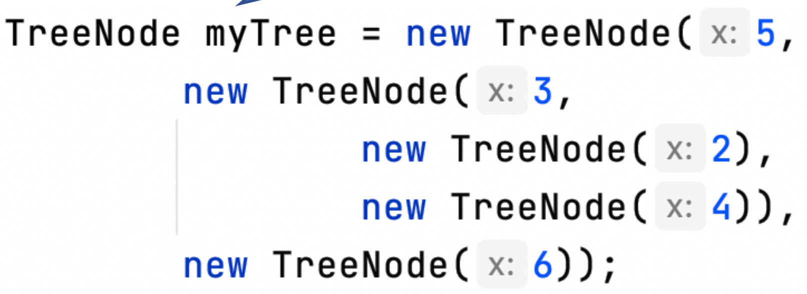
```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){
        info = x;
    }
    TreeNode(int x, TreeNode lNode, TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
```

Just call the TreeNode constructor for each new node and connect them.

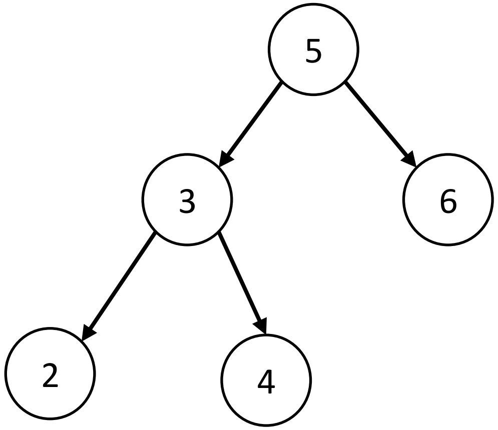


```
TreeNode root = new TreeNode( x: 5);
root.left = new TreeNode( x: 3);
root.right = new TreeNode( x: 6);
root.left.left = new TreeNode( x: 2);
root.left.right = new TreeNode( x: 4);
```

More terse version

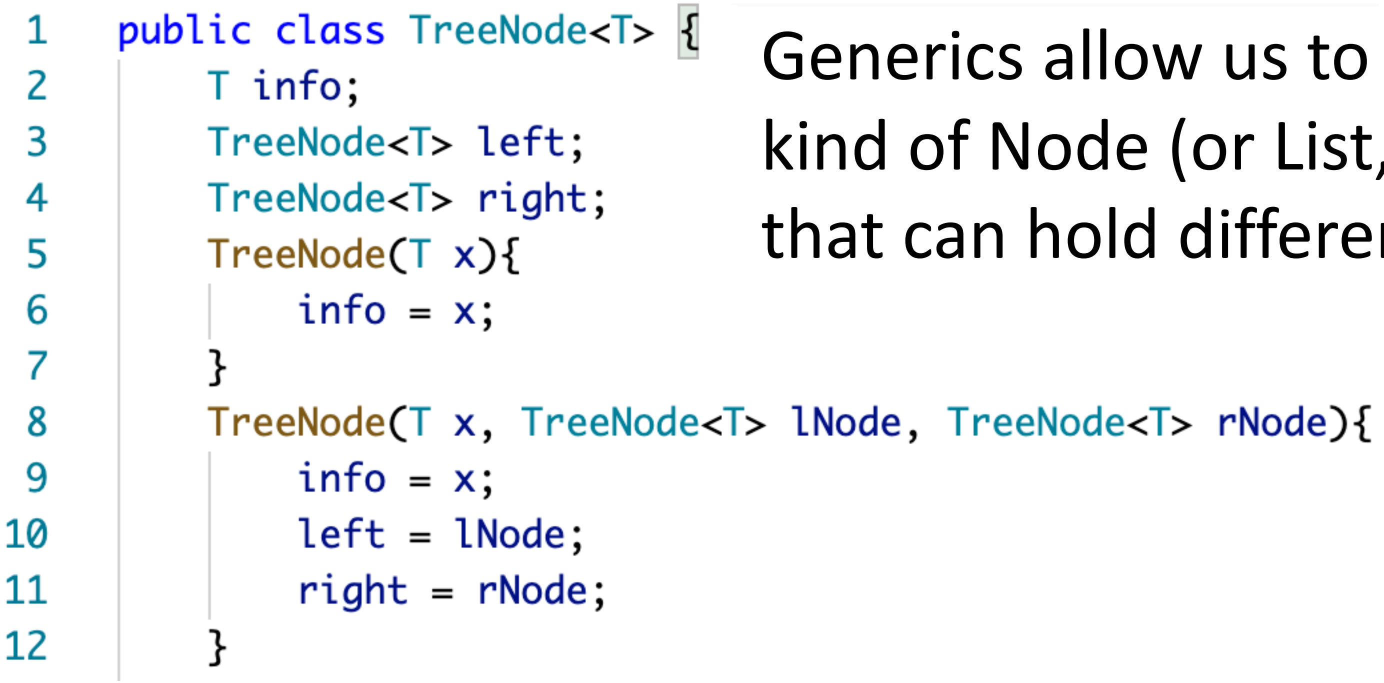


```
TreeNode myTree = new TreeNode( x: 5,
          new TreeNode( x: 3,
                    new TreeNode( x: 2),
                    new TreeNode( x: 4)),
          new TreeNode( x: 6));
```

