|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Program: How to reverse Singly Linked List?**   |  | | --- | | **Description:** | | Write a sample code to reverse Singly Linked List by iterating through it only once. |  |  | | --- | | **Recursive Method:** | | 1) Divide the list in two parts - first node and rest of the linked list.  2) Call reverse for the rest of the linked list.  3) Link rest to first.  4) Fix head pointer. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/revese-singly-linked-list/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94 | package com.javacoffee.ds.linkedlist;    public class SinglyLinkedListImpl<T> {        private Node<T> head;        public void add(T element){            Node<T> nd = new Node<T>();          nd.setValue(element);          System.out.println("Adding: "+element);          Node<T> tmp = head;          while(true){              if(tmp == null){                  //since there is only one element, both head and                  //tail points to the same object.                  head = nd;                  break;              } else if(tmp.getNextRef() == null){                  tmp.setNextRef(nd);                  break;              } else {                  tmp = tmp.getNextRef();              }          }      }        public void traverse(){            Node<T> tmp = head;          while(true){              if(tmp == null){                  break;              }              System.out.print(tmp.getValue()+"\t");              tmp = tmp.getNextRef();          }      }        public void reverse(){            System.out.println("\nreversing the linked list\n");          Node<T> prev = null;          Node<T> current = head;          Node<T> next = null;          while(current != null){              next = current.getNextRef();              current.setNextRef(prev);              prev = current;              current = next;          }          head = prev;      }        public static void main(String a[]){          SinglyLinkedListImpl<Integer> sl = new SinglyLinkedListImpl<Integer>();          sl.add(3);          sl.add(32);          sl.add(54);          sl.add(89);          System.out.println();          sl.traverse();          System.out.println();          sl.reverse();          sl.traverse();      }  }    class Node<T> implements Comparable<T> {        private T value;      private Node<T> nextRef;        public T getValue() {          return value;      }      public void setValue(T value) {          this.value = value;      }      public Node<T> getNextRef() {          return nextRef;      }      public void setNextRef(Node<T> ref) {          this.nextRef = ref;      }      @Override      public int compareTo(T arg) {          if(arg == this.value){              return 0;          } else {              return 1;          }      }  } | | |
| |  | | --- | | **Output:** | | Adding: 3  Adding: 32  Adding: 54  Adding: 89  3 32 54 89  reversing the linked list  89 54 32 3 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Find out duplicate number between 1 to N numbers.**   |  | | --- | | **Description:** | | You have got a range of numbers between 1 to N, where one of the number is  repeated. You need to write a program to find out the duplicate number. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/duplicate-number/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package com.javacoffee.algos;    import java.util.ArrayList;  import java.util.List;    public class DuplicateNumber {        public int findDuplicateNumber(List<Integer> numbers){            int highestNumber = numbers.size() - 1;          int total = getSum(numbers);          int duplicate = total - (highestNumber\*(highestNumber+1)/2);          return duplicate;      }        public int getSum(List<Integer> numbers){            int sum = 0;          for(int num:numbers){              sum += num;          }          return sum;      }        public static void main(String a[]){          List<Integer> numbers = new ArrayList<Integer>();          for(int i=1;i<30;i++){              numbers.add(i);          }          //add duplicate number into the list          numbers.add(22);          DuplicateNumber dn = new DuplicateNumber();          System.out.println("Duplicate Number: "+dn.findDuplicateNumber(numbers));      }  } | | |
| |  | | --- | | **Output:** | | Duplicate Number: 22 | |
| |  | | --- | |  | |  | |
|  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Find out middle index where sum of both ends are equal.**   |  | | --- | | **Description:** | | You are given an array of numbers. Find out the array index or position  where sum of numbers preceeding the index is equals to sum of numbers  succeeding the index. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/find-middle-index/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | package com.javacoffee.algos;    public class FindMiddleIndex {        public static int findMiddleIndex(int[] numbers) throws Exception {            int endIndex = numbers.length - 1;          int startIndex = 0;          int sumLeft = 0;          int sumRight = 0;          while (true) {              if (sumLeft > sumRight) {                  sumRight += numbers[endIndex--];              } else {                  sumLeft += numbers[startIndex++];              }              if (startIndex > endIndex) {                  if (sumLeft == sumRight) {                      break;                  } else {                      throw new Exception(                              "Please pass proper array to match the requirement");                  }              }          }          return endIndex;      }        public static void main(String a[]) {          int[] num = { 2, 4, 4, 5, 4, 1 };          try {              System.out.println("Starting from index 0, adding numbers till index "                              + findMiddleIndex(num) + " and");              System.out.println("adding rest of the numbers can be equal");          } catch (Exception ex) {              System.out.println(ex.getMessage());          }      }  } | | |
| |  | | --- | | **Output:** | | Starting from index 0, adding numbers till index 2 and  adding rest of the numbers can be equal | |
| |  | | --- | |  | |

**Program: Write a singleton class.**

|  |
| --- |
| **Description:** |
| Singleton class means you can create only one object for the given class. You can create a singleton class by making its constructor as private, so that you can restrict the creation of the object. Provide a static method to get instance of the object, wherein you can handle the object creation inside the class only. In this example we are creating object by using static block. |

|  |
| --- |
| **Code:** |
| [?](https://www.java2novice.com/java-interview-programs/java-singleton/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package com.javacoffee.algos;    public class MySingleton {        private static MySingleton myObj;        static{          myObj = new MySingleton();      }        private MySingleton(){        }        public static MySingleton getInstance(){          return myObj;      }        public void testMe(){          System.out.println("Hey.... it is working!!!");      }        public static void main(String a[]){          MySingleton ms = getInstance();          ms.testMe();      }  } | |

**Program: Write a program to create deadlock between two threads.**

|  |
| --- |
| **Description:** |
| Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlocks can occur in Java when the synchronized keyword causes the executing thread to block while waiting to get the lock, associated with the specified object. Since the thread might already hold locks associated with other objects, two threads could each be waiting for the other to release a lock. In such case, they will end up waiting forever. |

|  |
| --- |
| **Code:** |
| [?](https://www.java2novice.com/java-interview-programs/thread-deadlock/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package com.javacoffee.algos;    public class MyDeadlock {        String str1 = "Java";      String str2 = "UNIX";        Thread trd1 = new Thread("My Thread 1"){          public void run(){              while(true){                  synchronized(str1){                      synchronized(str2){                          System.out.println(str1 + str2);                      }                  }              }          }      };        Thread trd2 = new Thread("My Thread 2"){          public void run(){              while(true){                  synchronized(str2){                      synchronized(str1){                          System.out.println(str2 + str1);                      }                  }              }          }      };        public static void main(String a[]){          MyDeadlock mdl = new MyDeadlock();          mdl.trd1.start();          mdl.trd2.start();      }  } | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to reverse a string using recursive algorithm.**   |  | | --- | | **Description:** | | Write a program to reverse a string using recursive methods.  You should not use any string reverse methods to do this. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/string-reverse-recursive/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package com.javacoffee.algos;    public class StringRecursiveReversal {        String reverse = "";        public String reverseString(String str){            if(str.length() == 1){              return str;          } else {              reverse += str.charAt(str.length()-1)                      +reverseString(str.substring(0,str.length()-1));              return reverse;          }      }        public static void main(String a[]){          StringRecursiveReversal srr = new StringRecursiveReversal();          System.out.println("Result: "+srr.reverseString("Java2novice"));      }  } | | |
| |  | | --- | | **Output:** | | Result: ecivon2avaJ | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to reverse a number.**   |  | | --- | | **Description:** | | Write a program to reverse a number using numeric operations. Below example shows how to reverse a number using numeric operations. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/reverse-number/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package com.javacoffee.algos;    public class NumberReverse {        public int reverseNumber(int number){            int reverse = 0;          while(number != 0){              reverse = (reverse\*10)+(number%10);              number = number/10;          }          return reverse;      }        public static void main(String a[]){          NumberReverse nr = new NumberReverse();          System.out.println("Result: "+nr.reverseNumber(17868));      }  } | | |
| |  | | --- | | **Output:** | | Result: 86871 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to convert decimal number to binary format.**   |  | | --- | | **Description:** | | Write a program to convert decimal number to binary format using numeric operations. Below example shows how to convert decimal number to binary format using numeric operations. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/decimal-to-binary/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package com.javacoffee.algos;    public class DecimalToBinary {        public void printBinaryFormat(int number){          int binary[] = new int[25];          int index = 0;          while(number > 0){              binary[index++] = number%2;              number = number/2;          }          for(int i = index-1;i >= 0;i--){              System.out.print(binary[i]);          }      }        public static void main(String a[]){          DecimalToBinary dtb = new DecimalToBinary();          dtb.printBinaryFormat(25);      }  } | | |
