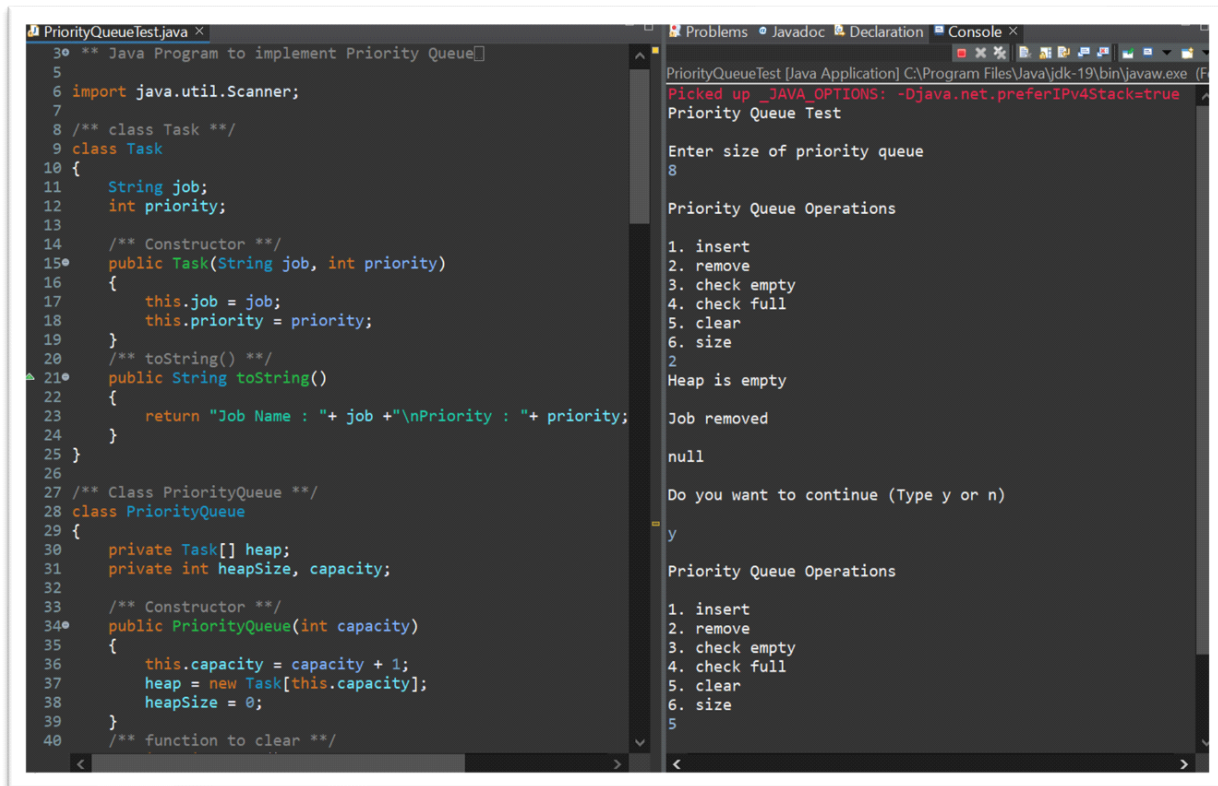


LABSHEET - 3

AIM 1: Understanding the concepts of Stack, Queue , Linked List (10points)

- Write a Java program to implement a priority queue ADT.



The screenshot shows an IDE with two panes. The left pane displays the source code for `PriorityQueueTest.java`. The right pane shows the console output of the program.

```
30 /** Java Program to implement Priority Queue */
5
6 import java.util.Scanner;
7
8 /** class Task */
9 class Task
10 {
11     String job;
12     int priority;
13
14     /** Constructor */
15     public Task(String job, int priority)
16     {
17         this.job = job;
18         this.priority = priority;
19     }
20     /** toString() */
21     public String toString()
22     {
23         return "Job Name : " + job + "\nPriority : " + priority;
24     }
25 }
26
27 /** Class PriorityQueue */
28 class PriorityQueue
29 {
30     private Task[] heap;
31     private int heapSize, capacity;
32
33     /** Constructor */
34     public PriorityQueue(int capacity)
35     {
36         this.capacity = capacity + 1;
37         heap = new Task[this.capacity];
38         heapSize = 0;
39     }
40     /** function to clear */
```