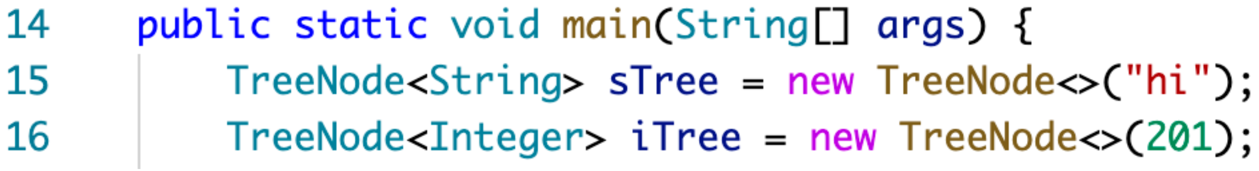


Aside: Generic TreeNode?

Generics allow us to write one kind of Node (or List, or Set, …) that can hold different types.



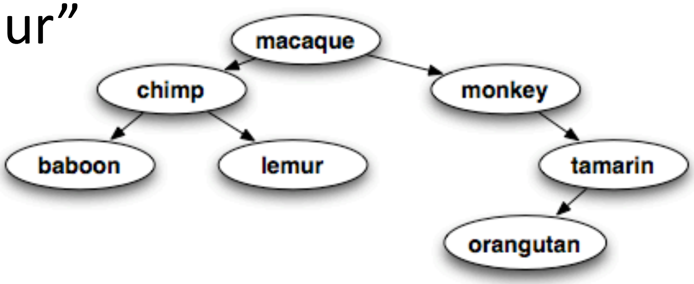
```java
public class TreeNode<T> {
    T info;
    TreeNode<T> left;
    TreeNode<T> right;
    TreeNode(T x){
        info = x;
    }
    TreeNode(T x, TreeNode<T> lNode, TreeNode<T> rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
```



```java
public static void main(String[] args) {
    TreeNode<String> sTree = new TreeNode<>("hi");
    TreeNode<Integer> iTree = new TreeNode<>(201);
```

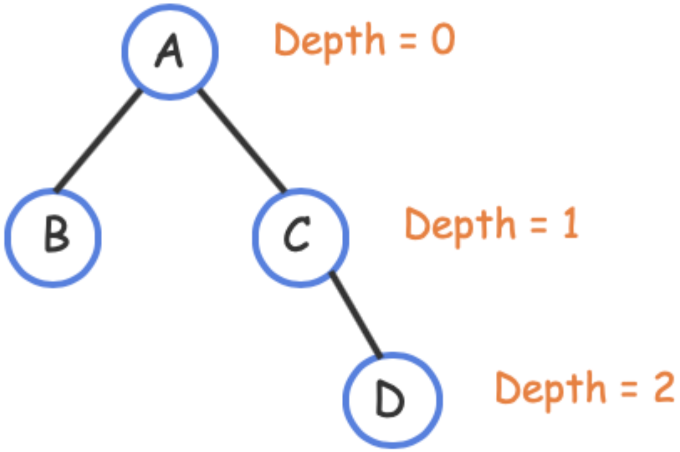
Tree terminology

- *Root*: "top node", has no parent, node you pass for the whole tree/subtree.
  - Example: Macaque
- *Leaf*: “bottom” nodes, have no children / both null
  - Example: Orangutan
- *Path*: sequence of parent-child nodes
  - Example: "macaque", "chimp", "lemur”
- *Subtree*: nodes at and beneath
  - ”chimp”, “baboon”, “lemur”



More Tree terminology

The **depth** of a node is the number of edges from the root to the node.



