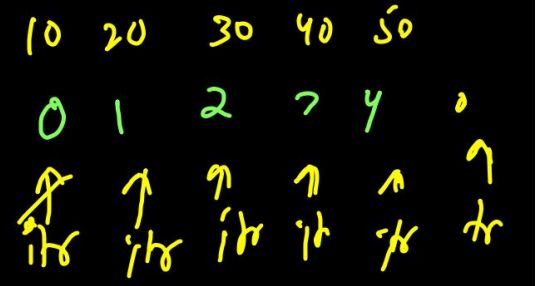
Forward Iteration

Iterable & Iterator



extended by

Collection

Interface

Collection HAS

A

iterator interface

reference variable

Syntatical Sugar

for each loop

```
for(object T: collection)
{

}
```

- next() :→ return current element & setup for next element (If no elements found, throws Exception)
- hasNext() :→ true if element(s) are remaining to be traversed otherwise false

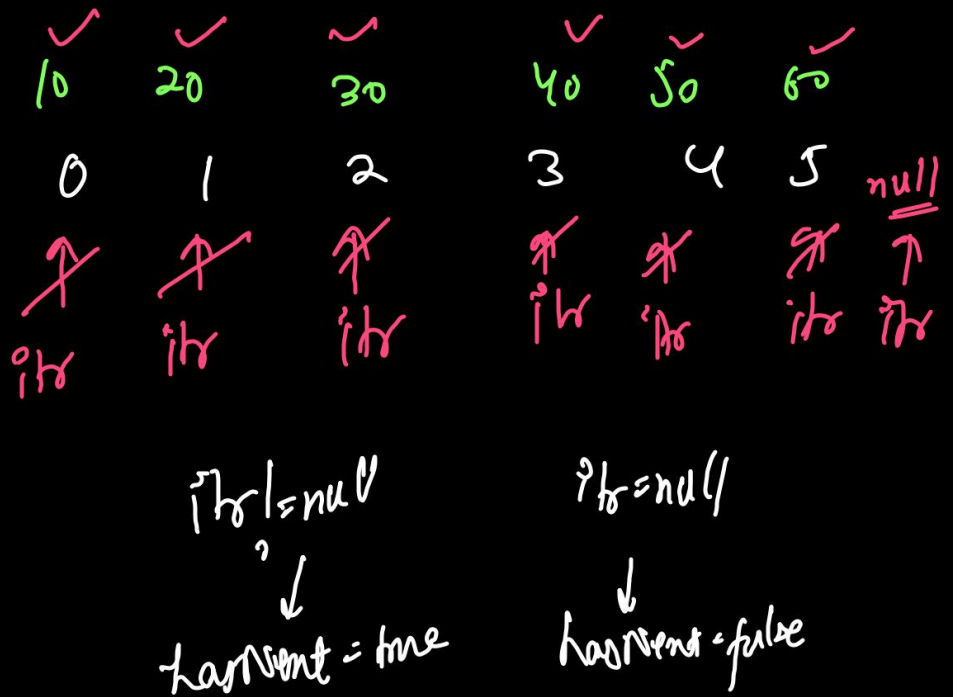
```java
ArrayList<Integer> arr = new ArrayList<>();
arr.add(e: 10);
arr.add(e: 20);
arr.add(e: 30);
arr.add(e: 40);
arr.add(e: 50);
arr.add(e: 60);

// Iterable : For Each Loop : Syntatical Sugar
for (Integer data : arr) {
    System.out.print(data + " ");
}
System.out.println();

// For Each Method (Java 8+ Feature)
arr.forEach((data) -> System.out.print(data + " "));
System.out.println();

// Iterator:
Iterator<Integer> itr = arr.iterator();
while (itr.hasNext() == true) {
    System.out.print(itr.next() + " ");
}
```

