Convert a given Binary Tree to Doubly Linked List

// Java program to convert a given Binary Tree to

// Doubly Linked List

/\* Structure for tree and Linked List \*/

class Node

{

    int data;

    Node left, right;

    public Node(int data)

    {

        this.data = data;

        left = right = null;

    }

}

class BinaryTree

{

    // 'root' - root of binary tree

    Node root;

    // 'head' - reference to head node of created

    //double linked list

    Node head;

    // A simple recursive function to convert a given

    // Binary tree to Doubly Linked List

    void BToDLL(Node root)

    {

        // Base cases

        if (root == null)

            return;

        // Recursively convert right subtree

        BToDLL(root.right);

        // insert root into DLL

        root.right = head;

        // Change left pointer of previous head

        if (head != null)

            (head).left = root;

        // Change head of Doubly linked list

        head = root;

        // Recursively convert left subtree

        BToDLL(root.left);

    }

    // Utility function for printing double linked list.

    void printList(Node head)

    {

        System.out.println("Extracted Double Linked List is : ");

        while (head != null)

        {

            System.out.print(head.data + " ");

            head = head.right;

        }

    }

    // Driver program to test the above functions

    public static void main(String[] args)

    {

        /\* Constructing below tree

               5

             /   \

            3     6

           / \     \

          1   4     8

         / \       / \

        0   2     7   9  \*/

        BinaryTree tree = new BinaryTree();

        tree.root = new Node(5);

        tree.root.left = new Node(3);

        tree.root.right = new Node(6);

        tree.root.left.right = new Node(4);

        tree.root.left.left = new Node(1);

        tree.root.right.right = new Node(8);

        tree.root.left.left.right = new Node(2);

        tree.root.left.left.left = new Node(0);

        tree.root.right.right.left = new Node(7);

        tree.root.right.right.right = new Node(9);

        tree.BToDLL(tree.root);

        tree.printList(tree.head);

    }

}

// This code has been contributed by Mayank Jaiswal(mayank\_24)

# Convert a Binary Tree to a Circular Doubly Link List

// Java Program to convert a Binary Tree to a

// Circular Doubly Linked List

// Node class represents a Node of a Tree

class Node

{

    int val;

    Node left,right;

    public Node(int val)

    {

        this.val = val;

        left = right = null;

    }

}

// A class to represent a tree

class Tree

{

    Node root;

    public Tree()

    {

        root = null;

    }

    // concatenate both the lists and returns the head

    // of the List

    public Node concatenate(Node leftList,Node rightList)

    {

        // If either of the list is empty, then

        // return the other list

        if (leftList == null)

            return rightList;

        if (rightList == null)

            return leftList;

        // Store the last Node of left List

        Node leftLast = leftList.left;

        // Store the last Node of right List

        Node rightLast = rightList.left;

        // Connect the last node of Left List

        // with the first Node of the right List

        leftLast.right = rightList;

        rightList.left = leftLast;

        // left of first node refers to

        // the last node in the list

        leftList.left = rightLast;

        // Right of last node refers to the first

        // node of the List

        rightLast.right = leftList;

        // Return the Head of the List

        return leftList;

    }

    // Method converts a tree to a circular

    // Link List and then returns the head

    // of the Link List

    public Node bTreeToCList(Node root)

    {

        if (root == null)

            return null;

        // Recursively convert left and right subtrees

        Node left = bTreeToCList(root.left);

        Node right = bTreeToCList(root.right);

        // Make a circular linked list of single node

        // (or root). To do so, make the right and

        // left pointers of this node point to itself

        root.left = root.right = root;

        // Step 1 (concatenate the left list with the list

        //         with single node, i.e., current node)

        // Step 2 (concatenate the returned list with the

        //         right List)

        return concatenate(concatenate(left, root), right);

    }

    // Display Circular Link List

    public void display(Node head)

    {

        System.out.println("Circular Linked List is :");

        Node itr = head;

        do

        {

            System.out.print(itr.val+ " " );

            itr = itr.right;

        }

        while (itr != head);

        System.out.println();

    }

}

// Driver Code

class Main

{

    public static void main(String args[])