The **height** of a tree is the maximum depth of any node.

OR sometimes defined as maximum number of nodes on any root to leaf path = 1 + max depth.

inOrder Traversal

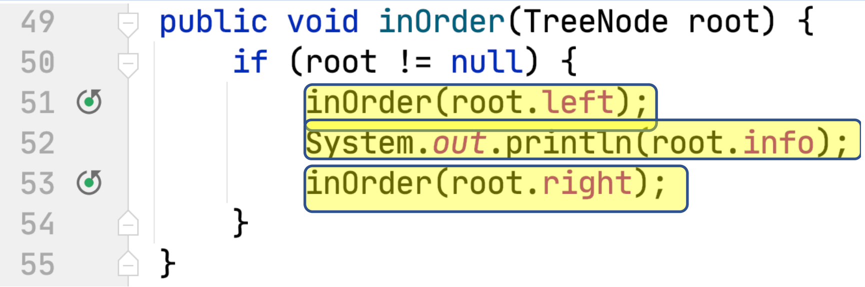
- How to “loop over” nodes in a tree? inOrder traversal and print
  - Search tree values printed in order
  - Could "visit" rather than print, every value

baboon, chimp, lemur

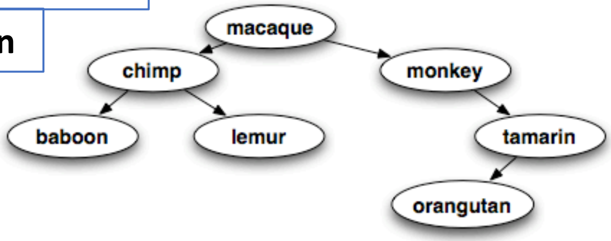
macaque

monkey, orangutan, tamarin

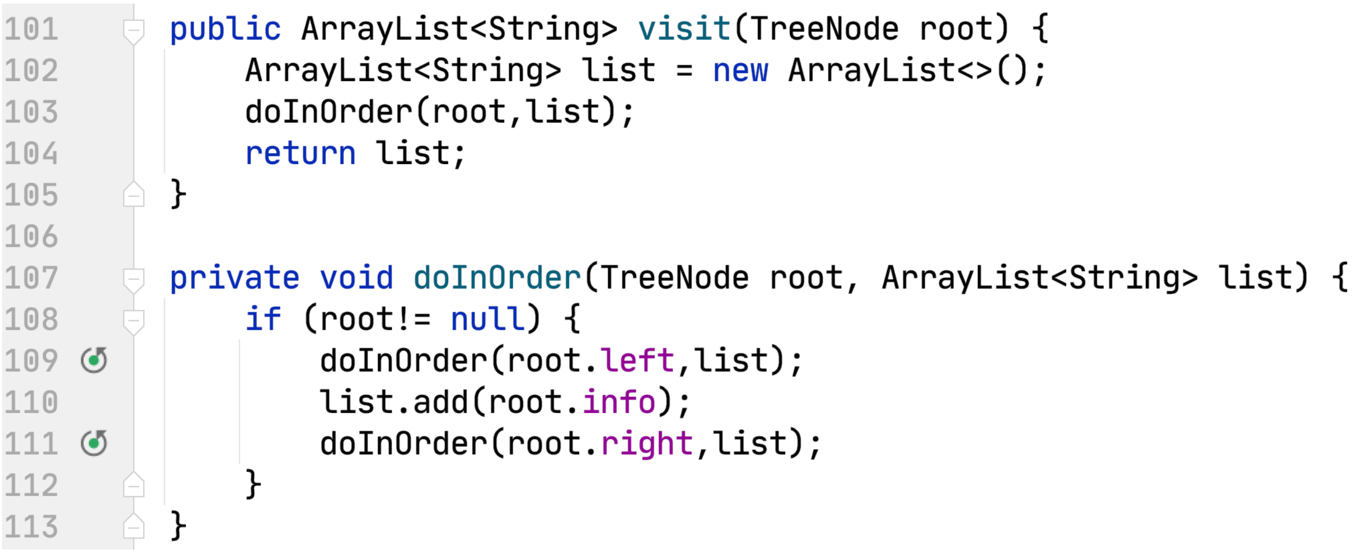
baboon, chimp, lemur, macaque, monkey, orangutan, tamarin



```java
public void inOrder(TreeNode root) {
    if (root != null) {
        inOrder(root.left);
        System.out.println(root.info);
        inOrder(root.right);
    }
}
```