hasA relationship



```java
// Enumeration: Iterate on Vector and Stack
Vector<Integer> v = new Vector<>();
v.add(e: 10);
v.add(e: 20);
v.add(e: 30);
v.add(e: 40);
v.add(e: 50);
v.add(e: 60);

Enumeration<Integer> e = v.elements();
while (e.hasMoreElements() == true) {
    System.out.print(e.nextElement() + " ");
}
System.out.println();

// List Iterator
ListIterator<Integer> li = arr.listIterator();
while (li.hasNext() == true) {
    System.out.print(li.next() + " ");
}
System.out.println();

ListIterator<Integer> bi = arr.listIterator(arr.size());
while (bi.hasPrevious() == true) {
    System.out.print(bi.previous() + " ");
}
System.out.println();
```

```
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
60 50 40 30 20 10
```

→ nth element is not included!

Custom Iterators in Java

(1) Peeking Iterator LC 284

(2) Flatten Nested List Iterator LC 341

(3) BST Iterator - I LC 173

(4) BST Iterator - II Leetcode Locked - CodeStudio

(5) Two Sum in BST LC 653

Peeking Iterator

(1) Constructor

(2) peek()

(3) next()

(4) hasNext()

```java
class PeekingIterator implements Iterator<Integer> {
    Iterator<Integer> itr;
    Integer data;

    public PeekingIterator(Iterator<Integer> itr) {
        this.itr = itr;
        next();
    }

    public Integer peek() {
        return data;
    }

    @Override
    public Integer next() {
        Integer temp = data;

        if(itr.hasNext() == true){
            data = itr.next();
        } else {
            data = null;
        }

        return temp;
    }

    @Override
    public boolean hasNext() {
        return (data != null);
    }
}
```

peek() → 10

next() → 10, d=20, i=30

peek() → 20

next() → 20, d=30, i=40

peek() → 30

next() → 30, d=40, i=50

peek() → 40

next() → 40, d=50, i=60

peek → 50

next() → 50, d=60

hasNext()
↳ data != null

peek → 60

next() → 60, d=null

hasNext()
↳ data == null

```java
class PeekingIterator implements Iterator<Integer> {
    Iterator<Integer> itr;
    Integer data;

    public PeekingIterator(Iterator<Integer> itr) {
        this.itr = itr;
        next();
    }

    public Integer peek() {
        return data;
    }

    @Override
    public Integer next() {
        Integer temp = data;
        if(itr.hasNext() == true){
            data = itr.next();
        } else {
            data = null;
        }

        return temp;
    }

    @Override
    public boolean hasNext() {
        return (data != null);
    }
}
```

extra functionality

hasnext on collection

Iterator on collection

For List (AL, LL, Vector, Stack)
↳ peek, next, hasNext → O(1)

For Queue (ArrayDeque & PriorityQueue)
↳ peek, next, hasNext → O(1)

For Set (HashMap)
↳ peek, next, hasNext → Θ(1) avg

For Set (TreeSet)
↳ peek → O(1)
hasnext → O(1)
next → Θ(1) avg

Flatten Nested List Iterator

NestedList ⇒ List< NestedInteger >

[10, [20, 30, [40, 50, [ ]], 60], [70 [80 [90]]]]

↓ convert to 1D list of integers

[10, 20, 30, 40, 50, 60, 70, 80, 90]