| |  | | --- | | **Output:** | | 11001 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to find perfect number or not.**   |  | | --- | | **Description:** | | A perfect number is a positive integer that is equal to the sum  of its proper positive divisors, that is, the sum of its positive  divisors excluding the number itself. Equivalently, a perfect number  is a number that is half the sum of all of its positive divisors.  The first perfect number is 6, because 1, 2 and 3 are its proper  positive divisors, and 1 + 2 + 3 = 6. Equivalently, the number 6  is equal to half the sum of all its positive divisors:  ( 1 + 2 + 3 + 6 ) / 2 = 6. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/perfect-number/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | package com.javacoffee.algos;    public class IsPerfectNumber {        public boolean isPerfectNumber(int number){            int temp = 0;          for(int i=1;i<=number/2;i++){              if(number%i == 0){                  temp += i;              }          }          if(temp == number){              System.out.println("It is a perfect number");              return true;          } else {              System.out.println("It is not a perfect number");              return false;          }      }        public static void main(String a[]){          IsPerfectNumber ipn = new IsPerfectNumber();          System.out.println("Is perfect number: "+ipn.isPerfectNumber(28));      }  } | | |
| |  | | --- | | **Output:** | | 28  It is a perfect number  Is perfect number: true | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to implement ArrayList.**   |  | | --- | | **Description:** | | Write a program to implement your own ArrayList class. It should  contain add(), get(), remove(), size() methods. Use dynamic array logic.  It should increase its size when it reaches threshold. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/arraylist-implementation/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | package com.javacoffee.algos;    import java.util.Arrays;    public class MyArrayList {        private Object[] myStore;      private int actSize = 0;        public MyArrayList(){          myStore = new Object[10];      }        public Object get(int index){          if(index < actSize){              return myStore[index];          } else {              throw new ArrayIndexOutOfBoundsException();          }      }        public void add(Object obj){          if(myStore.length-actSize <= 5){              increaseListSize();          }          myStore[actSize++] = obj;      }        public Object remove(int index){          if(index < actSize){              Object obj = myStore[index];              myStore[index] = null;              int tmp = index;              while(tmp < actSize){                  myStore[tmp] = myStore[tmp+1];                  myStore[tmp+1] = null;                  tmp++;              }              actSize--;              return obj;          } else {              throw new ArrayIndexOutOfBoundsException();          }        }        public int size(){          return actSize;      }        private void increaseListSize(){          myStore = Arrays.copyOf(myStore, myStore.length\*2);          System.out.println("\nNew length: "+myStore.length);      }        public static void main(String a[]){          MyArrayList mal = new MyArrayList();          mal.add(new Integer(2));          mal.add(new Integer(5));          mal.add(new Integer(1));          mal.add(new Integer(23));          mal.add(new Integer(14));          for(int i=0;i<mal.size();i++){              System.out.print(mal.get(i)+" ");          }          mal.add(new Integer(29));          System.out.println("Element at Index 5:"+mal.get(5));          System.out.println("List size: "+mal.size());          System.out.println("Removing element at index 2: "+mal.remove(2));          for(int i=0;i<mal.size();i++){              System.out.print(mal.get(i)+" ");          }      }  } | | |
| |  | | --- | | **Output:** | | 2 5 1 23 14  New length: 20  Element at Index 5:29  List size: 6  Removing element at index 2: 1  2 5 23 14 29 | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to find maximum repeated words from a file.**   |  | | --- | | **Description:** | | Write a program to read words from a file. Count the  repeated or duplicated words. Sort it by maximum repeated or  duplicated word count. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/max-repeated-words-file/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | package com.javacoffee.algos;    import java.io.BufferedReader;  import java.io.DataInputStream;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  import java.util.Set;  import java.util.StringTokenizer;  import java.util.Map.Entry;    public class MaxDuplicateWordCount {        public Map<String, Integer> getWordCount(String fileName){            FileInputStream fis = null;          DataInputStream dis = null;          BufferedReader br = null;          Map<String, Integer> wordMap = new HashMap<String, Integer>();          try {              fis = new FileInputStream(fileName);              dis = new DataInputStream(fis);              br = new BufferedReader(new InputStreamReader(dis));              String line = null;              while((line = br.readLine()) != null){                  StringTokenizer st = new StringTokenizer(line, " ");                  while(st.hasMoreTokens()){                      String tmp = st.nextToken().toLowerCase();                      if(wordMap.containsKey(tmp)){                          wordMap.put(tmp, wordMap.get(tmp)+1);                      } else {                          wordMap.put(tmp, 1);                      }                  }              }          } catch (FileNotFoundException e) {              e.printStackTrace();          } catch (IOException e) {              e.printStackTrace();          } finally{              try{if(br != null) br.close();}catch(Exception ex){}          }          return wordMap;      }        public List<Entry<String, Integer>> sortByValue(Map<String, Integer> wordMap){            Set<Entry<String, Integer>> set = wordMap.entrySet();          List<Entry<String, Integer>> list = new ArrayList<Entry<String, Integer>>(set);          Collections.sort( list, new Comparator<Map.Entry<String, Integer>>()          {              public int compare( Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2 )              {                  return (o2.getValue()).compareTo( o1.getValue() );              }          } );          return list;      }        public static void main(String a[]){          MaxDuplicateWordCount mdc = new MaxDuplicateWordCount();          Map<String, Integer> wordMap = mdc.getWordCount("C:/MyTestFile.txt");          List<Entry<String, Integer>> list = mdc.sortByValue(wordMap);          for(Map.Entry<String, Integer> entry:list){              System.out.println(entry.getKey()+" ==== "+entry.getValue());          }      }  } | |  |  | | |  | |
| |  | | --- | | **Output:** | | one ==== 3  the ==== 3  that ==== 3  of ==== 2  in ==== 2  some ==== 2  to ==== 1  summary ==== 1  but ==== 1  have ==== 1  common ==== 1  least ==== 1  simplest ==== 1 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to find out duplicate characters in a string.**   |  | | --- | | **Description:** | | Write a program to find out duplicate or repeated characters in a  string, and calculate the count of repeatation. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/duplicate-string-character-count/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | package com.javacoffee.algos;    import java.util.HashMap;  import java.util.Map;  import java.util.Set;    public class DuplicateCharsInString {        public void findDuplicateChars(String str){            Map<Character, Integer> dupMap = new HashMap<Character, Integer>();          char[] chrs = str.toCharArray();          for(Character ch:chrs){              if(dupMap.containsKey(ch)){                  dupMap.put(ch, dupMap.get(ch)+1);              } else {                  dupMap.put(ch, 1);              }          }          Set<Character> keys = dupMap.keySet();          for(Character ch:keys){              if(dupMap.get(ch) > 1){                  System.out.println(ch+"--->"+dupMap.get(ch));              }          }      }        public static void main(String a[]){          DuplicateCharsInString dcs = new DuplicateCharsInString();          dcs.findDuplicateChars("Java2Novice");      }  } | | |
|  |
| |  | | --- | | **Output:** | | v--->2  a--->2 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to find top two maximum numbers in a array.**   |  | | --- | | **Description:** | | Write a program to find top two maximum numbers in the  given array. You should not use any sorting functions. You  should iterate the array only once. You should not use any  kind of collections in java. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/two-max-numbers-in-array/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package com.javacoffee.algos;    public class TwoMaxNumbers {        public void printTwoMaxNumbers(int[] nums){          int maxOne = 0;          int maxTwo = 0;          for(int n:nums){              if(maxOne < n){                  maxTwo = maxOne;                  maxOne =n;              } else if(maxTwo < n){                  maxTwo = n;              }          }          System.out.println("First Max Number: "+maxOne);          System.out.println("Second Max Number: "+maxTwo);      }        public static void main(String a[]){          int num[] = {5,34,78,2,45,1,99,23};          TwoMaxNumbers tmn = new TwoMaxNumbers();          tmn.printTwoMaxNumbers(num);      }  } | | |
| |  | | --- | | **Output:** | | First Max Number: 99  Second Max Number: 78 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to sort a map by value.**   |  | | --- | | **Description:** | | Sort or order a HashMap or TreeSet or any map item by value. Write a comparator  which compares by value, not by key. Entry class might hleps you here. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/sort-a-map-by-value/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package com.javacoffee.algos;    import java.util.ArrayList;  import java.util.Collections;  import java.util.Comparator;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  import java.util.Set;  import java.util.Map.Entry;    public class OrderByValue {        public static void main(String a[]){          Map<String, Integer> map = new HashMap<String, Integer>();          map.put("java", 20);          map.put("C++", 45);          map.put("Java2Novice", 2);          map.put("Unix", 67);          map.put("MAC", 26);          map.put("Why this kolavari", 93);          Set<Entry<String, Integer>> set = map.entrySet();          List<Entry<String, Integer>> list = new ArrayList<Entry<String, Integer>>(set);          Collections.sort( list, new Comparator<Map.Entry<String, Integer>>()          {              public int compare( Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2 )              {                  return (o2.getValue()).compareTo( o1.getValue() );              }          } );          for(Map.Entry<String, Integer> entry:list){              System.out.println(entry.getKey()+" ==== "+entry.getValue());          }      }  } | | |