Helper method to return List



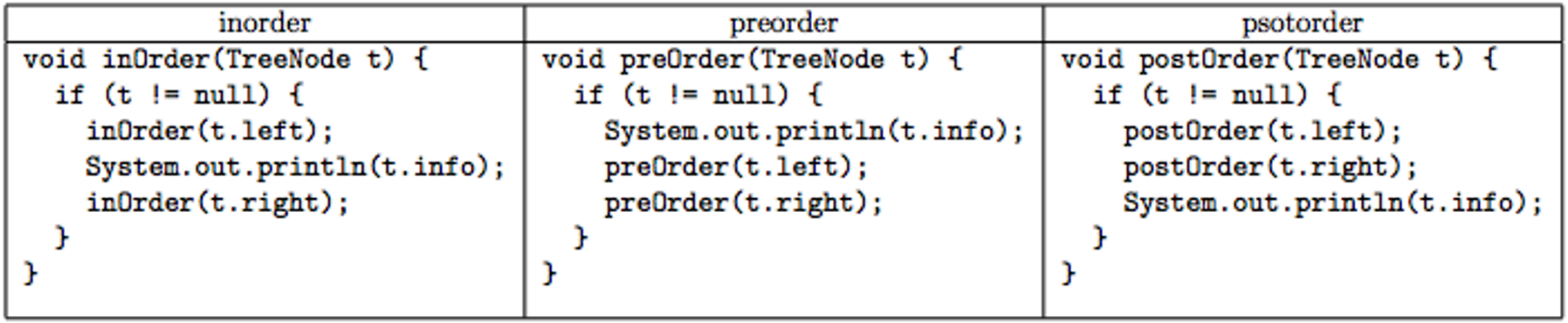
```java
public ArrayList<String> visit(TreeNode root) {
    ArrayList<String> list = new ArrayList<>();
    doInOrder(root,list);
    return list;
}

private void doInOrder(TreeNode root, ArrayList<String> list) {
    if (root!= null) {
        doInOrder(root.left,list);
        list.add(root.info);
        doInOrder(root.right,list);
    }
}
```

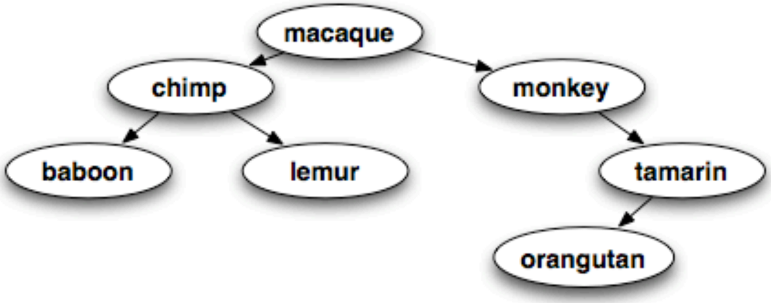
- In order traversal → list?
- Create list, call helper, return list
- values in returned list in order

