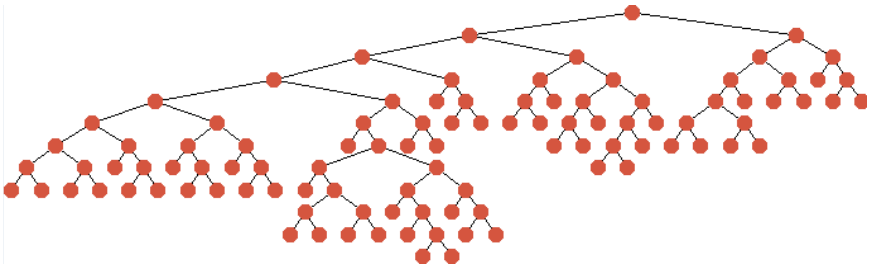


CSSE 230 Day 8

Binary Tree Iterators

After today, you should be able to…
… implement a simple iterator for trees
… implement _lazy_ iterators for trees

Questions?

Quiz question: What became clear to you as a result of class?

Another 230 student, not to be outdone:
Trees are unbeLEAFable fun when you can use recursion to traverse them, which helps you get to the ROOT of the problem.

Binary Tree Iterators

What if we want to iterate over the elements in the nodes of the tree one-at-a-time instead of just printing all of them?

What's an iterator?

- In Java, specified by **`java.util.Iterator<E>`**

`boolean`	**`hasNext`**`()` Returns `true` if the iteration has more elements.
`E`	**`next`**`()` Returns the next element in the iteration.
`void`	**`remove`**`()` Removes from the underlying collection the last element returned by the iterator (optional operation).

Using an Iterator

For any data structure that implements Iterable, (i.e., it defines the factory method iterator() which returns an iterator over the data) we can use the “foreach” syntax:

```
for (Integer val : iterableDataStruct) {
        ...
}
```

This is equivalent to:

```
for (Iterator<Integer> itr = iterableDataStruct.iterator();
                itr.hasNext();
                ) {
        Integer val = itr.next();
        ...
}
```

Using a Tree Iterator

Creating a tree iterator would allow us to traverse a tree iteratively (rather than recursively).

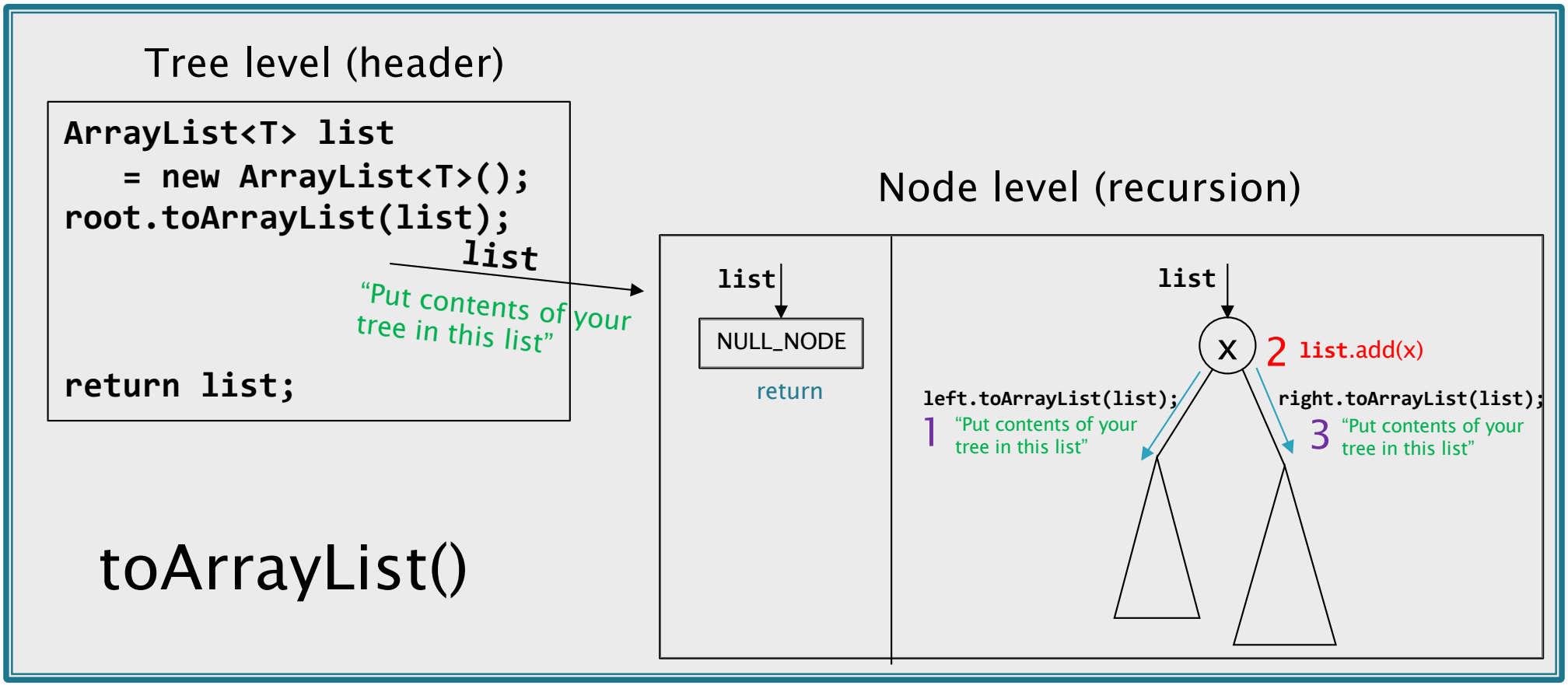
```
for (T item : binarySearchTree) {
        ...
}
```