The console output shows the following sequence of events:

```
PriorityQueueTest [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (F
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true
Priority Queue Test

Enter size of priority queue
8

Priority Queue Operations

1. insert
2. remove
3. check empty
4. check full
5. clear
6. size
2
Heap is empty

Job removed

null

Do you want to continue (Type y or n)
y

Priority Queue Operations

1. insert
2. remove
3. check empty
4. check full
5. clear
6. size
5
```

Code:

```
package q;
/**
 ** Java Program to implement Priority Queue
 **/

import java.util.Scanner;

/** class Task
 **/class Task
{
    String job;

    int priority;
```

```

    /** Constructor **/
    public Task(String job, int priority)
    {
        this.job =
        job;
        this.priority
        = priority;
    }
    /**
    toString()
    **/ public
    String
    toString()
    {
        return "Job Name : "+ job +"\nPriority : "+ priority;
    }
}

/** Class
PriorityQueue
**/class
PriorityQueue
{
    private Task[] heap;
    private int heapSize, capacity;

    /** Constructor **/
    public PriorityQueue(int capacity)
    {
        this.capacity =
        capacity + 1; heap =
        new
        Task[this.capacity];
        heapSize = 0;
    }
    /** function
    to clear **/
    public void
    clear()
    {
        heap = new
        Task[capacity];
        heapSize = 0;
    }
}

```

```

/** function to check if
empty */public boolean
isEmpty()
{
    return heapSize == 0;
}
/** function to check
if full */public
boolean isFull()
{
    return heapSize == capacity - 1;
}
/** function to get Size */

public int size()
{
    return heapSize;
}
/** function to insert task */
public void insert(String job, int priority)
{
    Task newJob = new Task(job, priority);

    heap[++heapSize
] = newJob;int
pos = heapSize;
while (pos != 1 && newJob.priority > heap[pos/2].priority)
{
    heap[pos] =
    heap[pos/2];
    pos /=2;
}
    heap[pos] = newJob;
}
/** function to
remove task */
public Task remove()
{
    int
    parent,
    child;
    Task
    item,
    temp;
    if

```

```

        (isEmpty
y() )
    {
        System.out.println("Heap
is empty");return null;
    }

    item = heap[1];
    temp = heap[heapSize--];

    parent = 1;
    child = 2;
    while (child <= heapSize)
    {
        if (child < heapSize && heap[child].priority <
heap[child +1].priority)
            child++;
        if (temp.priority >=
            heap[child].priority)break;

        heap[parent] =
            heap[child];
        parent = child;
        child *= 2;

    }
    heap[parent] = temp;

    return item;
}
}

/** Class
PriorityQueueTest
**/public class
PriorityQueueTest
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Priority Queue Test\n");

        System.out.println("Enter size of priority queue ");
        PriorityQueue pq = new PriorityQueue(scan.nextInt()
);
    }
}

```

```

char ch;
/* Perform Priority Queue
operations */do
{
    System.out.println("\nPriority Queue
Operations\n");System.out.println("1.
insert"); System.out.println("2.
remove"); System.out.println("3. check
empty"); System.out.println("4. check
full"); System.out.println("5. clear");
System.out.println("6. size");

    int choice =
scan.nextInt();
    switch (choice)
    {
    case 1 :
        System.out.println("Enter job name and
priority");pq.insert(scan.next(),
scan.nextInt() );
        b
r
e
a
k
;

        c
a
s
e

    2

:
        System.out.println("\nJob removed \n\n"+
pq.remove());break;
    case 3 :
        System.out.println("\nEmpty Status : "+
pq.isEmpty() );break;
    case 4 :
        System.out.println("\nFull Status : "+ pq.isFull() );

```

```

        break
    ;case 5 :
        System.out.println("\nPriority Queue
        Cleared");pq.clear();
        break
    ;case 6 :
        System.out.println("\nSize = "+ pq.size()
        );break;
    default :
        System.out.println("Wrong Entry \n
        ");break;
}

System.out.println("\nDo you want to continue (Type y or n)
\n");ch = scan.next().charAt(0);
} while (ch == 'Y' || ch == 'y');
}
}

```

- Write Java programs to implement the following using a singly linked list. (a) Stack ADT (b) QueueADT