Three ways to recursively traverse a tree

- Difference is in where the non-recursive part is

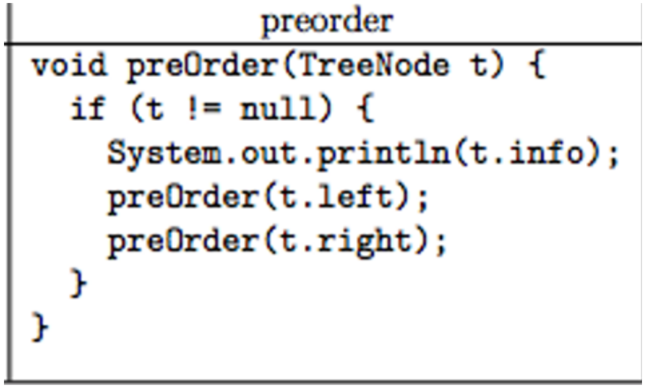


inorder	preorder	psotorder
`void inOrder(TreeNode t) {`	`void preOrder(TreeNode t) {`	`void postOrder(TreeNode t) {`
`  if (t != null) {`	`  if (t != null) {`	`  if (t != null) {`
`    inOrder(t.left);`	`    System.out.println(t.info);`	`    postOrder(t.left);`
`    System.out.println(t.info);`	`    preOrder(t.left);`	`    postOrder(t.right);`
`    inOrder(t.right);`	`    preOrder(t.right);`	`    System.out.println(t.info);`
`  }`	`  }`	`  }`
`}`	`}`	`}`



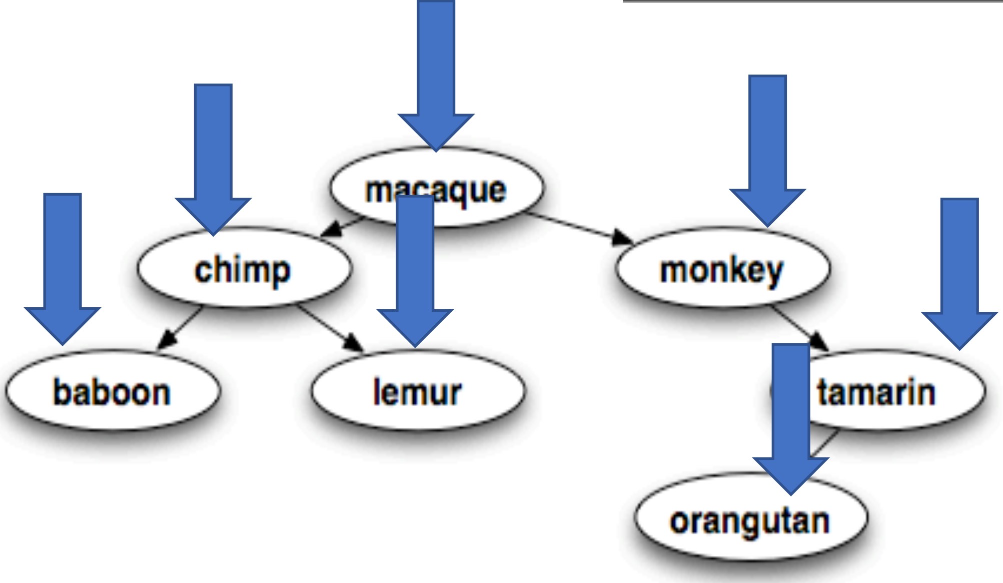
preOrder Traversal

- macaque
- chimp
- baboon
- lemur
- monkey
- tamarin
- orangutan

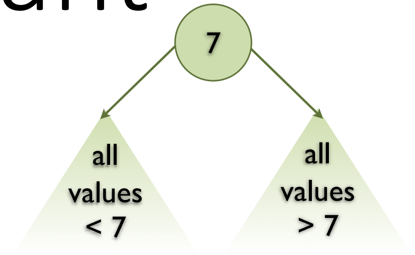


preorder

```
void preOrder(TreeNode t) {
  if (t != null) {
    System.out.println(t.info);
    preOrder(t.left);
    preOrder(t.right);
  }
}
```