| |  | | --- | | **Output:** | | Why this kolavari ==== 93  Unix ==== 67  C++ ==== 45  MAC ==== 26  java ==== 20  Java2Novice ==== 2 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to find common elements between two arrays.**   |  | | --- | | **Description:** | | Write a program to identify common elements or numbers between  two given arrays. You should not use any inbuilt methods are list to  find common values. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/common-elements-in-two-arrays/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package com.javacoffee.algos;    public class CommonElementsInArray {        public static void main(String a[]){          int[] arr1 = {4,7,3,9,2};          int[] arr2 = {3,2,12,9,40,32,4};          for(int i=0;i<arr1.length;i++){              for(int j=0;j<arr2.length;j++){                  if(arr1[i]==arr2[j]){                      System.out.println(arr1[i]);                  }              }          }      }  } | | |
| |  | | --- | | **Output:** | | 4  3  9  2 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: How to swap two numbers without using temporary variable?**   |  | | --- | | **Description:** | | Write a program to swap or exchange two numbers. You should  not use any temporary or third variable to swap. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/swap-two-numbers/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | package com.javacoffee.algos;    public class MySwapingTwoNumbers {        public static void main(String a[]){          int x = 10;          int y = 20;          System.out.println("Before swap:");          System.out.println("x value: "+x);          System.out.println("y value: "+y);          x = x+y;          y=x-y;          x=x-y;          System.out.println("After swap:");          System.out.println("x value: "+x);          System.out.println("y value: "+y);      }  } | | |
| |  | | --- | | **Output:** | | Before swap:  x value: 10  y value: 20  After swap:  x value: 20  y value: 10 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to print fibonacci series.**   |  | | --- | | **Description:** | | In mathematics, the Fibonacci numbers or Fibonacci series or Fibonacci sequence are the numbers in the following integer sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144... By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two. Below example shows how to create fibonacci series. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/fibonacci-series/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package com.javacoffee.algos;    public class MyFibonacci {        public static void main(String a[]){             int febCount = 15;           int[] feb = new int[febCount];           feb[0] = 0;           feb[1] = 1;           for(int i=2; i < febCount; i++){               feb[i] = feb[i-1] + feb[i-2];           }             for(int i=0; i< febCount; i++){                   System.out.print(feb[i] + " ");           }      }  } | | |
| |  | | --- | | **Output:** | | 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to find sum of each digit in the given number using recursion.**   |  | | --- | | **Description:** | | Below example shows how to find out sum of each digit in the given number using recursion logic. For example, if the number is 259, then the sum should be 2+5+9 = 16. |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  public class MyNumberSumRec {  int sum = 0;    public int getNumberSum(int number){    if(number == 0){  return sum;  } else {  sum += (number%10);  getNumberSum(number/10);  }  return sum;  }    public static void main(String a[]){  MyNumberSumRec mns = new MyNumberSumRec();  System.out.println("Sum is: "+mns.getNumberSum(223));  }  } | |
| |  | | --- | | **Output:** | | Sum is: 7 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to check the given number is a prime number or not?**   |  | | --- | | **Description:** | | A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. A natural number greater than 1 that is not a prime number is called a composite number. For example, 5 is prime, as only 1 and 5 divide it, whereas 6 is composite, since it has the divisors 2 and 3 in addition to 1 and 6. The fundamental theorem of arithmetic establishes the central role of primes in number theory: any integer greater than 1 can be expressed as a product of primes that is unique up to ordering. This theorem requires excluding 1 as a prime. |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  public class MyPrimeNumCheck {  public boolean isPrimeNumber(int number){    for(int i=2; i<=number/2; i++){  if(number % i == 0){  return false;  }  }  return true;  }    public static void main(String a[]){  MyPrimeNumCheck mpc = new MyPrimeNumCheck();  System.out.println("Is 17 prime number? "+mpc.isPrimeNumber(17));  System.out.println("Is 19 prime number? "+mpc.isPrimeNumber(19));  System.out.println("Is 15 prime number? "+mpc.isPrimeNumber(15));  }  } | |
| |  | | --- | | **Output:** | | Is 17 prime number? true  Is 19 prime number? true  Is 15 prime number? false | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to find the given number is Armstrong number or not?**   |  | | --- | | **Description:** | | Armstrong numbers are the sum of their own digits to the power of  the number of digits. It is also known as narcissistic numbers. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/armstrong-number/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | package com.javacoffee.algos;    public class MyArmstrongNumber {        public boolean isArmstrongNumber(int number){            int tmp = number;          int noOfDigits = String.valueOf(number).length();          int sum = 0;          int div = 0;          while(tmp > 0)          {              div = tmp % 10;              int temp = 1;              for(int i=0;i<noOfDigits;i++){                  temp \*= div;              }              sum += temp;              tmp = tmp/10;          }          if(number == sum) {              return true;          } else {              return false;          }      }        public static void main(String a[]){          MyArmstrongNumber man = new MyArmstrongNumber();          System.out.println("Is 371 Armstrong number? "+man.isArmstrongNumber(371));          System.out.println("Is 523 Armstrong number? "+man.isArmstrongNumber(523));          System.out.println("Is 153 Armstrong number? "+man.isArmstrongNumber(153));      }  } | | |
| |  | | --- | | **Output:** | | Is 371 Armstrong number? true  Is 523 Armstrong number? false  Is 153 Armstrong number? true | |
| |  | | --- | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to convert binary to decimal number.**   |  | | --- | | **Description:** | | Write a program to convert binary format to decimal number using numeric operations. Below example shows how to convert binary to decimal format using numeric operations. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/binary-to-decimal/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | package com.javacoffee.algos;    public class BinaryToDecimal {        public int getDecimalFromBinary(int binary){            int decimal = 0;          int power = 0;          while(true){              if(binary == 0){                  break;              } else {                  int tmp = binary%10;                  decimal += tmp\*Math.pow(2, power);                  binary = binary/10;                  power++;              }          }          return decimal;      }        public static void main(String a[]){          BinaryToDecimal bd = new BinaryToDecimal();          System.out.println("11 ===> "+bd.getDecimalFromBinary(11));          System.out.println("110 ===> "+bd.getDecimalFromBinary(110));          System.out.println("100110 ===> "+bd.getDecimalFromBinary(100110));      }  } | | |
| |  | | --- | | **Output:** | | 11 ===> 3  110 ===> 6  100110 ===> 38 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to check the given number is binary number or not?**   |  | | --- | | **Description:** | | The binary numeral system, or base-2 number system, represents numeric values using two symbols: 0 and 1. More specifically, the usual base-2 system is a positional notation with a radix of 2. Because of its straightforward implementation in digital electronic circuitry using logic gates, the binary system is used internally by almost all modern computers. |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  public class MyBinaryCheck {  public boolean isBinaryNumber(int binary){    boolean status = true;  while(true){  if(binary == 0){  break;  } else {  int tmp = binary%10;  if(tmp > 1){  status = false;  break;  }  binary = binary/10;  }  }  return status;  }    public static void main(String a[]){  MyBinaryCheck mbc = new MyBinaryCheck();  System.out.println("Is 1000111 binary? :"+mbc.isBinaryNumber(1000111));  System.out.println("Is 10300111 binary? :"+mbc.isBinaryNumber(10300111));  }  } | |
| |  | | --- | | **Output:** | | Is 1000111 binary? :true  Is 10300111 binary? :false | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program for Bubble Sort in java.**   |  | | --- | | **Description:** | | Bubble sort is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements bubble to the top of the list. Because it only uses comparisons to operate on elements, it is a comparison sort. You can see the code implementation below: |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  public class MyBubbleSort {  // logic to sort the elements  public static void bubble\_srt(int array[]) {  int n = array.length;  int k;  for (int m = n; m >= 0; m--) {  for (int i = 0; i < n - 1; i++) {  k = i + 1;  if (array[i] > array[k]) {  swapNumbers(i, k, array);  }  }  printNumbers(array);  }  }  private static void swapNumbers(int i, int j, int[] array) {  int temp;  temp = array[i];  array[i] = array[j];  array[j] = temp;  }  private static void printNumbers(int[] input) {    for (int i = 0; i < input.length; i++) {  System.out.print(input[i] + ", ");  }  System.out.println("\n");  }  public static void main(String[] args) {  int[] input = { 4, 2, 9, 6, 23, 12, 34, 0, 1 };  bubble\_srt(input);  }  } | |
| |  | | --- | | **Output:** | | 2, 4, 6, 9, 12, 23, 0, 1, 34,  2, 4, 6, 9, 12, 0, 1, 23, 34,  2, 4, 6, 9, 0, 1, 12, 23, 34,  2, 4, 6, 0, 1, 9, 12, 23, 34,  2, 4, 0, 1, 6, 9, 12, 23, 34,  2, 0, 1, 4, 6, 9, 12, 23, 34,  0, 1, 2, 4, 6, 9, 12, 23, 34,  0, 1, 2, 4, 6, 9, 12, 23, 34,  0, 1, 2, 4, 6, 9, 12, 23, 34,  0, 1, 2, 4, 6, 9, 12, 23, 34, | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program for Insertion Sort in java.**   |  | | --- | | **Description:** | | Insertion sort is a simple sorting algorithm that builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. Every repetition of insertion sort removes an element from the input data, inserting it into the correct position in the already-sorted list, until no input elements remain. The choice of which element to remove from the input is arbitrary, and can be made using almost any choice algorithm. You can see the code implementation below: |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  public class MyInsertionSort {  public static void main(String[] args) {    int[] input = { 4, 2, 9, 6, 23, 12, 34, 0, 1 };  insertionSort(input);  }    private static void printNumbers(int[] input) {    for (int i = 0; i < input.length; i++) {  System.out.print(input[i] + ", ");  }  System.out.println("\n");  }  public static void insertionSort(int array[]) {  int n = array.length;  for (int j = 1; j < n; j++) {  int key = array[j];  int i = j-1;  while ( (i > -1) && ( array [i] > key ) ) {  array [i+1] = array [i];  i--;  }  array[i+1] = key;  printNumbers(array);  }  }  } | |