- Stack ADK using Singly Linked List.

```

stackADK.java x
21
22 // print Top element of Stack
23 System.out.printf("\nTop element is %d\n",
24 obj.peek());
25
26 // Delete top element of Stack
27 obj.pop();
28 obj.pop();
29
30 // print Stack elements
31 obj.display();
32
33 // print Top element of Stack
34 System.out.printf("\nTop element is %d\n",
35 obj.peek());
36 }
37 }
38
39 // Create Stack Using Linked list
40 class StackUsingLinkedList {
41
42 // A linked list node
43 private class Node {
44
45 int data; // integer data
46 Node next; // reference to next Node
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 }
140
141 }
142
143 }
144
145 }
146
147 }
148
149 }
150
151 }
152
153 }
154
155 }
156
157 }
158
159 }
160
161 }
162
163 }
164
165 }
166
167 }
168
169 }
170
171 }
172
173 }
174
175 }
176
177 }
178
179 }
180
181 }
182
183 }
184
185 }
186
187 }
188
189 }
190
191 }
192
193 }
194
195 }
196
197 }
198
199 }
200
201 }
202
203 }
204
205 }
206
207 }
208
209 }
210
211 }
212
213 }
214
215 }
216
217 }
218
219 }
220
221 }
222
223 }
224
225 }
226
227 }
228
229 }
230
231 }
232
233 }
234
235 }
236
237 }
238
239 }
240
241 }
242
243 }
244
245 }
246
247 }
248
249 }
250
251 }
252
253 }
254
255 }
256
257 }
258
259 }
260
261 }
262
263 }
264
265 }
266
267 }
268
269 }
270
271 }
272
273 }
274
275 }
276
277 }
278
279 }
280
281 }
282
283 }
284
285 }
286
287 }
288
289 }
290
291 }
292
293 }
294
295 }
296
297 }
298
299 }
300
301 }
302
303 }
304
305 }
306
307 }
308
309 }
310
311 }
312
313 }
314
315 }
316
317 }
318
319 }
320
321 }
322
323 }
324
325 }
326
327 }
328
329 }
330
331 }
332
333 }
334
335 }
336
337 }
338
339 }
340
341 }
342
343 }
344
345 }
346
347 }
348
349 }
350
351 }
352
353 }
354
355 }
356
357 }
358
359 }
360
361 }
362
363 }
364
365 }
366
367 }
368
369 }
370
371 }
372
373 }
374
375 }
376
377 }
378
379 }
380
381 }
382
383 }
384
385 }
386
387 }
388
389 }
390
391 }
392
393 }
394
395 }
396
397 }
398
399 }
400
401 }
402
403 }
404
405 }
406
407 }
408
409 }
410
411 }
412
413 }
414
415 }
416
417 }
418
419 }
420
421 }
422
423 }
424
425 }
426
427 }
428
429 }
430
431 }
432
433 }
434
435 }
436
437 }
438
439 }
440
441 }
442
443 }
444
445 }
446
447 }
448
449 }
450
451 }
452
453 }
454
455 }
456
457 }
458
459 }
460
461 }
462
463 }
464
465 }
466
467 }
468
469 }
470
471 }
472
473 }
474
475 }
476
477 }
478
479 }
480
481 }
482
483 }
484
485 }
486
487 }
488
489 }
490
491 }
492
493 }
494
495 }
496
497 }
498
499 }
500
501 }
502
503 }
504
505 }
506
507 }
508
509 }
510
511 }
512
513 }
514
515 }
516
517 }
518
519 }
520
521 }
522
523 }
524
525 }
526
527 }
528
529 }
530
531 }
532
533 }
534
535 }
536
537 }
538
539 }
540
541 }
542
543 }
544
545 }
546
547 }
548
549 }
550
551 }
552
553 }
554
555 }
556
557 }
558
559 }
560
561 }
562
563 }
564
565 }
566
567 }
568
569 }
570
571 }
572
573 }
574
575 }
576
577 }
578
579 }
580