Binary Search Tree Invariant



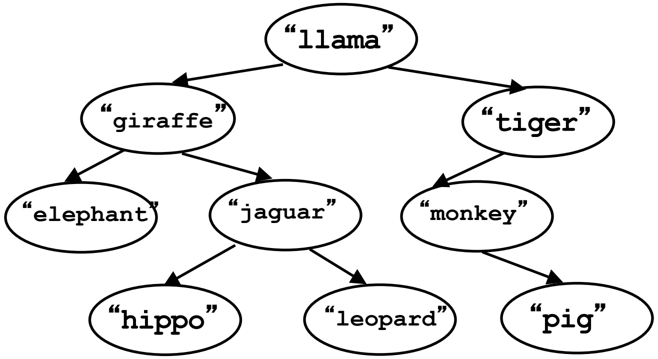
A binary tree is a binary **search** tree if *for every node*:

- Left subtree values are all less than the node’s value

AND

- Right subtree values are all greater than the node’s value

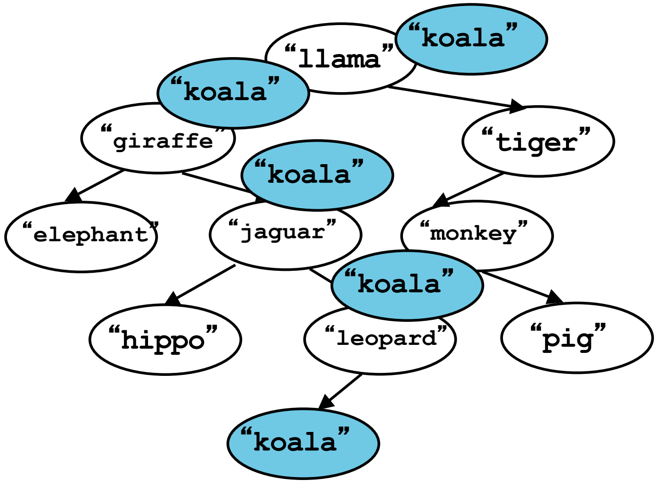
According to some ordering (comparable or comparator)

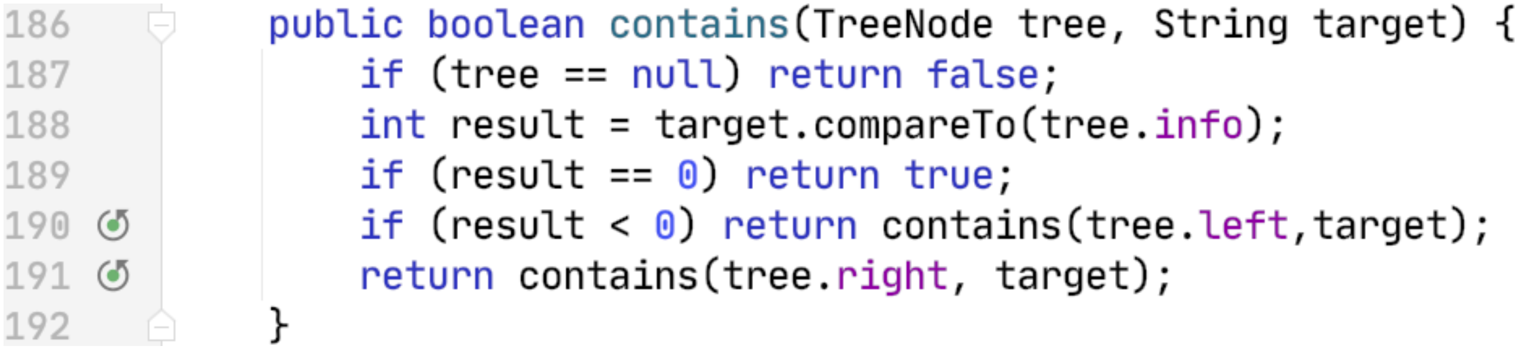


Enables efficient search, similar to binary search!