| |  | | --- | | **Output:** | | 2, 4, 9, 6, 23, 12, 34, 0, 1,  2, 4, 9, 6, 23, 12, 34, 0, 1,  2, 4, 6, 9, 23, 12, 34, 0, 1,  2, 4, 6, 9, 23, 12, 34, 0, 1,  2, 4, 6, 9, 12, 23, 34, 0, 1,  2, 4, 6, 9, 12, 23, 34, 0, 1,  0, 2, 4, 6, 9, 12, 23, 34, 1,  0, 1, 2, 4, 6, 9, 12, 23, 34, | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to implement hashcode and equals.**   |  | | --- | | **Description:** | | The hashcode of a Java Object is simply a number, it is 32-bit signed int, that allows an object to be managed by a hash-based data structure. We know that hash code is an unique id number allocated to an object by JVM. But actually speaking, Hash code is not an unique number for an object. If two objects are equals then these two objects should return same hash code. So we have to implement hashcode() method of a class in such way that if two objects are equals, ie compared by equal() method of that class, then those two objects must return same hash code. If you are overriding hashCode you need to override equals method also.   The below example shows how to override equals and hashcode methods. The class Price overrides equals and hashcode. If you notice the hashcode implementation, it always generates unique hashcode for each object based on their state, ie if the object state is same, then you will get same hashcode. A HashMap is used in the example to store Price objects as keys. It shows though we generate different objects, but if state is same, still we can use this as key. |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  import java.util.HashMap;  public class MyHashcodeImpl {  public static void main(String a[]){    HashMap<Price, String> hm = new HashMap<Price, String>();  hm.put(new Price("Banana", 20), "Banana");  hm.put(new Price("Apple", 40), "Apple");  hm.put(new Price("Orange", 30), "Orange");  //creating new object to use as key to get value  Price key = new Price("Banana", 20);  System.out.println("Hashcode of the key: "+key.hashCode());  System.out.println("Value from map: "+hm.get(key));  }  }  class Price{    private String item;  private int price;    public Price(String itm, int pr){  this.item = itm;  this.price = pr;  }    public int hashCode(){  System.out.println("In hashcode");  int hashcode = 0;  hashcode = price\*20;  hashcode += item.hashCode();  return hashcode;  }    public boolean equals(Object obj){  System.out.println("In equals");  if (obj instanceof Price) {  Price pp = (Price) obj;  return (pp.item.equals(this.item) && pp.price == this.price);  } else {  return false;  }  }    public String getItem() {  return item;  }  public void setItem(String item) {  this.item = item;  }  public int getPrice() {  return price;  }  public void setPrice(int price) {  this.price = price;  }    public String toString(){  return "item: "+item+" price: "+price;  }  } | |
| |  | | --- | | **Output:** | | In hashcode  In hashcode  In hashcode  In hashcode  Hashcode of the key: 1982479637  In hashcode  In equals  Value from map: Banana | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: How to get distinct elements from an array by avoiding duplicate elements?**   |  | | --- | | **Description:** | | The below example shows how to avoid duplicate elements from an array and disply only distinct elements. Please use only arrays to process it. |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  public class MyDisticntElements {  public static void printDistinctElements(int[] arr){    for(int i=0;i<arr.length;i++){  boolean isDistinct = false;  for(int j=0;j<i;j++){  if(arr[i] == arr[j]){  isDistinct = true;  break;  }  }  if(!isDistinct){  System.out.print(arr[i]+" ");  }  }  }    public static void main(String a[]){    int[] nums = {5,2,7,2,4,7,8,2,3};  MyDisticntElements.printDistinctElements(nums);  }  } | |
| |  | | --- | | **Output:** | | 5 2 7 4 8 3 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to get distinct word list from the given file.**   |  | | --- | | **Description:** | | Write a program to find all distinct words from the given file. Remove special chars like ".,;:" etc. Ignore case sensitivity. |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  import java.io.BufferedReader;  import java.io.DataInputStream;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.ArrayList;  import java.util.List;  import java.util.StringTokenizer;  public class MyDistinctFileWords {  public List<String> getDistinctWordList(String fileName){  FileInputStream fis = null;  DataInputStream dis = null;  BufferedReader br = null;  List<String> wordList = new ArrayList<String>();  try {  fis = new FileInputStream(fileName);  dis = new DataInputStream(fis);  br = new BufferedReader(new InputStreamReader(dis));  String line = null;  while((line = br.readLine()) != null){  StringTokenizer st = new StringTokenizer(line, " ,.;:\"");  while(st.hasMoreTokens()){  String tmp = st.nextToken().toLowerCase();  if(!wordList.contains(tmp)){  wordList.add(tmp);  }  }  }  } catch (FileNotFoundException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  } finally{  try{if(br != null) br.close();}catch(Exception ex){}  }  return wordList;  }    public static void main(String a[]){    MyDistinctFileWords distFw = new MyDistinctFileWords();  List<String> wordList = distFw.getDistinctWordList("C:/sample.txt");  for(String str:wordList){  System.out.println(str);  }  }  } | |
| |  | | --- | | **Output:** | | the  while  statement  verifies  condition  before  entering  into  loop  to  see  whether  next  iteration  should  occur  or  not  do-while  executes  first  without  checking  it  after  finishing  each  will  always  execute  body  of  a  at  least  once | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to get a line with max word count from the given file.**   |  | | --- | | **Description:** | | Below example shows how to find out the line with maximum number of word count in the given file. In case if it has multiple lines with max number of words, then it has to list all those lines. |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  import java.io.BufferedReader;  import java.io.DataInputStream;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.ArrayList;  import java.util.List;  public class MaxWordCountInLine {  private int currentMaxCount = 0;  private List<String> lines = new ArrayList<String>();    public void readMaxLineCount(String fileName){  FileInputStream fis = null;  DataInputStream dis = null;  BufferedReader br = null;    try {  fis = new FileInputStream(fileName);  dis = new DataInputStream(fis);  br = new BufferedReader(new InputStreamReader(dis));  String line = null;  while((line = br.readLine()) != null){    int count = (line.split("\\s+")).length;  if(count > currentMaxCount){  lines.clear();  lines.add(line);  currentMaxCount = count;  } else if(count == currentMaxCount){  lines.add(line);  }  }  } catch (FileNotFoundException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  } finally{  try{  if(br != null) br.close();  }catch(Exception ex){}  }  }  public int getCurrentMaxCount() {  return currentMaxCount;  }  public void setCurrentMaxCount(int currentMaxCount) {  this.currentMaxCount = currentMaxCount;  }  public List<String> getLines() {  return lines;  }  public void setLines(List<String> lines) {  this.lines = lines;  }    public static void main(String a[]){    MaxWordCountInLine mdc = new MaxWordCountInLine();  mdc.readMaxLineCount("/Users/ngootooru/MyTestFile.txt");  System.out.println("Max number of words in a line is: "+mdc.getCurrentMaxCount());  System.out.println("Line with max word count:");  List<String> lines = mdc.getLines();  for(String l:lines){  System.out.println(l);  }  }  } | |
| |  | | --- | | **MyTestFile.txt:** | | true, false, and null might seem like keywords, but they are actually literals.  You cannot use them as identifiers in your programs. The servlet context  is an interface which helps to communicate with other servlets. It contains  information about the Web application and container. It is kind of  application environment. Using the context, a servlet can obtain URL  references to resources, and store attributes that other servlets in the  context can use. | |
| |  | | --- | | **Output:** | | Max number of words in a line is: 13  Line with max word count:  true, false, and null might seem like keywords, but they are actually literals. | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to convert string to number without using Integer.parseInt() method.**   |  | | --- | | **Description:** | | Below example shows how to convert string format of a number to number without calling Integer.parseInt() method. We can do this by converting each character into ascii format and form the number. |  |  | | --- | | **Code:** | | package com.javacoffee.algos;  public class MyStringToNumber {  public static int convert\_String\_To\_Number(String numStr){    char ch[] = numStr.toCharArray();  int sum = 0;  //get ascii value for zero  int zeroAscii = (int)'0';  for(char c:ch){  int tmpAscii = (int)c;  sum = (sum\*10)+(tmpAscii-zeroAscii);  }  return sum;  }    public static void main(String a[]){    System.out.println("\"3256\" == "+convert\_String\_To\_Number("3256"));  System.out.println("\"76289\" == "+convert\_String\_To\_Number("76289"));  System.out.println("\"90087\" == "+convert\_String\_To\_Number("90087"));  }  } | |
| |  | | --- | | **Output:** | | "3256" == 3256  "76289" == 76289  "90087" == 90087 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to find two lines with max characters in descending order.**   |  | | --- | | **Description:** | | Write a program to read a multiple line text file and write the 'N' longest lines to the output console, where the file to be read is specified as command line aruguments. The program should read an input file. The first line should contain the value of the number 'N' followed by multiple lines. 'N' should be a valid positive integer. |  |  | | --- | | **Code:** | | package com.longest.lines;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileNotFoundException;  import java.io.FileReader;  import java.io.IOException;  import java.util.Comparator;  import java.util.Set;  import java.util.TreeSet;  public class Main {  public static void main(String[] args) {    BufferedReader br = null;  String filePath = args[0];  int topList = 0;  Set<Entries> liSet = new TreeSet<Entries>(new MyComp());  try {  br = new BufferedReader(new FileReader(new File(filePath)));  String line = br.readLine();  topList = Integer.parseInt(line.trim());  while((line = br.readLine()) != null){  line = line.trim();  if(!"".equals(line)){  liSet.add(new Entries(line.length(), line));  }  }  int count = 0;  for(Entries ent:liSet){  System.out.println(ent.line);  if(++count == topList){  break;  }  }  } catch (FileNotFoundException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  public static class Entries{  Integer length;  String line;  public Entries(Integer l,String line){  length = l;  this.line = line;  }  }    public static class MyComp implements Comparator<Entries>{  @Override  public int compare(Entries e1, Entries e2) {  if(e2.length > e1.length){  return 1;  } else {  return -1;  }  }    }  } | |
| |  | | --- | | **Sample input file:** | | 3  Java2novice  My Test line 123  Java world  I know java language  This is a test program  java is simple | |
