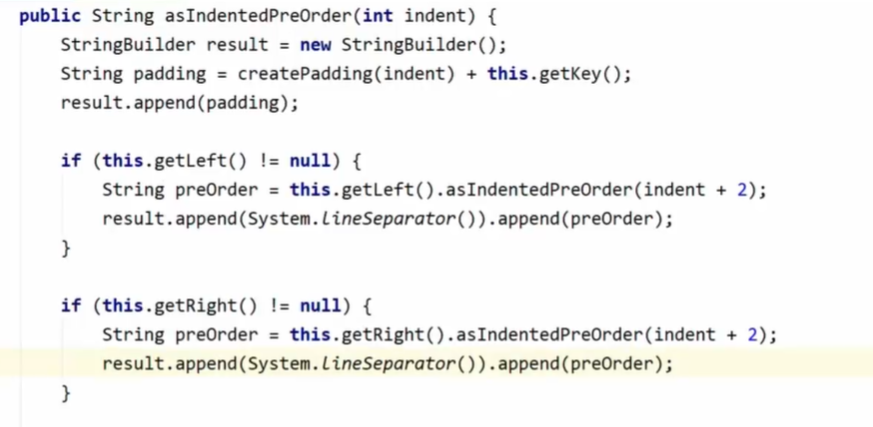
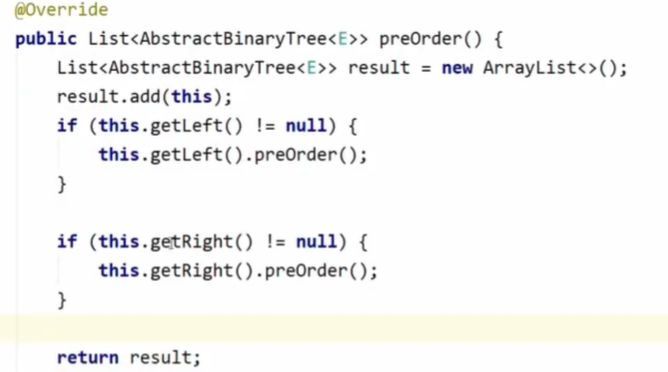
1. Binary Trees

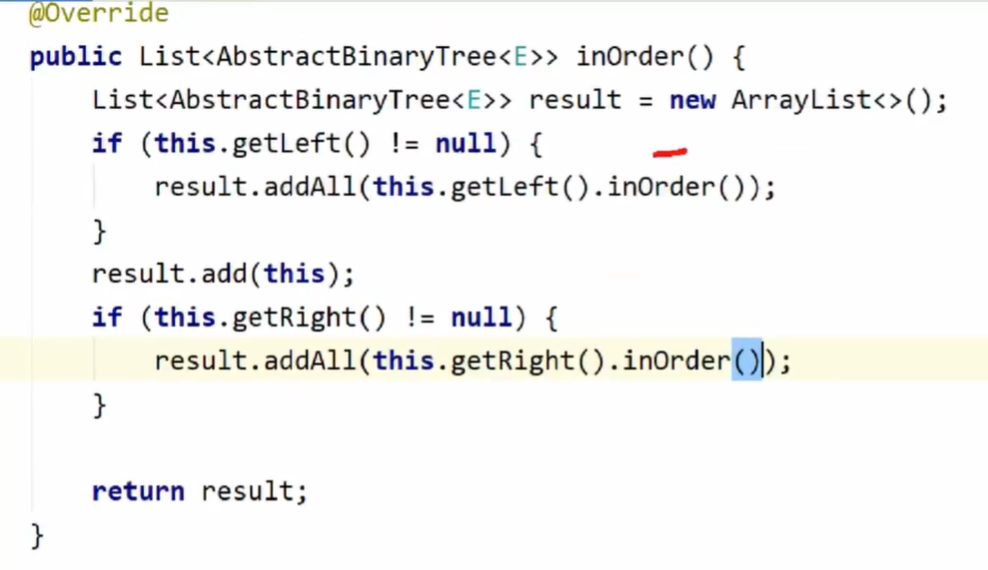
* Traverse with recursion



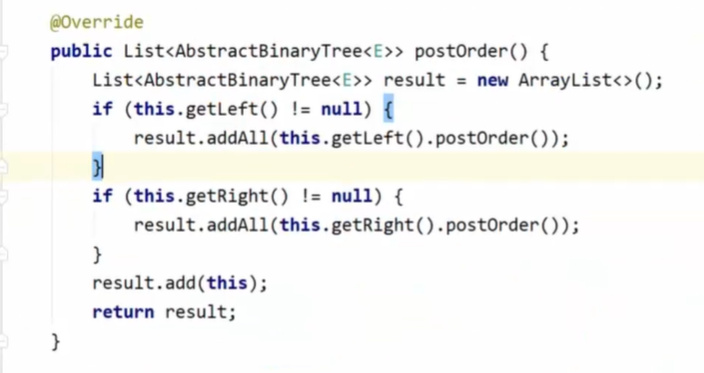
* Pre-order
* Starting from the root and printing the left child first, then the left child’s nodes
* Used for copying arrays