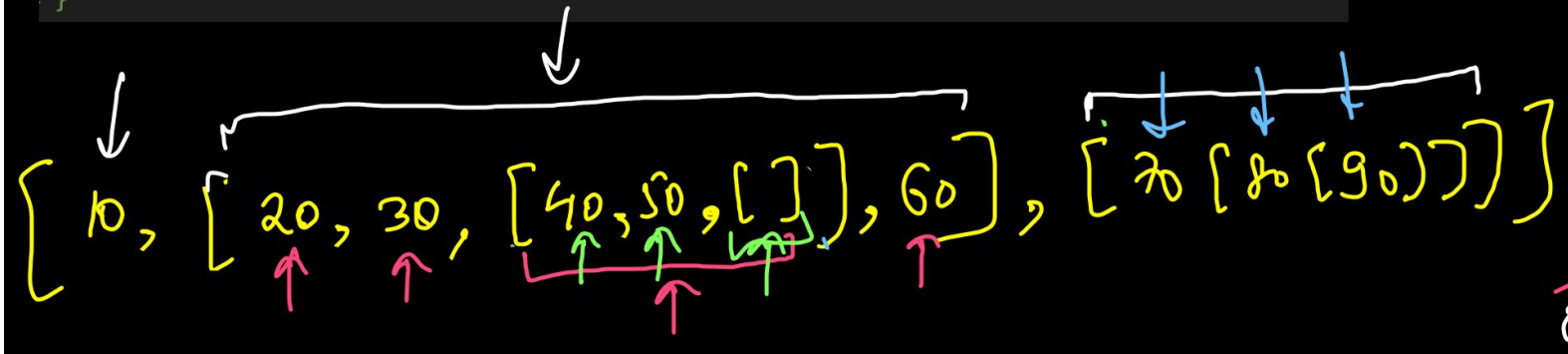
↓ iterator (inbuilt) next & hasNext

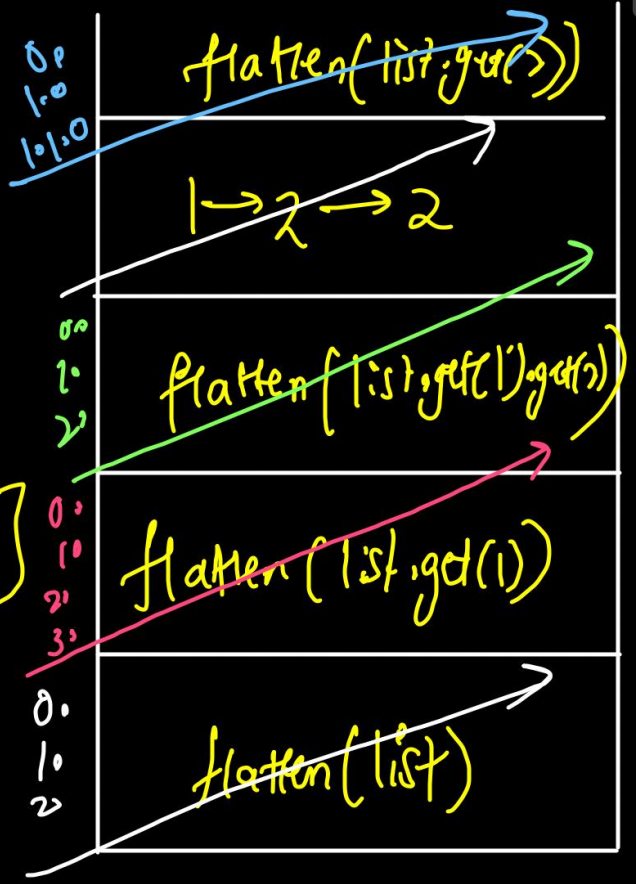
```java
public interface NestedInteger {

    // @return true if this NestedInteger holds a single integer, rather than a nested list.
    public boolean isInteger();

    // @return the single integer that this NestedInteger holds, if it holds a single integer
    // Return null if this NestedInteger holds a nested list
    public Integer getInteger();

    // @return the nested list that this NestedInteger holds, if it holds a nested list
    // Return empty list if this NestedInteger holds a single integer
    public List<NestedInteger> getList();
}
```