    {

        // Build the tree

        Tree tree = new Tree();

        tree.root = new Node(10);

        tree.root.left = new Node(12);

        tree.root.right = new Node(15);

        tree.root.left.left = new Node(25);

        tree.root.left.right = new Node(30);

        tree.root.right.left = new Node(36);

        // head refers to the head of the Link List

        Node head = tree.bTreeToCList(tree.root);

        // Display the Circular LinkedList

        tree.display(head);

    }

}

|  |
| --- |
|  |
| **Program: How to check the given Binary Tree is Binary Search Tree (BST) or not?** |

public class IsBinarySearchTree {

    public boolean isBinarySearchTree(BstNode root) {

        if(root == null) return Boolean.TRUE;

        return isBstValid(root, Integer.MIN\_VALUE, Integer.MAX\_VALUE);

    }

    private boolean isBstValid(BstNode root, Integer minValue, Integer maxValue) {

        if(root == null) return Boolean.TRUE;

        if(root.getData() >= minValue && root.getData() < maxValue

                && isBstValid(root.getLeft(), minValue, root.getData())

                && isBstValid(root.getRight(), root.getData(), maxValue)) {

            return Boolean.TRUE;

        } else {

            return Boolean.FALSE;

        }

    }

}

public class BstNode {

    private BstNode left;

    private BstNode right;

    private Integer data;

    public BstNode(Integer data) {

        this.data = data;

    }

    public BstNode getLeft() {

        return left;

    }

    public void setLeft(BstNode left) {

        this.left = left;

    }

    public BstNode getRight() {

        return right;

    }

    public void setRight(BstNode right) {

        this.right = right;

    }

    public Integer getData() {

        return data;

    }

}

public static void main(String a[]) {

    BstNode root = new BstNode(3);

    // left sub tree

    BstNode node\_2 = new BstNode(2); root.setLeft(node\_2);

    BstNode node\_1 = new BstNode(1); node\_2.setLeft(node\_1);

    BstNode node\_4 = new BstNode(4); node\_2.setRight(node\_4);

    // right sub tree

    BstNode node\_6 = new BstNode(6); root.setRight(node\_6);

    BstNode node\_5 = new BstNode(5); node\_6.setLeft(node\_5);

    BstNode node\_7 = new BstNode(7); node\_6.setRight(node\_7);

    IsBinarySearchTree ibsTree = new IsBinarySearchTree();

    System.out.println(ibsTree.isBinarySearchTree(root));

}

# Sorted Array to Balanced BST

|  |
| --- |
| // Java program to print BST in given range    // A binary tree node  class Node {        int data;      Node left, right;        Node(int d) {          data = d;          left = right = null;      }  }    class BinaryTree {        static Node root;        /\* A function that constructs Balanced Binary Search Tree       from a sorted array \*/      Node sortedArrayToBST(int arr[], int start, int end) {            /\* Base Case \*/          if (start > end) {              return null;          }            /\* Get the middle element and make it root \*/          int mid = (start + end) / 2;          Node node = new Node(arr[mid]);            /\* Recursively construct the left subtree and make it           left child of root \*/          node.left = sortedArrayToBST(arr, start, mid - 1);            /\* Recursively construct the right subtree and make it           right child of root \*/          node.right = sortedArrayToBST(arr, mid + 1, end);            return node;      }        /\* A utility function to print preorder traversal of BST \*/      void preOrder(Node node) {          if (node == null) {              return;          }          System.out.print(node.data + " ");          preOrder(node.left);          preOrder(node.right);      }        public static void main(String[] args) {          BinaryTree tree = new BinaryTree();          int arr[] = new int[]{1, 2, 3, 4, 5, 6, 7};          int n = arr.length;          root = tree.sortedArrayToBST(arr, 0, n - 1);          System.out.println("Preorder traversal of constructed BST");          tree.preOrder(root);      }  }    // This code has been contributed by Mayank Jaiswal |

Run on IDE

Time Complexity: O(n)  
Following is the recurrance relation for sortedArrayToBST().