| |  | | --- | | **Output:** | | This is a test program  I know java language  My Test line 123 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: Write a program to find the sum of the first 1000 prime numbers.**   |  | | --- | | **Description:** | | Write a program to find the sum of the first 1000 prime numbers. |  |  | | --- | | **Code:** | | package com.primesum;  public class Main {  public static void main(String args[]){    int number = 2;  int count = 0;  long sum = 0;  while(count < 1000){  if(isPrimeNumber(number)){  sum += number;  count++;  }  number++;  }  System.out.println(sum);  }    private static boolean isPrimeNumber(int number){    for(int i=2; i<=number/2; i++){  if(number % i == 0){  return false;  }  }  return true;  }  } | |
| |  | | --- | | **Output:** | | 3682913 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Find longest substring without repeating characters.**   |  | | --- | | **Description:** | | Given a string, find the longest substrings without repeating characters. Iterate through the given string, find the longest maximum substrings. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/longest-substring/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63 | package com.javacoffee.algos;    import java.util.HashSet;  import java.util.Set;    public class MyLongestSubstr {        private Set<String> subStrList = new HashSet<String>();      private int finalSubStrSize = 0;        public Set<String> getLongestSubstr(String input){          //reset instance variables          subStrList.clear();          finalSubStrSize = 0;          // have a boolean flag on each character ascii value          boolean[] flag = new boolean[256];          int j = 0;          char[] inputCharArr = input.toCharArray();          for (int i = 0; i < inputCharArr.length; i++) {              char c = inputCharArr[i];              if (flag[c]) {                  extractSubString(inputCharArr,j,i);                  for (int k = j; k < i; k++) {                      if (inputCharArr[k] == c) {                          j = k + 1;                          break;                      }                      flag[inputCharArr[k]] = false;                  }              } else {                  flag[c] = true;              }          }          extractSubString(inputCharArr,j,inputCharArr.length);          return subStrList;      }        private String extractSubString(char[] inputArr, int start, int end){            StringBuilder sb = new StringBuilder();          for(int i=start;i<end;i++){              sb.append(inputArr[i]);          }          String subStr = sb.toString();          if(subStr.length() > finalSubStrSize){              finalSubStrSize = subStr.length();              subStrList.clear();              subStrList.add(subStr);          } else if(subStr.length() == finalSubStrSize){              subStrList.add(subStr);          }            return sb.toString();      }        public static void main(String a[]){          MyLongestSubstr mls = new MyLongestSubstr();          System.out.println(mls.getLongestSubstr("java2novice"));          System.out.println(mls.getLongestSubstr("java\_language\_is\_sweet"));          System.out.println(mls.getLongestSubstr("java\_java\_java\_java"));          System.out.println(mls.getLongestSubstr("abcabcbb"));      }  } | | |
| |  | | --- | | **Output:** | | [a2novice]  [uage\_is]  [\_jav, va\_j]  [cab, abc, bca] | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program: Write a program to remove duplicates from sorted array.**   |  | | --- | | **Description:** | | Given array is already sorted, and it has duplicate elements. Write a program to remove duplicate elements and return new array without any duplicate elements. The array should contain only unique elements. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/remove-duplicates-sorted-array/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package com.javacoffee.algos;    public class MyDuplicateElements {        public static int[] removeDuplicates(int[] input){            int j = 0;          int i = 1;          //return if the array length is less than 2          if(input.length < 2){              return input;          }          while(i < input.length){              if(input[i] == input[j]){                  i++;              }else{                  input[++j] = input[i++];              }          }          int[] output = new int[j+1];          for(int k=0; k<output.length; k++){              output[k] = input[k];          }            return output;      }        public static void main(String a[]){          int[] input1 = {2,3,6,6,8,9,10,10,10,12,12};          int[] output = removeDuplicates(input1);          for(int i:output){              System.out.print(i+" ");          }      }  } | | |
| |  | | --- | | **Output:** | | 2 3 6 8 9 10 12 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Program: How to sort a Stack using a temporary Stack?**   |  | | --- | | **Description:** | | You have a stack with full of integers. Sort it in the ascending order using another temporary array by using all stack functionality. |  |  | | --- | | **Code:** | | package com.javacoffee.algo;  import java.util.Stack;  public class StackSort {  public static Stack<Integer> sortStack(Stack<Integer> input){    Stack<Integer> tmpStack = new Stack<Integer>();  System.out.println("=============== debug logs ================");  while(!input.isEmpty()) {  int tmp = input.pop();  System.out.println("Element taken out: "+tmp);  while(!tmpStack.isEmpty() && tmpStack.peek() > tmp) {  input.push(tmpStack.pop());  }  tmpStack.push(tmp);  System.out.println("input: "+input);  System.out.println("tmpStack: "+tmpStack);  }  System.out.println("=============== debug logs ended ================");  return tmpStack;  }    public static void main(String a[]){    Stack<Integer> input = new Stack<Integer>();  input.add(34);  input.add(3);  input.add(31);  input.add(98);  input.add(92);  input.add(23);  System.out.println("input: "+input);  System.out.println("final sorted list: "+sortStack(input));  }  } | |
| |  | | --- | | **Output:** | | input: [34, 3, 31, 98, 92, 23]  =============== debug logs ================  Element taken out: 23  input: [34, 3, 31, 98, 92]  tmpStack: [23]  Element taken out: 92  input: [34, 3, 31, 98]  tmpStack: [23, 92]  Element taken out: 98  input: [34, 3, 31]  tmpStack: [23, 92, 98]  Element taken out: 31  input: [34, 3, 98, 92]  tmpStack: [23, 31]  Element taken out: 92  input: [34, 3, 98]  tmpStack: [23, 31, 92]  Element taken out: 98  input: [34, 3]  tmpStack: [23, 31, 92, 98]  Element taken out: 3  input: [34, 98, 92, 31, 23]  tmpStack: [3]  Element taken out: 23  input: [34, 98, 92, 31]  tmpStack: [3, 23]  Element taken out: 31  input: [34, 98, 92]  tmpStack: [3, 23, 31]  Element taken out: 92  input: [34, 98]  tmpStack: [3, 23, 31, 92]  Element taken out: 98  input: [34]  tmpStack: [3, 23, 31, 92, 98]  Element taken out: 34  input: [98, 92]  tmpStack: [3, 23, 31, 34]  Element taken out: 92  input: [98]  tmpStack: [3, 23, 31, 34, 92]  Element taken out: 98  input: []  tmpStack: [3, 23, 31, 34, 92, 98]  =============== debug logs ended ================  final sorted list: [3, 23, 31, 34, 92, 98] | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Program: Write a program to print all permutations of a given string.  |  | | --- | | **Description:** | | This page gives an example to print all permutations of a given string. For example, *xy* would be *xy* and *yx*.  Here is the steps to implement string permutations:   1. Take out the first char and keep it constant. And permute rest of the characters. 2. User recursive method call to permute rest of the string except first character. 3. While making recursive call, we accumulate each character being constant along with recursive call response. |  |  | | --- | | **Code:** | | [?](https://www.java2novice.com/java-interview-programs/string-permutations/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | package com.javacoffee.algo;    import java.util.ArrayList;  import java.util.List;    public class StringPermutationsEx {        public static void main(String a[]) {            List<String> output = StringPermutationsEx.generatePermutations("xyz");          System.out.println("Result size: "+output.size());          output.stream().forEach(System.out::println);          System.out.println("------------------");            output = StringPermutationsEx.generatePermutations("ABCD");          System.out.println("Result size: "+output.size());          output.stream().forEach(System.out::println);      }        public static List<String> generatePermutations(String input) {            List<String> strList = new ArrayList<String>();          StringPermutationsEx.permutations("", input, strList);            return strList;      }        private static void permutations(String consChars, String input, List<String> opContainer) {            if(input.isEmpty()) {              opContainer.add(consChars+input);              return;          }            for(int i=0; i<input.length(); i++) {              permutations(consChars+input.charAt(i),                              input.substring(0, i)+input.substring(i+1),                              opContainer);          }      }  } | | |
| |  | | --- | | **Output:** | | Result size: 6  xyz  xzy  yxz  yzx  zxy  zyx  ------------------  Result size: 24  ABCD  ABDC  ACBD  ACDB  ADBC  ADCB  BACD  BADC  BCAD  BCDA  BDAC  BDCA  CABD  CADB  CBAD  CBDA  CDAB  CDBA  DABC  DACB  DBAC  DBCA  DCAB  DCBA | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Implement Binary Search Tree (BST)  |  | | --- | | **Description:** | | For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root.  Here is an example picture of binary search tree (BST):  Binary Search Tree |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/implement-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.javacoffee.ds;    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;      }  } | |  |  | | --- | | **BinarySearchTreeImpl** | | [?](https://www.java2novice.com/java-interview-programs/implement-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | package com.javacoffee.ds;    public class BinarySearchTreeImpl {        private BstNode root;        public boolean isEmpty() {            return (this.root == null);      }        public void insert(Integer data) {            System.out.print("[input: "+data+"]");          if(root == null) {              this.root = new BstNode(data);              System.out.println(" -> inserted: "+data);              return;          }            insertNode(this.root, data);          System.out.print(" -> inserted: "+data);          System.out.println();      }        private BstNode insertNode(BstNode root, Integer data) {            BstNode tmpNode = null;          System.out.print(" ->"+root.getData());          if(root.getData() >= data) {              System.out.print(" [L]");              if(root.getLeft() == null) {                  root.setLeft(new BstNode(data));                  return root.getLeft();              } else {                  tmpNode = root.getLeft();              }          } else {              System.out.print(" [R]");              if(root.getRight() == null) {                  root.setRight(new BstNode(data));                  return root.getRight();              } else {                  tmpNode = root.getRight();              }          }            return insertNode(tmpNode, data);      }        public static void main(String a[]) {            BinarySearchTreeImpl bst = new BinarySearchTreeImpl();          bst.insert(10);          bst.insert(20);          bst.insert(21);          bst.insert(8);          bst.insert(6);          bst.insert(16);          bst.insert(23);      }  } | | |