[10, 20, 30, 40, 50, 60, 70, 80, 90]

```java
public class NestedIterator implements Iterator<Integer> {
    List<Integer> arr;
    Iterator<Integer> itr;

    public NestedIterator(List<NestedInteger> nestedList) {
        arr = new ArrayList<>();
        flatten(nestedList);
        itr = arr.iterator();
    }

    public void flatten(List<NestedInteger> nestedList){
        for(NestedInteger data: nestedList){
            if(data.isInteger() == true){
                arr.add(data.getInteger());
            } else {
                flatten(data.getList());
            }
        }
    }

    @Override
    public Integer next() {
        return itr.next();
    }

    @Override
    public boolean hasNext() {
        return itr.hasNext();
    }
}
```

extra space flattened list → O(n) space

preprocessing

O(n) time

O(1) time on list

driver code

NestedIterator

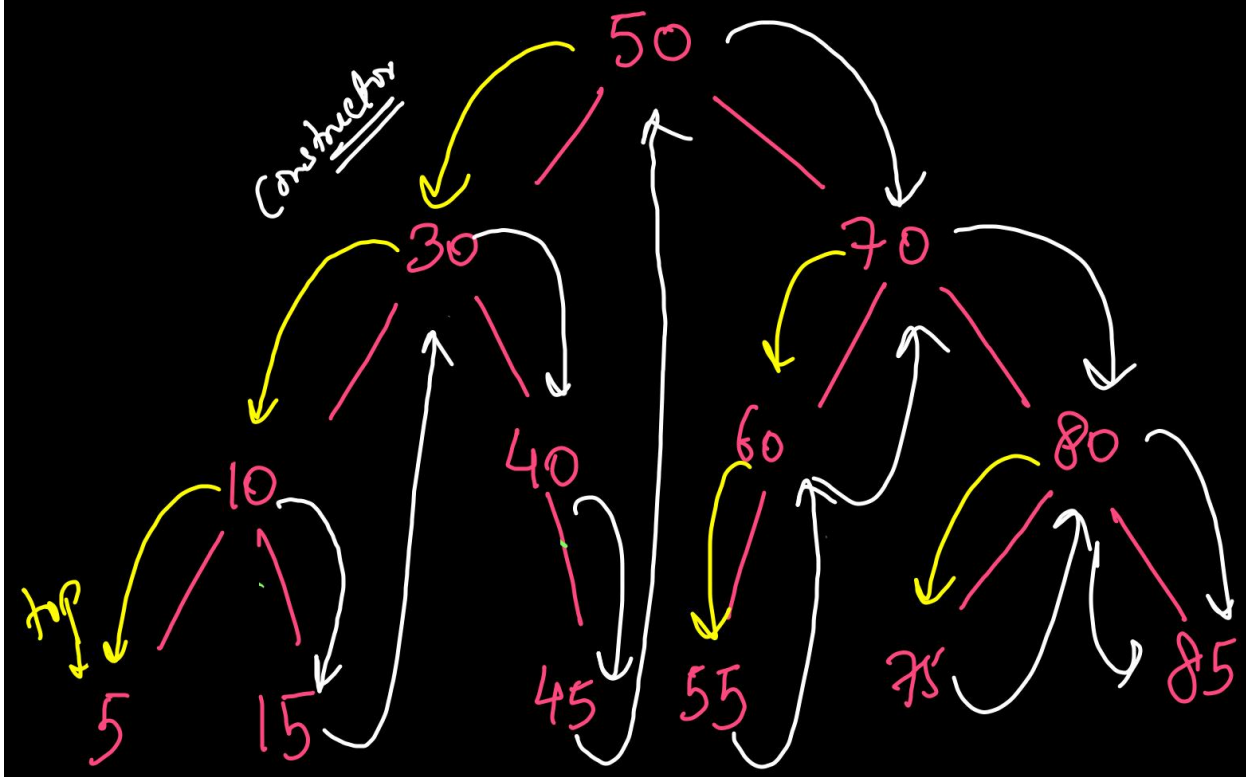
obj = new Nested(Iterator);

① Constructor call

while(obj.hasNext())

Syso(obj.next());