581 }
582
583 }
584
585 }
586
587 }
588
589 }
590
591 }
592
593 }
594
595 }
596
597 }
598
599 }
600
601 }
602
603 }
604
605 }
606
607 }
608
609 }
610
611 }
612
613 }
614
615 }
616
617 }
618
619 }
620
621 }
622
623 }
624
625 }
626
627 }
628
629 }
630
631 }
632
633 }
634
635 }
636
637 }
638
639 }
640
641 }
642
643 }
644
645 }
646
647 }
648
649 }
650
651 }
652
653 }
654
655 }
656
657 }
658
659 }
660
661 }
662
663 }
664
665 }
666
667 }
668
669 }
670
671 }
672
673 }
674
675 }
676
677 }
678
679 }
680
681 }
682
683 }
684
685 }
686
687 }
688
689 }
690
691 }
692
693 }
694
695 }
696
697 }
698
699 }
700
701 }
702
703 }
704
705 }
706
707 }
708
709 }
710
711 }
712
713 }
714
715 }
716
717 }
718
719 }
720
721 }
722
723 }
724
725 }
726
727 }
728
729 }
730
731 }
732
733 }
734
735 }
736
737 }
738
739 }
740
741 }
742
743 }
744
745 }
746
747 }
748
749 }
750
751 }
752
753 }
754
755 }
756
757 }
758
759 }
760
761 }
762
763 }
764
765 }
766
767 }
768
769 }
770
771 }
772
773 }
774
775 }
776
777 }
778
779 }
780
781 }
782
783 }
784
785 }
786
787 }
788
789 }
790
791 }
792
793 }
794
795 }
796
797 }
798
799 }
800
801 }
802
803 }
804
805 }
806
807 }
808
809 }
810
811 }
812
813 }
814
815 }
816
817 }
818
819 }
820
821 }
822
823 }
824
825 }
826
827 }
828
829 }
830
831 }
832
833 }
834
835 }
836
837 }
838
839 }
840
841 }
842
843 }
844
845 }
846
847 }
848
849 }
850
851 }
852
853 }
854
855 }
856
857 }
858
859 }
860
861 }
862
863 }
864
865 }
866
867 }
868
869 }
870
871 }
872
873 }
874
875 }
876
877 }
878
879 }
880
881 }
882
883 }
884
885 }
886
887 }
888
889 }
890
891 }
892
893 }
894
895 }
896
897 }
898
899 }
900
901 }
902
903 }
904
905 }
906
907 }
908
909 }
910
911 }
912
913 }
914
915 }
916
917 }
918
919 }
920
921 }
922
923 }
924
925 }
926
927 }
928
929 }
930
931 }
932
933 }
934
935 }
936
937 }
938
939 }
940
941 }
942
943 }
944
945 }
946
947 }
948
949 }
950
951 }
952
953 }
954
955 }
956
957 }
958
959 }
960
961 }
962
963 }
964
965 }
966
967 }
968
969 }
970
971 }
972
973 }
974
975 }
976
977 }
978
979 }
980
981 }
982
983 }
984
985 }
986
987 }
988
989 }
990
991 }
992
993 }
994
995 }
996
997 }
998
999 }
1000
1001 }
1002
1003 }
1004
1005 }
1006
1007 }
1008
1009 }
1010
1011 }
1012
1013 }
1014
1015 }
1016
1017 }
1018
1019 }
1020
1021 }
1022
1023 }
1024
1025 }
1026
1027 }
1028
1029 }
1030
1031 }
1032
1033 }
1034
1035 }
1036
1037 }
1038
1039 }
1040
1041 }
1042
1043 }
1044
1045 }
1046
1047 }
1048
1049 }
1050
1051 }
1052
1053 }
1054
1055 }
1056
1057 }
1058
1059 }
1060
1061 }
1062
1063 }
1064
1065 }
1066
1067 }
1068
1069 }
1070
1071 }
1072
1073 }
1074
1075 }
1076
1077 }
1078
1079 }
1080
1081 }
1082
1083 }
1084
1085 }
1086
1087 }
1088
1089 }
1090
1091 }
1092
1093 }
1094
1095 }
1096
1097 }
1098
1099 }
1100
1101 }
1102
1103 }
1104
1105 }
1106
1107 }
1108
1109 }
1110
1111 }
1112
1113 }
1114
1115 }
1116
1117 }
1118
1119 }
1120
1121 }
1122