* In-order
* Sorting elements

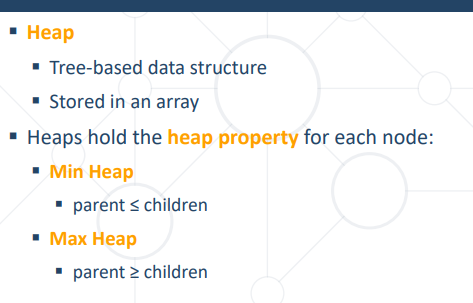


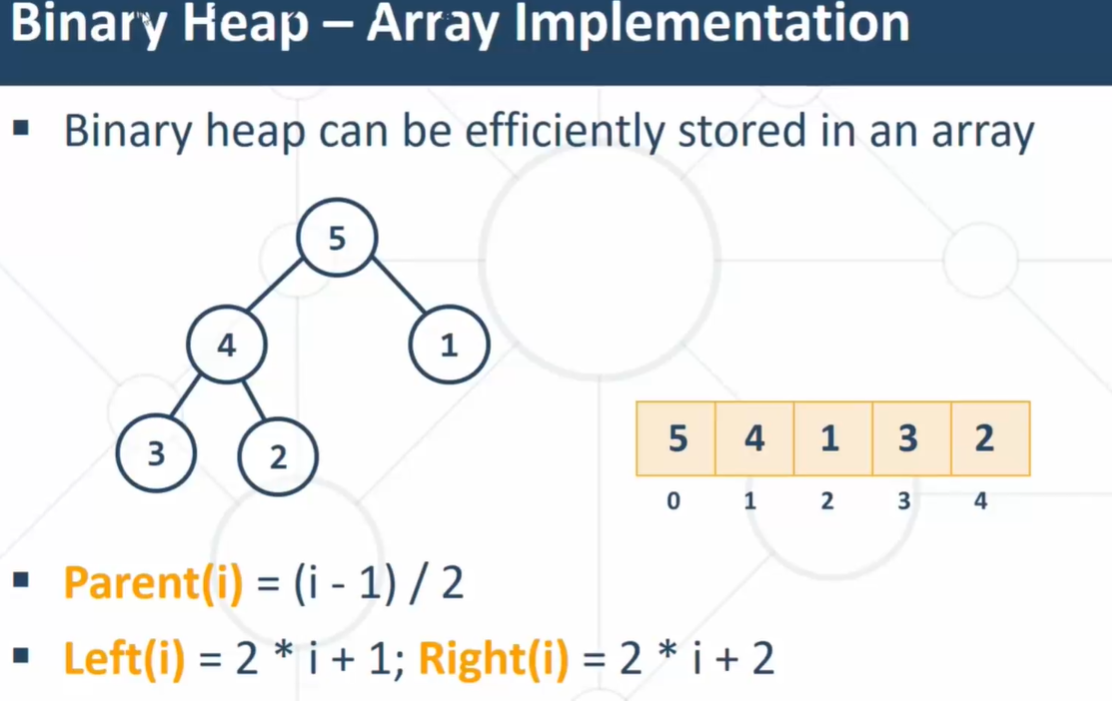
* Post-order
* Root is in the end of the result



1. Heaps

* Heaps are binary trees represented as an array
* It’s mandatory to be completed BST
* Max Heap and Min Heap
* Heapify up when parent < element (when max heap)
* Adding element is O(logN)

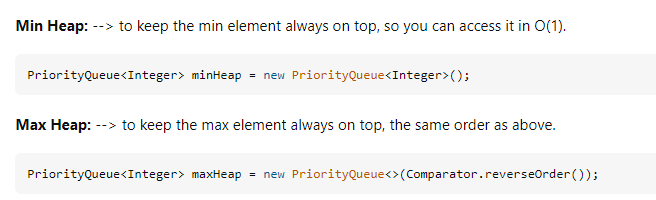




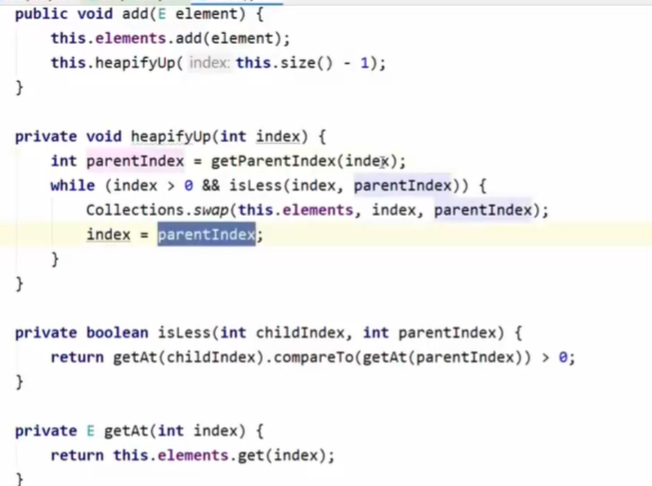
* To implement Heap, the class must extends Comparable.