Binary Search Tree Iterator



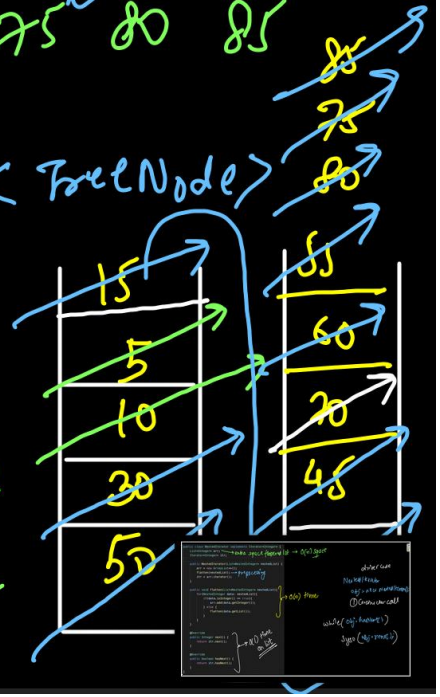
root

5 10 15 30 40

45 50 55 60

70 75 80 85

Stack<TreeNode>



stk.size() == 0 : false

stk.size() > 0 : true

```java
class BSTIterator{
    Stack<TreeNode> stk;

    public BSTIterator(TreeNode root) {
        stk = new Stack<>();
        inorderSucc(root);
    }

    public void inorderSucc(TreeNode curr){
        while(curr != null){
            stk.push(curr);
            curr = curr.left;
        }
    }

    public int next() {
        TreeNode curr = stk.pop();
        inorderSucc(curr.right);
        return curr.val;
    }

    public boolean hasNext() {
        return (stk.size() > 0);
    }
}
```

→ preprocessing → min^m node is not root
↳ go to leftmost node in BST (min^m node)

→ $O(\log_2 N) = O(h)$ in worst case

→ current ele return
→ next ele setup (ceil/next larger value of stk.top())

Asymptotic
↳ worst $O(\log_2 n)$

Amortized
↳ $O(1)$ avg case

→ $O(1)$

```java
// Single BST Node
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode() {
    }

    TreeNode(int val) {
        this.val = val;
    }
}
```

```java
// Collection of Nodes
class BinarySearchTree {
    private TreeNode root;

    public void insert(int val) {
        root = insert(root, val);
    }

    private TreeNode insert(TreeNode root, int val) {
        if (root == null)
            return new TreeNode(val);

        if (val < root.val)
            root.left = insert(root.left, val);

        else if (val > root.val)
            root.right = insert(root.right, val);

        return root;
    }
}
```

```java
public static void main(String[] args) {
    BinarySearchTree tree = new BinarySearchTree();
    tree.insert(val: 70);
    tree.insert(val: 50);
    tree.insert(val: 90);
    tree.insert(val: 30);
    tree.insert(val: 40);
    tree.insert(val: 80);
    tree.insert(val: 100);
}
```

Custom class
↳ Iterable & Iteral

BST class

```java
// Collection of Nodes
class BinarySearchTree implements Iterable<Integer> {
    TreeNode root;

    public void insert(int val) {
        root = insert(root, val);
    }

    private TreeNode insert(TreeNode root, int val) {
        if (root == null)
            return new TreeNode(val);

        if (val < root.val)
            root.left = insert(root.left, val);

        else if (val > root.val)
            root.right = insert(root.right, val);

        return root;
    }

    @Override
    public Iterator<Integer> iterator() {
        BSTIterator itr = new BSTIterator(root);
        return itr;
    }
}
```

for each method

Iterator class

```java
class BSTIterator implements Iterator<Integer> {
    Stack<TreeNode> stk;

    public BSTIterator(TreeNode root) {
        stk = new Stack<>();
        inorderSucc(root);
    }

    public void inorderSucc(TreeNode curr) {
        while (curr != null) {
            stk.push(curr);
            curr = curr.left;
        }
    }

    @Override
    public Integer next() {
        TreeNode curr = stk.pop();
        inorderSucc(curr.right);
        return curr.val;
    }

    @Override
    public boolean hasNext() {
        return (stk.size() > 0);
    }
}
```

