Heap implementation

```
1.  proceduredesign_min_heap
2.  Array arr: of size n => array of elements
3.  // call min_heapify procedure for each element of the array to form min heap
4.  repeat for (k = n/2 ; k >= 1 ; k--)
5.      call procedure min_heapify (arr, k);
6.  proceduremin_heapify (vararr[ ] , var k, varn)
7.  {
8.  varleft_child  = 2*k;
9.  varright_child = 2*k+1;
10. var smallest;
11. if(left_child<= n and arr[left_child ] <arr[ k ] )
12. smallest = left_child;
13. else
14. smallest = k;
15. if(right_child<= n and arr[right_child ] <arr[smallest] )
16. smallest = right_child;
17. if(smallest != k)
18.     {
19. swaparr[ k ] and  arr[ smallest ]);
20. callmin_heapify (arr, smallest, n);
21.     }
22. }
```

MinHeapJavaImplementation.java

```
1.  // import required classes and packages
2.  packagejavaTpoint.javacodes;
3.
4.  importjava.util.Scanner;
5.
6.  // create class MinHeap to construct Min heap in Java
7.  classMinHeap {
8.      // declare array and variables
9.  privateint[] heapData;
10. privateintsizeOfHeap;
11. privateintheapMaxSize;
```

```
12.
13. private static final int FRONT = 1;
14.     //use constructor to initialize heapData array
15. publicMinHeap(intheapMaxSize)  {
16. this.heapMaxSize = heapMaxSize;
17. this.sizeOfHeap = 0;
18. heapData = new int[this.heapMaxSize + 1];
19. heapData[0] = Integer.MIN_VALUE;
20.     }
21.
22.     // create getParentPos() method that returns parent position for the node
23. privateintgetParentPosition(int position)  {
24. return position / 2;
25.     }
26.
27.     // create getLeftChildPosition() method that returns the position of left child

28. privateintgetLeftChildPosition(int position)  {
29. return (2 * position);
30.     }
31.
32.     // create getRightChildPosition() method that returns the position of right c
    hild
33. privateintgetRightChildPosition(int position)  {
34. return (2 * position) + 1;
35.     }
36.
37.     // checks whether the given node is leaf or not
38. privatebooleancheckLeaf(int position)  {
39. if (position >= (sizeOfHeap / 2) && position <= sizeOfHeap) {
40. return true;
41.         }
42. return false;
43.     }
44.
45.     // create swapNodes() method that perform swapping of the given nodes o
    f the heap
```

```
46.     // firstNode and secondNode are the positions of the nodes
47. private void swap(intfirstNode, intsecondNode)  {
48. int temp;
49. temp = heapData[firstNode];
50. heapData[firstNode] = heapData[secondNode];
51. heapData[secondNode] = temp;
52.     }
53.
54.     // create minHeapify() method to heapify the node for maintaining the hea
    p property
55. private void minHeapify(int position)  {
56.
57.         //check whether the given node is non-
    leaf and greater than its right and left child
58. if (!checkLeaf(position)) {
59. if (heapData[position] >heapData[getLeftChildPosition(position)] || heapData[
    position] >heapData[getRightChildPosition(position)]) {
60.
61.                 // swap with left child and then heapify the left child
62. if (heapData[getLeftChildPosition(position)] <heapData[getRightChildPosition(
    position)]) {
63. swap(position, getLeftChildPosition(position));
64. minHeapify(getLeftChildPosition(position));
65.                 }
66.
67.                 // Swap with the right child and heapify the right child
68. else {
69. swap(position, getRightChildPosition(position));
70. minHeapify(getRightChildPosition(position));
71.                 }
72.             }
73.         }
74.     }
75.
76.     // create insertNode() method to insert element in the heap
77. public void insertNode(int data)  {
78. if (sizeOfHeap>= heapMaxSize) {
```

```
79. return;
80.         }
81. heapData[++sizeOfHeap] = data;
82. int current = sizeOfHeap;
83.
84. while (heapData[current] <heapData[getParentPosition(current)]) {
85. swap(current, getParentPosition(current));
86. current = getParentPosition(current);
87.         }
88.     }
89.
90.     // crreatedisplayHeap() method to print the data of the heap
91. public void displayHeap()  {
92. System.out.println("PARENT NODE" + "\t" + "LEFT CHILD NODE" + "\t" + "RIG
    HT CHILD NODE");
93. for (int k = 1; k <= sizeOfHeap / 2; k++) {
94. System.out.print(" " + heapData[k] + "\t\t" + heapData[2 * k] + "\t\t" + heapD
    ata[2 * k + 1]);
95. System.out.println();
96.         }
97.     }
98.
99.    // create designMinHeap() method to construct min heap
100.         public void designMinHeap()  {
101.         for (int position = (sizeOfHeap / 2); position >= 1; position--) {
102.         minHeapify(position);
103.                 }
104.             }
105.
106.             // create removeRoot() method for removing minimum element from
    the heap
107.         publicintremoveRoot()  {
108.         intpopElement = heapData[FRONT];
109.         heapData[FRONT] = heapData[sizeOfHeap--];
110.         minHeapify(FRONT);
111.         returnpopElement;
112.             }
```

```
113.        }
114.
115.        // create MinHeapJavaImplementation class to create heap in Java
116.        classMinHeapJavaImplementation{
117.
118.            // main() method start
119.        public static void main(String[] arg)  {
120.            // declare variable
121.            intheapSize;
122.
123.            // create scanner class object
124.            Scanner sc = new Scanner(System.in);
125.
126.            System.out.println("Enter the size of Min Heap");
127.            heapSize = sc.nextInt();
128.
129.            MinHeapheapObj = new MinHeap(heapSize);
130.
131.            for(inti = 1; i<= heapSize; i++) {
132.                System.out.print("Enter "+i+" element: ");
133.                int data = sc.nextInt();
134.                heapObj.insertNode(data);
135.            }
136.
137.                // close scanner class obj
138.        sc.close();
139.
140.                //construct a min heap from given data
141.        heapObj.designMinHeap();
142.
143.                //display the min heap data
144.        System.out.println("The Min Heap is ");
145.        heapObj.displayHeap();
146.
147.                //removing the root node from the heap
148.        System.out.println("After removing the minimum element(Root Node) "
    +heapObj.removeRoot()+", Min heap is:");
```

```
149.        heapObj.displayHeap();
150.
151.            }
152.        }
```