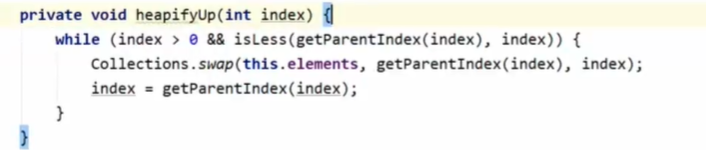


* Java implementation of heap
* Min heap – each parent’s value is less than its children. (or equal)
* Max heap – each parent’s value is bigger than its children. (or equal)

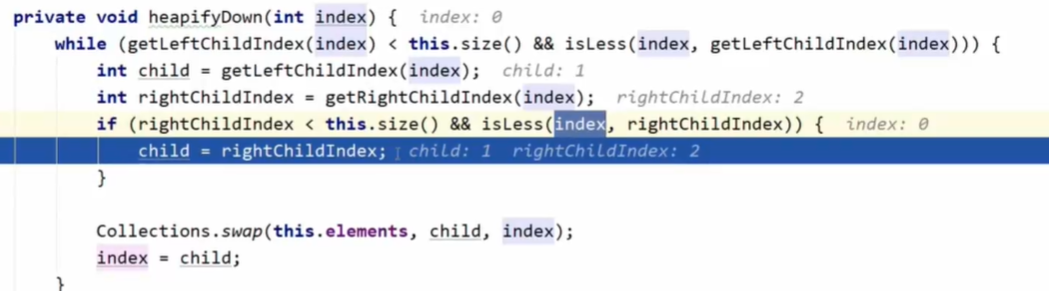


* Adding elements in a heap – heapifyUp
* Swap child with parent while the parent > child to keep the property



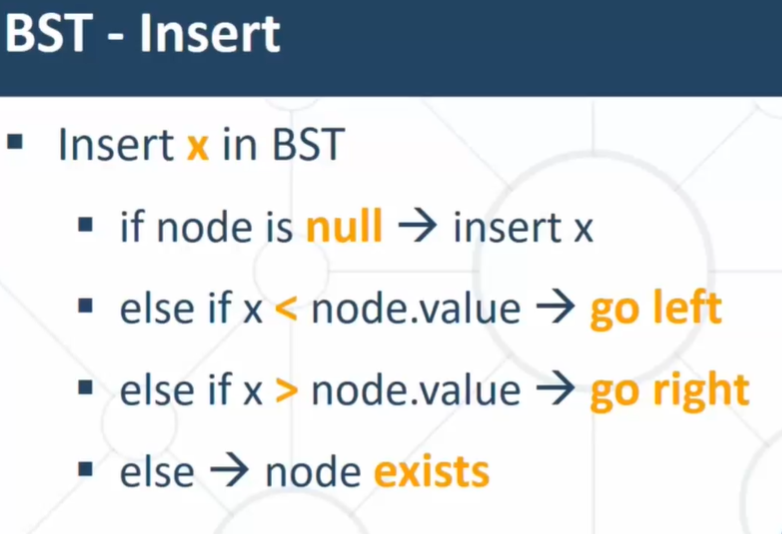


* Removing elements in a heap – heapifyDown
* Saving in a variable the root element
* Swap first and last
* While the parent < child -> heapify down the parent
* Left child < right child -> swap with the right child
* Left child > right child -> swap with the left child

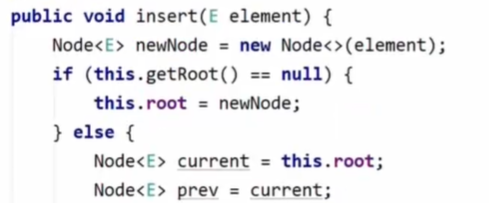


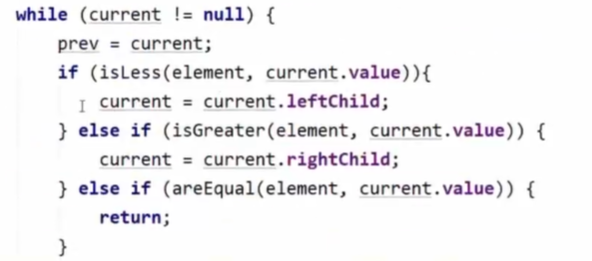
1. BST – Binary Search Tree

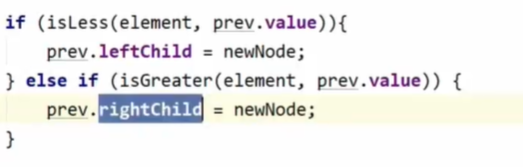
* Going left when the value is less than the parent and going right when the value is higher than the parent