1123 }
1124
1125 }
1126
1127 }
1128
1129 }
1130
1131 }
1132
1133 }
1134
1135 }
1136
1137 }
1138
1139 }
1140
1141 }
1142
1143 }
1144
1145 }
1146
1147 }
1148
1149 }
1150
1151 }
1152
1153 }
1154
1155 }
1156
1157 }
1158
1159 }
1160
1161 }
1162
1163 }
1164
1165 }
1166
1167 }
1168
1169 }
1170
1171 }
1172
1173 }
1174
1175 }
1176
1177 }
1178
1179 }
1180
1181 }
1182
1183 }
1184
1185 }
1186
1187 }
1188
1189 }
1190
1191 }
1192
1193 }
1194
1195 }
1196
1197 }
1198
1199 }
1200
1201 }
1202
1203 }
1204
1205 }
1206
1207 }
1208
1209 }
1210
1211 }
1212
1213 }
1214
1215 }
1216
1217 }
1218
1219 }
1220
1221 }
1222
1223 }
1224
1225 }
1226
1227 }
1228
1229 }
1230
1231 }
1232
1233 }
1234
1235 }
1236
1237 }
1238
1239 }
1240
1241 }
1242
1243 }
1244
1245 }
1246
1247 }
1248
1249 }
1250
1251 }
1252
1253 }
1254
1255 }
1256
1257 }
1258
1259 }
1260
1261 }
1262
1263 }
1264
1265 }
1266
1267 }
1268
1269 }
1270
1271 }
1272
1273 }
1274
1275 }
1276
1277 }
1278
1279 }
1280
1281 }
1282
1283 }
1284
1285 }
1286
1287 }
1288
1289 }
1290
1291 }
1292
1293 }
1294
1295 }
1296
1297 }
1298
1299 }
1300
1301 }
1302
1303 }
1304
1305 }
1306
1307 }
1308
1309 }
1310
1311 }
1312
1313 }
1314
1315 }
1316
1317 }
1318
1319 }
1320
1321 }
1322
1323 }
1324
1325 }
1326
1327 }
1328
1329 }
1330
1331 }
1332
1333 }
1334
1335 }
1336
1337 }
1338
1339 }
1340
1341 }
1342
1343 }
1344
1345 }
1346
1347 }
1348
1349 }
1350
1351 }
1352
1353 }
1354
1355 }
1356
1357 }
1358
1359 }
1360
1361 }
1362
1363 }
1364
1365 }
1366
1367 }
1368
1369 }
1370
1371 }
1372
1373 }
1374
1375 }
1376
1377 }
1378
1379 }
1380
1381 }
1382
1383 }
1384
1385 }
1386
1387 }
1388
1389 }
1390
1391 }
1392
1393 }
1394
1395 }
1396
1397 }
1398
1399 }
1400
1401 }
1402
1403 }
1404
1405 }
1406
1407 }
1408
1409 }
1410
1411 }
1412
1413 }
1414
1415 }
1416
1417 }
1418
1419 }
1420
1421 }
1422
1423 }
1424
1425 }
1426
1427 }
1428
1429 }
1430
1431 }
1432
1433 }
1434
1435 }
1436
1437 }
1438
1439 }
1440
1441 }
1442
1443 }
1444
1445 }
1446
1447 }
1448
1449 }
1450
1451 }
1452
1453 }
1454
1455 }
1456
1457 }
1458
1459 }
1460
1461 }
1462
1463 }
1464
1465 }
1466
1467 }
1468
1469 }
1470
1471 }
1472
1473 }
1474
1475 }
1476
1477 }
1478
1479 }
1480
1481 }
1482
1483 }
1484
1485 }
1486
1487 }
1488
1489 }
1490
1491 }
1492
1493 }
1494
1495 }
1496
1497 }
1498
1499 }
1500
1501 }
1502
1503 }
1504
1505 }
1506
1507 }
1508
1509 }
1510
1511 }
1512
1513 }
1514
1515 }
1516
1517 }
1518
1519 }
1520
1521 }
1522
1523 }
1524
1525 }
1526
1527 }
1528
1529 }
1530
1531 }
1532
1533 }
1534
1535 }
1536
1537 }
1538
1539 }
1540
1541 }
1542
1543 }
1544
1545 }
1546
1547 }
1548
1549 }
1550
1551 }
1552
1553 }
1554
1555 }
1556
1557 }
1558
1559 }
1560
1561 }
1562
1563 }
1564
1565 }
1566
1567 }
1568
1569 }
1570
1571 }
1572
1573 }
1574
1575 }
1576
1577 }
1578
1579 }
1580
1581 }
1582
1583 }
1584
1585 }
1586
1587 }
1588
1589 }
1590
1591 }
1592
1593 }
1594