| |  | | --- | | **Output:** | | [input: 10] -> inserted: 10  [input: 20] ->10 [R] -> inserted: 20  [input: 21] ->10 [R] ->20 [R] -> inserted: 21  [input: 8] ->10 [L] -> inserted: 8  [input: 6] ->10 [L] ->8 [L] -> inserted: 6  [input: 16] ->10 [R] ->20 [L] -> inserted: 16  [input: 23] ->10 [R] ->20 [R] ->21 [R] -> inserted: 23 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Find min and max value from Binary Search Tree (BST)  |  | | --- | | **Description:** | | For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root. This example shows how to find min & max value of a binary search tree.  Here is an example picture of binary search tree (BST):  Binary Search Tree |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/min-max-value-from-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.javacoffee.ds;    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;      }  } | |  |  | | --- | | **BinarySearchTreeImpl** | | [?](https://www.java2novice.com/java-interview-programs/min-max-value-from-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92 | package com.javacoffee.ds;    public class BinarySearchTreeImpl {        private BstNode root;        public boolean isEmpty() {            return (this.root == null);      }        public void insert(Integer data) {            System.out.print("[input: "+data+"]");          if(root == null) {              this.root = new BstNode(data);              System.out.println(" -> inserted: "+data);              return;          }            insertNode(this.root, data);          System.out.print(" -> inserted: "+data);          System.out.println();      }        private BstNode insertNode(BstNode root, Integer data) {            BstNode tmpNode = null;          System.out.print(" ->"+root.getData());          if(root.getData() >= data) {              System.out.print(" [L]");              if(root.getLeft() == null) {                  root.setLeft(new BstNode(data));                  return root.getLeft();              } else {                  tmpNode = root.getLeft();              }          } else {              System.out.print(" [R]");              if(root.getRight() == null) {                  root.setRight(new BstNode(data));                  return root.getRight();              } else {                  tmpNode = root.getRight();              }          }            return insertNode(tmpNode, data);      }        public Integer findMinValue() {            return minValue(this.root);      }        public Integer findMaxValue() {            return maxValue(this.root);      }        private Integer minValue(BstNode node) {            if(node.getLeft() != null) {              return minValue(node.getLeft());          }          return node.getData();      }        private Integer maxValue(BstNode node) {            if(node.getRight() != null) {              return maxValue(node.getRight());          }          return node.getData();      }        public static void main(String a[]) {            BinarySearchTreeImpl bst = new BinarySearchTreeImpl();          bst.insert(10);          bst.insert(20);          bst.insert(21);          bst.insert(8);          bst.insert(6);          bst.insert(16);          bst.insert(23);          bst.insert(2);          System.out.println("-------------------");          System.out.println("Min value: "+bst.findMinValue());          System.out.println("Max value: "+bst.findMaxValue());      }  } | | |
| |  | | --- | | **Output:** | | [input: 10] -> inserted: 10  [input: 20] ->10 [R] -> inserted: 20  [input: 21] ->10 [R] ->20 [R] -> inserted: 21  [input: 8] ->10 [L] -> inserted: 8  [input: 6] ->10 [L] ->8 [L] -> inserted: 6  [input: 16] ->10 [R] ->20 [L] -> inserted: 16  [input: 23] ->10 [R] ->20 [R] ->21 [R] -> inserted: 23  [input: 2] ->10 [L] ->8 [L] ->6 [L] -> inserted: 2  -------------------  Min value: 2  Max value: 23 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Find height of a Binary Search Tree (BST)  |  | | --- | | **Description:** | | For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root. This example shows how to find height of a binary search tree.  Here is an example picture of binary search tree (BST):  Binary Search Tree |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/height-of-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.javacoffee.ds;    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;      }  } | |  |  | | --- | | **BinarySearchTreeImpl** | | [?](https://www.java2novice.com/java-interview-programs/height-of-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | package com.javacoffee.ds;    public class BinarySearchTreeImpl {        private BstNode root;        public boolean isEmpty() {            return (this.root == null);      }        public void insert(Integer data) {            System.out.print("[input: "+data+"]");          if(root == null) {              this.root = new BstNode(data);              System.out.println(" -> inserted: "+data);              return;          }            insertNode(this.root, data);          System.out.print(" -> inserted: "+data);          System.out.println();      }        private BstNode insertNode(BstNode root, Integer data) {            BstNode tmpNode = null;          System.out.print(" ->"+root.getData());          if(root.getData() >= data) {              System.out.print(" [L]");              if(root.getLeft() == null) {                  root.setLeft(new BstNode(data));                  return root.getLeft();              } else {                  tmpNode = root.getLeft();              }          } else {              System.out.print(" [R]");              if(root.getRight() == null) {                  root.setRight(new BstNode(data));                  return root.getRight();              } else {                  tmpNode = root.getRight();              }          }            return insertNode(tmpNode, data);      }        public Integer findHeight(){            return getNodeHeight(this.root);      }        private Integer getNodeHeight(BstNode node) {            if(node == null) {              return -1;          }            return Math.max(getNodeHeight(node.getLeft()), getNodeHeight(node.getRight()))+1;      }        public static void main(String a[]) {            BinarySearchTreeImpl bst = new BinarySearchTreeImpl();          bst.insert(10);          bst.insert(20);          bst.insert(21);          bst.insert(8);          bst.insert(6);          bst.insert(16);          bst.insert(23);          bst.insert(2);          System.out.println("-------------------");          System.out.println("Height of the tree: "+bst.findHeight());      }  } | | |
| |  | | --- | | **Output:** | | [input: 10] -> inserted: 10  [input: 20] ->10 [R] -> inserted: 20  [input: 21] ->10 [R] ->20 [R] -> inserted: 21  [input: 8] ->10 [L] -> inserted: 8  [input: 6] ->10 [L] ->8 [L] -> inserted: 6  [input: 16] ->10 [R] ->20 [L] -> inserted: 16  [input: 23] ->10 [R] ->20 [R] ->21 [R] -> inserted: 23  [input: 2] ->10 [L] ->8 [L] ->6 [L] -> inserted: 2  -------------------  Height of the tree: 3 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Find height of a Binary Search Tree (BST)  |  | | --- | | **Description:** | | For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root. This example shows how to find height of a binary search tree.  Here is an example picture of binary search tree (BST):  Binary Search Tree |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/height-of-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.javacoffee.ds;    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;      }  } | |  |  | | --- | | **BinarySearchTreeImpl** | | [?](https://www.java2novice.com/java-interview-programs/height-of-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | package com.javacoffee.ds;    public class BinarySearchTreeImpl {        private BstNode root;        public boolean isEmpty() {            return (this.root == null);      }        public void insert(Integer data) {            System.out.print("[input: "+data+"]");          if(root == null) {              this.root = new BstNode(data);              System.out.println(" -> inserted: "+data);              return;          }            insertNode(this.root, data);          System.out.print(" -> inserted: "+data);          System.out.println();      }        private BstNode insertNode(BstNode root, Integer data) {            BstNode tmpNode = null;          System.out.print(" ->"+root.getData());          if(root.getData() >= data) {              System.out.print(" [L]");              if(root.getLeft() == null) {                  root.setLeft(new BstNode(data));                  return root.getLeft();              } else {                  tmpNode = root.getLeft();              }          } else {              System.out.print(" [R]");              if(root.getRight() == null) {                  root.setRight(new BstNode(data));                  return root.getRight();              } else {                  tmpNode = root.getRight();              }          }            return insertNode(tmpNode, data);      }        public Integer findHeight(){            return getNodeHeight(this.root);      }        private Integer getNodeHeight(BstNode node) {            if(node == null) {              return -1;          }            return Math.max(getNodeHeight(node.getLeft()), getNodeHeight(node.getRight()))+1;      }        public static void main(String a[]) {            BinarySearchTreeImpl bst = new BinarySearchTreeImpl();          bst.insert(10);          bst.insert(20);          bst.insert(21);          bst.insert(8);          bst.insert(6);          bst.insert(16);          bst.insert(23);          bst.insert(2);          System.out.println("-------------------");          System.out.println("Height of the tree: "+bst.findHeight());      }  } | | |
| |  | | --- | | **Output:** | | [input: 10] -> inserted: 10  [input: 20] ->10 [R] -> inserted: 20  [input: 21] ->10 [R] ->20 [R] -> inserted: 21  [input: 8] ->10 [L] -> inserted: 8  [input: 6] ->10 [L] ->8 [L] -> inserted: 6  [input: 16] ->10 [R] ->20 [L] -> inserted: 16  [input: 23] ->10 [R] ->20 [R] ->21 [R] -> inserted: 23  [input: 2] ->10 [L] ->8 [L] ->6 [L] -> inserted: 2  -------------------  Height of the tree: 3 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Implement Binary Search Tree (BST) pre-order traversal (depth first).  |  | | --- | | **Description:** | | For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root. This example shows the implementation of a binary search tree pre-order traversal (depth first). What is pre-order traversal (depth first)? Tree traversal means we visiting all nodes in the tree, visiting means either of accessing node data or processing node data. Traversal can be specified by the order of visiting 3 nodes, ie current node, left subtree and right subtree. In pre-order traversal, first we visit the current node, then left subtree and then right subtree. In our current example we use recursive approach to implement pre-order traversal.  Binary Search Tree (pre-order traversal)  Pre-order: F, B, A, D, C, E, G, I, H.  Here is an example picture of binary search tree (BST) for our example code:  Binary Search Tree  Here is the steps to implement pre-order traversal:   1. Start with root node. 2. Check if the current node is empty / null. 3. Display the data part of the root (or current node). 4. Traverse the left subtree by recursively calling the pre-order function. 