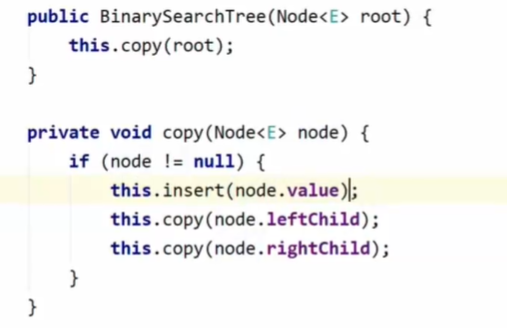
* Insert
* Iterating while the next node isn’t null and then check whether the element to add is lower or higher than the previous (parent)

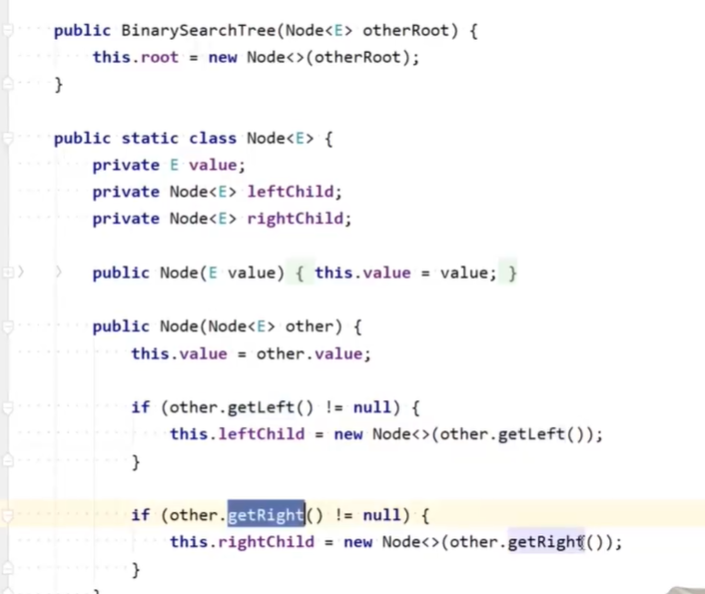




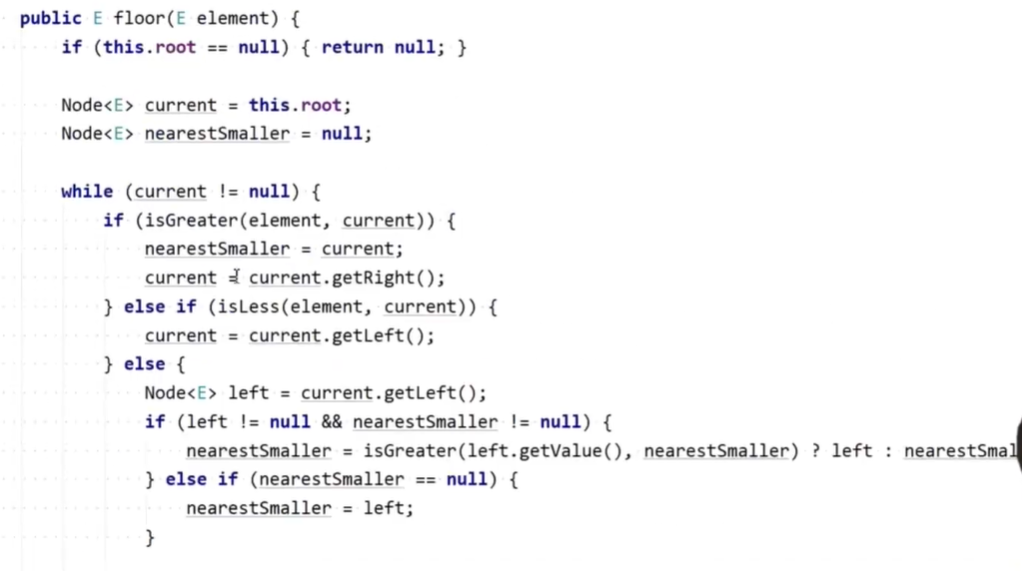


* Copy BST pre-order method
* For each left and right child -> insert the node’s value (E) .





* Floor – nearest smaller element





* BST Complexity
* Search – O(h) Или O(n)
* Insert – O(n)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

* Log(N) is achieved only with Balanced trees, because the height is Log(N)
* Not balanced trees are with complexity O(N)