We could have different iterators for different traversal orders.

```
Iterator<T> preOrderIt = new PreOrderIterator();
while (preOrderIt.hasNext()) {
        T item = preOrderIt.next();
        ...
}
```

Implement an (inefficient) inorder iterator using toArrayList().

- Pros: easy to write.
- Cons? We'll see shortly!



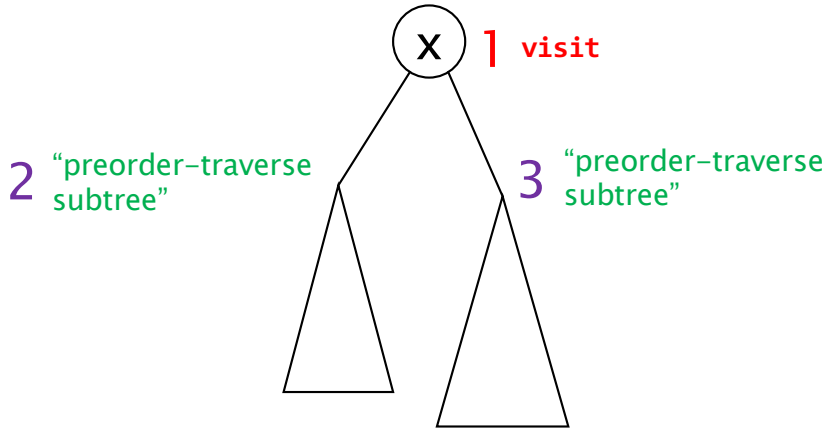
Why is the ArrayListIterator an inefficient iterator?

- Consider a tree with 1 million elements.
- What if we only end up iterating over the first 10 elements?

- To improve efficiency, the iterator should iterate on the tree itself.
  - Constructor should do minimal setup
  - On each .next() query, only do as much work as needed to respond & set up for future queries
  - In this context, laziness means efficiency!

Design an efficient preorder iterator

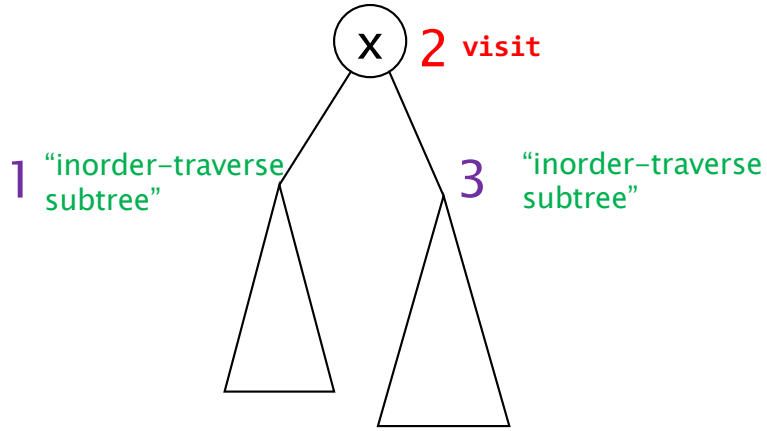
- Preorder: root, left, right



- Rather than carrying out all instructions at once, we should lazily handle them
- Store "tokens" representing pending instructions in a data structure (what data structure?)

Design an efficient inorder iterator

- Inorder: left, root, right



- Consider two types of instruction tokens:
  - 0: "traverse subtree":
  - 1: "visit node and traverse its right subtree"

Could represent tokens with, say, a compound class:

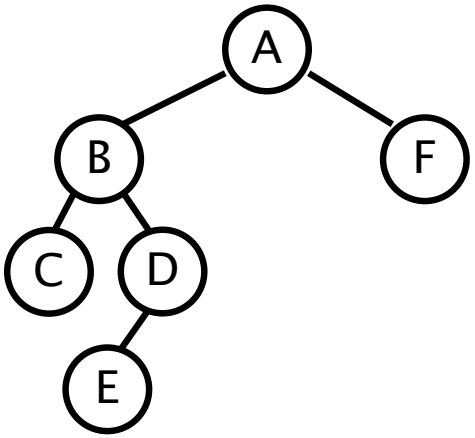
Use a `Stack<Token>` to store instructions

```
class Token {
        BinaryNode node;
        int tag;
}
```

- Loop: pop and either (0) push it and left, or (1) push right and return data

Design an efficient inorder iterator

- The idea of using instruction tokens extends nicely to postorder iterators too.
- If you just need an inorder iterator, an alternative is, whenever you see a node for the first time, to proactively push a chain of its left children.



Another Iterator

- What happens if we replace the Stack in the preorder iterator with a Queue?

Work time

Suggestion: work on Doublets
with your partner!