driver code

```java
public class IteratorIterable {
    Run | Debug
    public static void main(String[] args) {
        BinarySearchTree tree = new BinarySearchTree();
        tree.insert(val: 70);
        tree.insert(val: 50);
        tree.insert(val: 90);
        tree.insert(val: 30);
        tree.insert(val: 40);
        tree.insert(val: 80);
        tree.insert(val: 100);

        // Iterable: For Each Loop
        for (Integer data : tree) {
            System.out.print(data + " ");
        }
        System.out.println();

        // Iterator
        BSTIterator itr = new BSTIterator(tree.root);
        while (itr.hasNext() == true) {
            System.out.print(itr.next() + " ");
        }
    }
}
```

root should not be private

Output (Sorted → Inorder)

30 40 50 70 80 90 100

```java
// Collection of Nodes
class BinarySearchTree implements Iterable<Integer> {
    private TreeNode root;

    public void insert(int val) {
        root = insert(root, val);
    }

    private TreeNode insert(TreeNode root, int val) {
        if (root == null)
            return new TreeNode(val);

        if (val < root.val)
            root.left = insert(root.left, val);

        else if (val > root.val)
            root.right = insert(root.right, val);

        return root;
    }

    @Override
    public Iterator<Integer> iterator() {
        BSTIterator itr = new BSTIterator(root);
        return itr;
    }
}
```

```java
public static void main(String[] args) {
    BinarySearchTree tree = new BinarySearchTree();
    tree.insert(val: 70);
    tree.insert(val: 50);
    tree.insert(val: 90);
    tree.insert(val: 30);
    tree.insert(val: 40);
    tree.insert(val: 80);
    tree.insert(val: 100);

    // Iterable: For Each Loop
    for (Integer data : tree) {
        System.out.print(data + " ");
    }
    System.out.println();

    // Iterator
    Iterator<Integer> itr = tree.iterator();
    while (itr.hasNext() == true) {
        System.out.print(itr.next() + " ");
    }
}
```

Collection HAS A Iterator

```java
static class ForwardIterator{
    Stack<TreeNode> stk;

    public ForwardIterator(TreeNode root) {
        stk = new Stack<>();
        inorderSucc(root);
    }

    public void inorderSucc(TreeNode curr){
        while(curr != null){
            stk.push(curr);
            curr = curr.left;
        }
    }

    public int peek(){
        if(hasNext() == false) return 0;
        return stk.peek().val;
    }

    public int next() {
        if(hasNext() == false) return 0;
        TreeNode curr = stk.pop();
        inorderSucc(curr.right);
        return curr.val;
    }

    public boolean hasNext() {
        return (stk.size() > 0);
    }
}
```

```java
static class BackWardIterator{
    Stack<TreeNode> stk;

    public BackWardIterator(TreeNode root) {
        stk = new Stack<>();
        inorderPred(root);
    }

    public void inorderPred(TreeNode curr){
        while(curr != null){
            stk.push(curr);
            curr = curr.right;
        }
    }

    public int peek(){
        if(hasPrev() == false) return 0;
        return stk.peek().val;
    }

    public int prev() {
        if(hasPrev() == false) return 0;
        TreeNode curr = stk.pop();
        inorderPred(curr.left);
        return curr.val;
    }

    public boolean hasPrev() {
        return (stk.size() > 0);
    }
}
```

Extra Space → $O(h) = O(\log_2 n)$

Time → $O(n/2 + n/2)$

$= O(n)$

Iterate on inorder

```java
public boolean findTarget(TreeNode root, int target) {
    if(root == null || (root.left == null && root.right == null)) return false;

    ForwardIterator left = new ForwardIterator(root);
    BackWardIterator right = new BackWardIterator(root);

    while(left.hasNext() == true && right.hasPrev() == true && left.peek() < right.peek()){
        if(left.peek() + right.peek() == target) return true;
        if(left.peek() + right.peek() < target) left.next();
        else right.prev();
    }

    return false;
}
```