5. Traverse the right subtree by recursively calling the pre-order function. |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/pre-order-traversal-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.javacoffee.ds;    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;      }  } | |  |  | | --- | | **BinarySearchTreeImpl** | | [?](https://www.java2novice.com/java-interview-programs/pre-order-traversal-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82 | package com.javacoffee.ds;    import java.util.LinkedList;  import java.util.Queue;    public class BinarySearchTreeImpl {        private BstNode root;        public boolean isEmpty() {            return (this.root == null);      }        public void insert(Integer data) {            System.out.print("[input: "+data+"]");          if(root == null) {              this.root = new BstNode(data);              System.out.println(" -> inserted: "+data);              return;          }            insertNode(this.root, data);          System.out.print(" -> inserted: "+data);          System.out.println();      }        private BstNode insertNode(BstNode root, Integer data) {            BstNode tmpNode = null;          System.out.print(" ->"+root.getData());          if(root.getData() >= data) {              System.out.print(" [L]");              if(root.getLeft() == null) {                  root.setLeft(new BstNode(data));                  return root.getLeft();              } else {                  tmpNode = root.getLeft();              }          } else {              System.out.print(" [R]");              if(root.getRight() == null) {                  root.setRight(new BstNode(data));                  return root.getRight();              } else {                  tmpNode = root.getRight();              }          }            return insertNode(tmpNode, data);      }        public void preOrderTraversal() {          doPreOrder(this.root);      }        private void doPreOrder(BstNode root) {            if(root == null) return;          System.out.print(root.getData()+" ");          doPreOrder(root.getLeft());          doPreOrder(root.getRight());      }        public static void main(String a[]) {            BinarySearchTreeImpl bst = new BinarySearchTreeImpl();          bst.insert(8);          bst.insert(10);          bst.insert(14);          bst.insert(3);          bst.insert(6);          bst.insert(7);          bst.insert(1);          bst.insert(4);          bst.insert(13);          System.out.println("\n-------------------");          System.out.println("Pre Order Traversal");          bst.preOrderTraversal();      }  } | | |
| |  | | --- | | **Output:** | | [input: 8] -> inserted: 8  [input: 10] ->8 [R] -> inserted: 10  [input: 14] ->8 [R] ->10 [R] -> inserted: 14  [input: 3] ->8 [L] -> inserted: 3  [input: 6] ->8 [L] ->3 [R] -> inserted: 6  [input: 7] ->8 [L] ->3 [R] ->6 [R] -> inserted: 7  [input: 1] ->8 [L] ->3 [L] -> inserted: 1  [input: 4] ->8 [L] ->3 [R] ->6 [L] -> inserted: 4  [input: 13] ->8 [R] ->10 [R] ->14 [L] -> inserted: 13  -------------------  Pre Order Traversal  8 3 1 6 4 7 10 14 13 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Implement Binary Search Tree (BST) in-order traversal (depth first).  |  | | --- | | **Description:** | | For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root. This example shows the implementation of a binary search tree in-order traversal (depth first). What is in-order traversal (depth first)? Tree traversal means we visiting all nodes in the tree, visiting means either of accessing node data or processing node data. Traversal can be specified by the order of visiting 3 nodes, ie current node, left subtree and right subtree. In in-order traversal, first we visit the left subtree, then current node and then right subtree. In-order traversal gives data in the sorted order. In our current example we use recursive approach to implement in-order traversal.  Binary Search Tree (in-order traversal)  In-order: A, B, C, D, E, F, G, H, I.  Here is an example picture of binary search tree (BST) for our example code:  Binary Search Tree  Here is the steps to implement in-order traversal:   1. Start with root node. 2. Check if the current node is empty / null. 3. Traverse the left subtree by recursively calling the in-order function. 4. Display the data part of the root (or current node). 5. Traverse the right subtree by recursively calling the in-order function. |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/in-order-traversal-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.javacoffee.ds;    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;      }  } | |  |  | | --- | | **BinarySearchTreeImpl** | | [?](https://www.java2novice.com/java-interview-programs/in-order-traversal-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82 | package com.javacoffee.ds;    import java.util.LinkedList;  import java.util.Queue;    public class BinarySearchTreeImpl {        private BstNode root;        public boolean isEmpty() {            return (this.root == null);      }        public void insert(Integer data) {            System.out.print("[input: "+data+"]");          if(root == null) {              this.root = new BstNode(data);              System.out.println(" -> inserted: "+data);              return;          }            insertNode(this.root, data);          System.out.print(" -> inserted: "+data);          System.out.println();      }        private BstNode insertNode(BstNode root, Integer data) {            BstNode tmpNode = null;          System.out.print(" ->"+root.getData());          if(root.getData() >= data) {              System.out.print(" [L]");              if(root.getLeft() == null) {                  root.setLeft(new BstNode(data));                  return root.getLeft();              } else {                  tmpNode = root.getLeft();              }          } else {              System.out.print(" [R]");              if(root.getRight() == null) {                  root.setRight(new BstNode(data));                  return root.getRight();              } else {                  tmpNode = root.getRight();              }          }            return insertNode(tmpNode, data);      }        public void inOrderTraversal() {          doInOrder(this.root);      }        private void doInOrder(BstNode root) {            if(root == null) return;          doInOrder(root.getLeft());          System.out.print(root.getData()+" ");          doInOrder(root.getRight());      }        public static void main(String a[]) {            BinarySearchTreeImpl bst = new BinarySearchTreeImpl();          bst.insert(8);          bst.insert(10);          bst.insert(14);          bst.insert(3);          bst.insert(6);          bst.insert(7);          bst.insert(1);          bst.insert(4);          bst.insert(13);          System.out.println("\n-------------------");          System.out.println("In Order Traversal");          bst.inOrderTraversal();      }  } | | |
| |  | | --- | | **Output:** | | [input: 8] -> inserted: 8  [input: 10] ->8 [R] -> inserted: 10  [input: 14] ->8 [R] ->10 [R] -> inserted: 14  [input: 3] ->8 [L] -> inserted: 3  [input: 6] ->8 [L] ->3 [R] -> inserted: 6  [input: 7] ->8 [L] ->3 [R] ->6 [R] -> inserted: 7  [input: 1] ->8 [L] ->3 [L] -> inserted: 1  [input: 4] ->8 [L] ->3 [R] ->6 [L] -> inserted: 4  [input: 13] ->8 [R] ->10 [R] ->14 [L] -> inserted: 13  -------------------  In Order Traversal  1 3 4 6 7 8 10 13 14 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Program: Implement Binary Search Tree (BST) post-order traversal (depth first).  |  | | --- | | **Description:** | | For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root. This example shows the implementation of a binary search tree post-order traversal (depth first). What is post-order traversal (depth first)? Tree traversal means we visiting all nodes in the tree, visiting means either of accessing node data or processing node data. Traversal can be specified by the order of visiting 3 nodes, ie current node, left subtree and right subtree. In post-order traversal, first we visit the left subtree, then the right subtree, and then current node. In our current example we use recursive approach to implement post-order traversal.  Binary Search Tree (post-order traversal)  Post-order: A, C, E, D, B, H, I, G, F.  Here is an example picture of binary search tree (BST) for our example code:  Binary Search Tree  Here is the steps to implement post-order traversal:   1. Start with root node. 2. Check if the current node is empty / null. 3. Traverse the left subtree by recursively calling the post-order function. 4. Traverse the right subtree by recursively calling the post-order function. 5. Display the data part of the root (or current node). |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/post-order-traversal-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.javacoffee.ds;    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;      }  } | |  |  | | --- | | **BinarySearchTreeImpl** | | [?](https://www.java2novice.com/java-interview-programs/post-order-traversal-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82 | package com.javacoffee.ds;    import java.util.LinkedList;  import java.util.Queue;    public class BinarySearchTreeImpl {        private BstNode root;        public boolean isEmpty() {            return (this.root == null);      }        public void insert(Integer data) {            System.out.print("[input: "+data+"]");          if(root == null) {              this.root = new BstNode(data);              System.out.println(" -> inserted: "+data);              return;          }            insertNode(this.root, data);          System.out.print(" -> inserted: "+data);          System.out.println();      }        private BstNode insertNode(BstNode root, Integer data) {            BstNode tmpNode = null;          System.out.print(" ->"+root.getData());          if(root.getData() >= data) {              System.out.print(" [L]");              if(root.getLeft() == null) {                  root.setLeft(new BstNode(data));                  return root.getLeft();              } else {                  tmpNode = root.getLeft();              }          } else {              System.out.print(" [R]");              if(root.getRight() == null) {                  root.setRight(new BstNode(data));                  return root.getRight();              } else {                  tmpNode = root.getRight();              }          }            return insertNode(tmpNode, data);      }        public void postOrderTraversal() {          doPostOrder(this.root);      }        private void doPostOrder(BstNode root) {            if(root == null) return;          doPostOrder(root.getLeft());          doPostOrder(root.getRight());          System.out.print(root.getData()+" ");      }        public static void main(String a[]) {            BinarySearchTreeImpl bst = new BinarySearchTreeImpl();          bst.insert(8);          bst.insert(10);          bst.insert(14);          bst.insert(3);          bst.insert(6);          bst.insert(7);          bst.insert(1);          bst.insert(4);          bst.insert(13);          System.out.println("\n-------------------");          System.out.println("Post Order Traversal");          bst.postOrderTraversal();      }  } | | |