T(n) = 2T(n/2) + C

T(n) --> Time taken for an array of size n

C --> Constant (Finding middle of array and linking root to left

and right subtrees take constant time)

# Java Program to Reverse a String using Recursion

public class JavaExample {

public static void main(String[] args) {

String str = "Welcome to Beginnersbook";

String reversed = reverseString(str);

System.out.println("The reversed string is: " + reversed);

}

public static String reverseString(String str)

{

if (str.isEmpty())

return str;

//Calling Function Recursively

return reverseString(str.substring(1)) + str.charAt(0);

}

}

# Reverse a String in Java

public class ReverseString

{

    public static void main(String[] args)

    {

        System.out.println("Enter string to reverse:");

        Scanner read = new Scanner(System.in);

        String str = read.nextLine();

        String reverse = "";

        for(int i = str.length() - 1; i >= 0; i--)

        {

            reverse = reverse + str.charAt(i);

        }

        System.out.println("Reversed string is:");

        System.out.println(reverse);

    }

}

# Prime Number Program in Java

import java.util.Scanner;

class PrimeCheck

{

public static void main(String args[])

{

int temp;

boolean isPrime=true;

Scanner scan= new Scanner(System.in);

System.out.println("Enter any number:");

//capture the input in an integer

int num=scan.nextInt();

scan.close();

for(int i=2;i<=num/2;i++)

{

temp=num%i;

if(temp==0)

{

isPrime=false;

break;

}

}

//If isPrime is true then the number is prime else not

if(isPrime)

System.out.println(num + " is a Prime Number");

else

System.out.println(num + " is not a Prime Number");

}

}

## Reverse a Number using a while loop in Java

public class ReverseNumber {

public static void main(String[] args) {

int num = 1234, reversed = 0;

while(num != 0) {

int digit = num % 10;

reversed = reversed \* 10 + digit;

num /= 10;

}

System.out.println("Reversed Number: " + reversed);

}

}

### Reverse a number using recursion

import java.util.Scanner;

class RecursionReverseDemo

{

//A method for reverse

public static void reverseMethod(int number) {

if (number < 10) {

System.out.println(number);

return;

}

else {

System.out.print(number % 10);

//Method is calling itself: recursion

reverseMethod(number/10);

}

}

public static void main(String args[])

{

int num=0;

System.out.println("Input your number and press enter: ");

Scanner in = new Scanner(System.in);

num = in.nextInt();

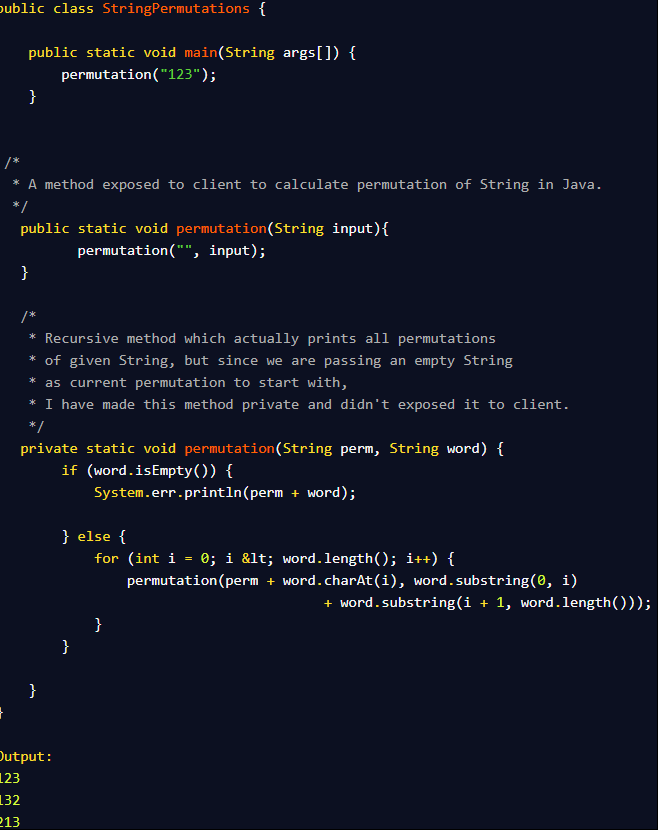
System.out.print("Reverse of the input number is:");

reverseMethod(num);

System.out.println();