Recursive Search in Binary Search Tree

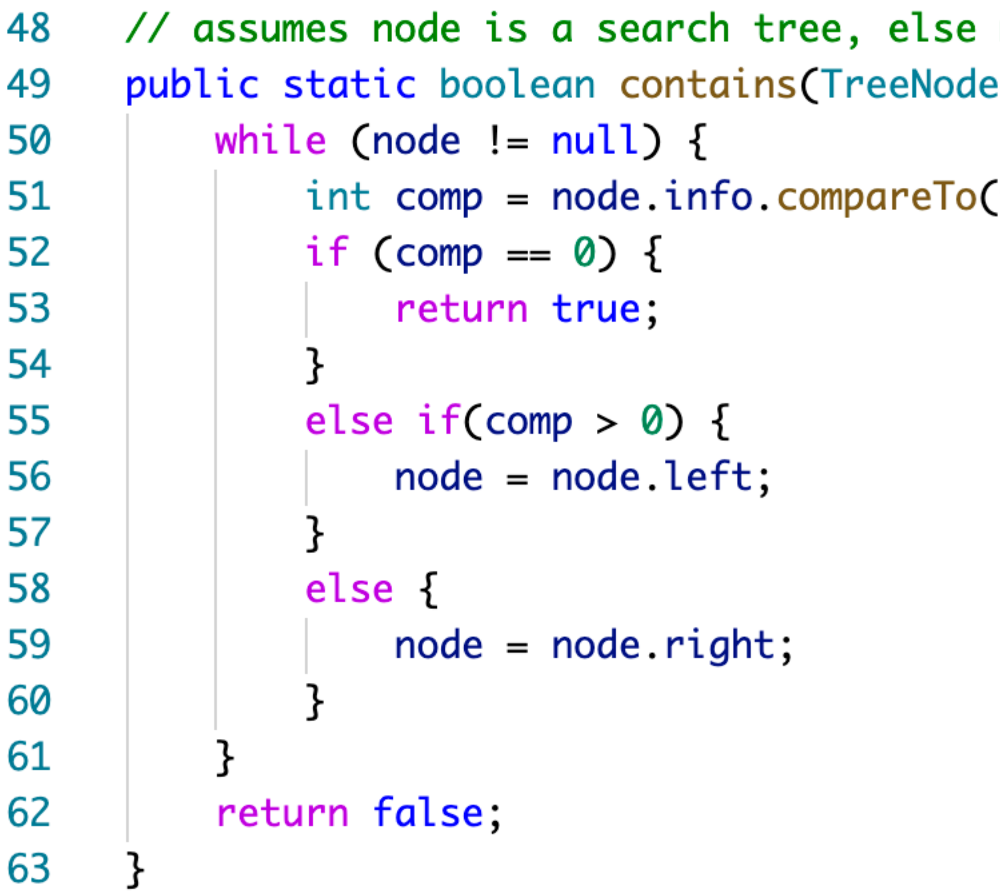
- Code for search
  - Insertion is very similar
  - **`target.compareTo(…)`**



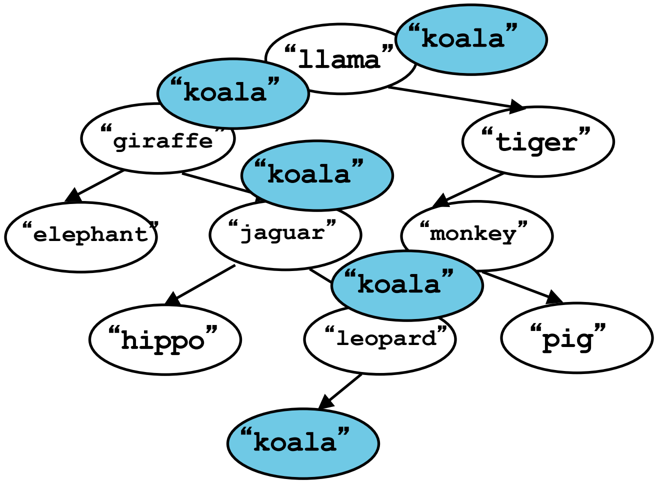


```java
public boolean contains(TreeNode tree, String target) {
    if (tree == null) return false;
    int result = target.compareTo(tree.info);
    if (result == 0) return true;
    if (result < 0) return contains(tree.left,target);
    return contains(tree.right, target);
}
```

Iterative search in binary search tree



```java
// assumes node is a search tree, else may return false negatives
public static boolean contains(TreeNode<String> node, String target) {
    while (node != null) {
        int comp = node.info.compareTo(target);
        if (comp == 0) {
            return true;
        }
        else if(comp > 0) {
            node = node.left;
        }
        else {
            node = node.right;
        }
    }
    return false;
}
```



Again, insertion is very similar