| |  | | --- | | **Output:** | | [input: 8] -> inserted: 8  [input: 10] ->8 [R] -> inserted: 10  [input: 14] ->8 [R] ->10 [R] -> inserted: 14  [input: 3] ->8 [L] -> inserted: 3  [input: 6] ->8 [L] ->3 [R] -> inserted: 6  [input: 7] ->8 [L] ->3 [R] ->6 [R] -> inserted: 7  [input: 1] ->8 [L] ->3 [L] -> inserted: 1  [input: 4] ->8 [L] ->3 [R] ->6 [L] -> inserted: 4  [input: 13] ->8 [R] ->10 [R] ->14 [L] -> inserted: 13  -------------------  Post Order Traversal  1 4 7 6 3 13 14 10 8 | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Program: How to check the given Binary Tree is Binary Search Tree (BST) or not?**   |  | | --- | | **Description:** | | In a Binary Tree, each node can have at most two nodes. For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root.  There are various ways to validate Binary Search Tree. One of the simple way is: The in-order traversal of a binary search tree results natural order. So, we can do in-order traversal and check for natural order. If the order sorted, then it is binary search tree. We will give this implementation in the coming pages.  In this page we follow different approach. We will set min and max value for each node and validate node data against min and max value. The same approach will continue for each left and right sub binary search tree in recursive way.  Here is the steps to validate binary search tree:   1. Start with root node. In this case root node data min & max values can be extreme integer ranges. Pass min value as Integer.MIN\_VALUE and max value as Integer.MAX\_VALUE. 2. Make sure node data is falling under min & max values. 3. Along with the above check, make sure the left and right sub trees are also go through similar checks. 4. Make a recursive call on left node with no change in min value and node data as max value. 5. Make a recursive call on right node with node data as min value and no change in max value. 6. Check the the code for better understanding. |  |  | | --- | | **IsBinarySearchTree** | | [?](https://www.java2novice.com/java-interview-programs/validate-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package com.javacoffee.ds;    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;          }      }  } | |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/validate-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package com.javacoffee.ds;    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;      }  } | | |
| |  | | --- | | **Input Tree:** | | 3  / \  2 6  / \ / \  1 4 5 7 | |
| |  | | --- | | **Input Tree in main method:** | | [?](https://www.java2novice.com/java-interview-programs/validate-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | 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));  } | | |
| |  | | --- | | **Output:** | | false | |
| |  | | --- | | **Input Tree:** | | 8  / \  3 10  / \ \  1 6 14 | |
| |  | | --- | | **Input Tree in main method:** | | [?](https://www.java2novice.com/java-interview-programs/validate-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public static void main(String a[]) {        BstNode root = new BstNode(8);      // left sub tree      BstNode node\_3 = new BstNode(3); root.setLeft(node\_3);      BstNode node\_1 = new BstNode(1); node\_3.setLeft(node\_1);      BstNode node\_6 = new BstNode(6); node\_3.setRight(node\_6);      // right sub tree      BstNode node\_10 = new BstNode(10); root.setRight(node\_10);      BstNode node\_14 = new BstNode(14); node\_10.setRight(node\_14);        IsBinarySearchTree ibsTree = new IsBinarySearchTree();      System.out.println(ibsTree.isBinarySearchTree(root));  } | | |
| |  | | --- | | **Output:** | | true | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Program: How to delete a node from Binary Search Tree (BST)?**   |  | | --- | | **Description:** | | In a Binary Tree, each node can have at most two nodes. For a binary tree to be a binary search tree (BST), the data of all the nodes in the left sub-tree of the root node should be less than or equals to the data of the root. The data of all the nodes in the right subtree of the root node should be greater than the data of the root.  Deleting a node from Binary search tree is little complicated compare to inserting a node. It includes two steps:   1. Search the node with given value. 2. Delete the node.   The algorithm has 3 cases while deleting node:   1. Node to be deleted has is a leaf node (no children). 2. Node to be deleted has one child (eight left or right child node). 3. Node to be deleted has two nodes.   We will use simple recursion to find the node and delete it from the tree.  Here is the steps to delete a node from binary search tree:  ***Case 1: Node to be deleted has is a leaf node (no children).***   1. This is very simple implementation. First find the node reference with given value. 2. Set corresponding link of the parent node to null. With this the node to be deleted lost its connectivity and eligible for garbage collection.   ***Case 2: Node to be deleted has one child (eight left or right child node).***   1. First find the node reference with given value. 2. Take the reference of the child node and assign its reference to the corresponding link of the parent node. With this the node to be deleted lost its connectivity and eligible for garbage collection.   ***Case 3: Node to be deleted has two nodes.***   1. It is little complicated process. 2. First find the node reference with given value. 3. Find the minimum/maximum value of the right/left sub tree. 4. Replace the node value with the minimum/maximum value. 5. Now delete the minimum/maximum value from the nodes right/left sub tree.   We will use below binary tree for our code output:  Binary Search Tree |  |  | | --- | | **BinarySearchTreeImpl** | | [?](https://www.java2novice.com/java-interview-programs/delete-node-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139 | package com.javacoffee.ds;    import java.util.LinkedList;  import java.util.Queue;    public class BinarySearchTreeImpl {        private BstNode root;        public boolean isEmpty() {            return (this.root == null);      }        public BstNode getRoot() {          return this.root;      }        public void insert(Integer data) {            System.out.print("[input: "+data+"]");          if(root == null) {              this.root = new BstNode(data);              System.out.println(" -> inserted: "+data);              return;          }            insertNode(this.root, data);          System.out.print(" -> inserted: "+data);          System.out.println();      }        private BstNode insertNode(BstNode root, Integer data) {            BstNode tmpNode = null;          System.out.print(" ->"+root.getData());          if(root.getData() >= data) {              System.out.print(" [L]");              if(root.getLeft() == null) {                  root.setLeft(new BstNode(data));                  return root.getLeft();              } else {                  tmpNode = root.getLeft();              }          } else {              System.out.print(" [R]");              if(root.getRight() == null) {                  root.setRight(new BstNode(data));                  return root.getRight();              } else {                  tmpNode = root.getRight();              }          }            return insertNode(tmpNode, data);      }        public void delete(Integer data) {            deleteNode(this.root, data);      }        private BstNode deleteNode(BstNode root, Integer data) {            if(root == null) return root;            if(data < root.getData()) {              root.setLeft(deleteNode(root.getLeft(), data));          } else if(data > root.getData()) {              root.setRight(deleteNode(root.getRight(), data));          } else {              // node with no leaf nodes              if(root.getLeft() == null && root.getRight() == null) {                  System.out.println("deleting "+data);                  return null;              } else if(root.getLeft() == null) {                  // node with one node (no left node)                  System.out.println("deleting "+data);                  return root.getRight();              } else if(root.getRight() == null) {                  // node with one node (no right node)                  System.out.println("deleting "+data);                  return root.getLeft();              } else {                  // nodes with two nodes                  // search for min number in right sub tree                  Integer minValue = minValue(root.getRight());                  root.setData(minValue);                  root.setRight(deleteNode(root.getRight(), minValue));                  System.out.println("deleting "+data);              }          }            return root;      }        private Integer minValue(BstNode node) {            if(node.getLeft() != null) {              return minValue(node.getLeft());          }          return node.getData();      }        public void inOrderTraversal() {          doInOrder(this.root);      }        private void doInOrder(BstNode root) {            if(root == null) return;          doInOrder(root.getLeft());          System.out.print(root.getData()+" ");          doInOrder(root.getRight());      }        public static void main(String a[]) {            BinarySearchTreeImpl bst = new BinarySearchTreeImpl();          bst.insert(8);          bst.insert(10);          bst.insert(14);          bst.insert(3);          bst.insert(6);          bst.insert(7);          bst.insert(1);          bst.insert(4);          bst.insert(13);          System.out.println("-------------------");          System.out.println("In Order Traversal");          bst.inOrderTraversal();          System.out.println();          bst.delete(13);          bst.inOrderTraversal();          System.out.println();          bst.delete(14);          bst.inOrderTraversal();      }  } | |  |  | | --- | | **BstNode** | | [?](https://www.java2novice.com/java-interview-programs/delete-node-binary-search-tree-bst/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | package com.javacoffee.ds;    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 void setData(Integer data) {          this.data = data;      }  } | | |
| |  | | --- | | **Output:** | | [input: 8] -> inserted: 8  [input: 10] ->8 [R] -> inserted: 10  [input: 14] ->8 [R] ->10 [R] -> inserted: 14  [input: 3] ->8 [L] -> inserted: 3  [input: 6] ->8 [L] ->3 [R] -> inserted: 6  [input: 7] ->8 [L] ->3 [R] ->6 [R] -> inserted: 7  [input: 1] ->8 [L] ->3 [L] -> inserted: 1  [input: 4] ->8 [L] ->3 [R] ->6 [L] -> inserted: 4  [input: 13] ->8 [R] ->10 [R] ->14 [L] -> inserted: 13  -------------------  In Order Traversal  1 3 4 6 7 8 10 13 14  deleting 13  1 3 4 6 7 8 10 14  deleting 14  1 3 4 6 7 8 10 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Program: Write a program to find common integers between two sorted arrays.  |  | | --- | | **Description:** | | Write a program to find common integers between two sorted arrays. Both arrays are sorted in ASC order. Both arrays doesn't have any duplicate numbers. Make sure you navigate through both arrays only once. |  |  | | --- | | **CommonElementsInArr.java** | | [?](https://www.java2novice.com/java-interview-programs/common-number-in-two-arrays/)   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package com.javacoffee.algos;    public class CommonElementsInArr {        public static void main(String a[]) {            int[] input1 = {2,7,17,19,20,45,56,159,239};          int[] intput2 = {7,12,15,19,22,34,55,150,159};          int index1 = 0;          int index2 = 0;          while(true) {              if(index1 >= input1.length || index2 >= intput2.length) {                  break;              }              if(input1[index1] == intput2[index2]) {                  System.out.print(input1[index1]);                  System.out.print("  ");                  index1 += 1;              } else if(input1[index1] < intput2[index2]) {                  index1 += 1;              } else {                  index2 += 1;              }          }        }  } | | |
| |  | | --- | | **Output:** | | 7 19 159 | |