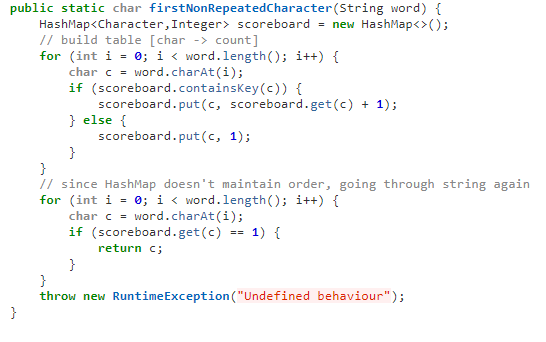
}

}

How to Find All Permutations of String in Java using Recursion  
  


Java Program to find duplicate words in String  
  


3 ways to Find First Non Repeated Character in a String - Java Programming Problem  
  

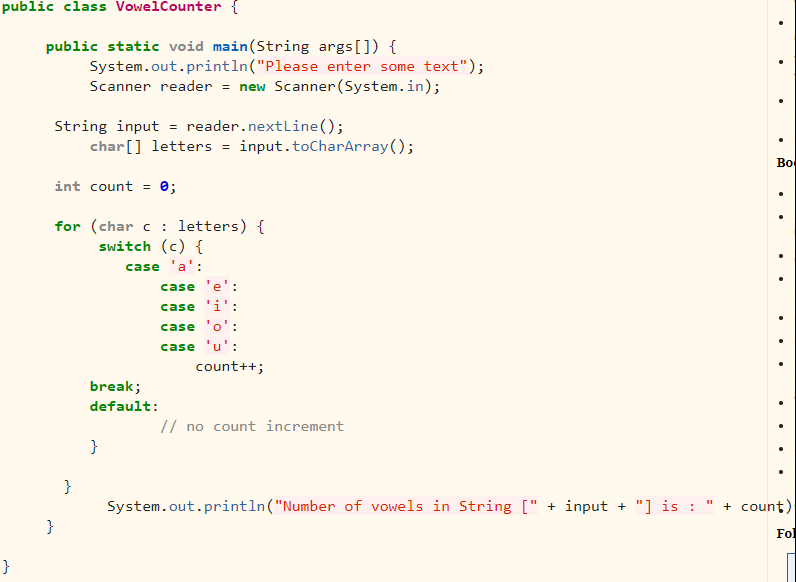
How to Count Occurrences of a Character in String - Java Programming Exercise Example

**import** org.springframework.util.StringUtils;

/\*\*  
 \* Java program to count the number of occurrence of any character on String.  
 \* @author Javin Paul  
 \*/  
**public** **class** CountCharacters {  
  
    **public** **static** **void** main(**String** args[]) {  
            
        **String** input = "Today is Monday"; *//count number of "a" on this String.*  
        
        *//Using Spring framework StringUtils class for finding occurrence of another String*  
        **int** count = StringUtils.countOccurrencesOf(input, "a");  
        
        **System**.out.println("count of occurrence of character 'a' on String: " +

                " Today is Monday' using Spring StringUtils " + count);  
  
        
        *//Using Apache commons lang StringUtils class*  
        **int** number = org.apache.commons.lang.StringUtils.countMatches(input, "a");  
        **System**.out.println("count of character 'a' on String: 'Today is Monday' using commons StringUtils " + number);  
        
        *//counting occurrence of character with loop*  
        **int** charCount = 0;  
        for(**int** i =0 ; i<input.length(); i++){  
            if(input.charAt(i) == 'a'){  
                charCount++;  
            }  
        }  
        **System**.out.println("count of character 'a' on String: 'Today is Monday' using for loop  " + charCount);  
        
        *//a more elegant way of counting occurrence of character in String using foreach loop*  
        
        charCount = 0; *//resetting character count*  
        for(**char** ch: input.toCharArray()){  
            if(ch == 'a'){  
                charCount++;  
            }  
        }       
        **System**.out.println("count of character 'a' on String: 'Today is Monday' using for each loop  " + charCount);  
    }  
    
          
}

Java Program to count vowels and consonants in String



Reversing order of words in a Sentence in Java - Solution