1595 }
1596
1597 }
1598
1599 }
1600
1601 }
1602
1603 }
1604
1605 }
1606
1607 }
1608
1609 }
1610
1611 }
1612
1613 }
1614
1615 }
1616
1617 }
1618
1619 }
1620
1621 }
1622
1623 }
1624
1625 }
1626
1627 }
1628
1629 }
1630
1631 }
1632
1633 }
1634
1635 }
1636
1637 }
1638
1639 }
1640
1641 }
1642
1643 }
1644
1645 }
1646
1647 }
1648
1649 }
1650
1651 }
1652
1653 }
1654
1655 }
1656
1657 }
1658
1659 }
1660
1661 }
1662
1663 }
1664
1665 }
1666
1667 }
1668
1669 }
1670
1671 }
1672
1673 }
1674
1675 }
1676
1677 }
1678
1679 }
1680
1681 }
1682
1683 }
1684
1685 }
1686
1687 }
1688
1689 }
1690
1691 }
1692
1693 }
1694
1695 }
1696
1697 }
1698
1699 }
1700
1701 }
1702
1703 }
1704
1705 }
1706
1707 }
1708
1709 }
1710
1711 }
1712
1713 }
1714
1715 }
1716
1717 }
1718
1719 }
1720
1721 }
1722
1723 }
1724
1725 }
1726
1727 }
1728
1729 }
1730
1731 }
1732
1733 }
1734
1735 }
1736
1737 }
1738
1739 }
1740
1741 }
1742
1743 }
1744
1745 }
1746
1747 }
1748
1749 }
1750
1751 }
1752
1753 }
1754
1755 }
1756
1757 }
1758
1759 }
1760
1761 }
1762
1763 }
1764
1765 }
1766
1767 }
1768
1769 }
1770
1771 }
1772
1773 }
1774
1775 }
1776
1777 }
1778
1779 }
1780
1781 }
1782
1783 }
1784
1785 }
1786
1787 }
1788
1789 }
1790
1791 }
1792
1793 }
1794
1795 }
1796
1797 }
1798
1799 }
1800
1801 }
1802
1803 }
1804
1805 }
1806
1807 }
1808
1809 }
1810
1811 }
1812
1813 }
1814
1815 }
1816
1817 }
1818
1819 }
1820
1821 }
1822
1823 }
1824
1825 }
1826
1827 }
1828
1829 }
1830
1831 }
1832
1833 }
1834
1835 }
1836
1837 }
1838
1839 }
1840
1841 }
1842
1843 }
1844
1845 }
1846
1847 }
1848
1849 }
1850
1851 }
1852
1853 }
1854
1855 }
1856
1857 }
1858
1859 }
1860
1861 }
1862
1863 }
1864
1865 }
1866
1867 }
1868
1869 }
1870
1871 }
1872
1873 }
1874
1875 }
1876
1877 }
1878
1879 }
1880
1881 }
1882
1883 }
1884
1885 }
1886
1887 }
1888
1889 }
1890
1891 }
1892
1893 }
1894
1895 }
1896
1897 }
1898
1899 }
1900
1901 }
1902
1903 }
1904
1905 }
1906
1907 }
1908
1909 }
1910
1911 }
1912
1913 }
1914
1915 }
1916
1917 }
1918
1919 }
1920
1921 }
1922
1923 }
1924
1925 }
1926
1927 }
1928
1929 }
1930
1931 }
1932
1933 }
1934
1935 }
1936
1937 }
1938
1939 }
1940
1941 }
1942
1943 }
1944
1945 }
1946
1947 }
1948
1949 }
1950
1951 }
1952
1953 }
1954
1955 }
1956
1957 }
1958
1959 }
1960
1961 }
1962
1963 }
1964
1965 }
1966
1967 }
1968
1969 }
1970
1971 }
1972
1973 }
1974
1975 }
1976
1977 }
1978
1979 }
1980
1981 }
1982
1983 }
1984
1985 }
1986
1987 }
1988
1989 }
1990
1991 }
1992
1993 }
1994
1995 }
1996
1997 }
1998
1999 }
2000
2001 }
2002
2003 }
2004
2005 }
2006
2007 }
2008
2009 }
2010
2011 }
2012
2013 }
2014
2015 }
2016
2017 }
2018
2019 }
2020
2021 }
2022
2023 }
2024
2025 }
2026
2027 }
2028
2029 }
2030
2031 }
2032
2033 }
2034
2035 }
2036
2037 }
2038
2039 }
2040
2041 }
2042
2043 }
2044
2045 }
2046
2047 }
2048
2049 }
2050
2051 }
2052
2053 }
2054
2055 }
2056
2057 }
2058
2059 }
2060
2061 }
2062
2063 }
2064
2065 }
2066
2067 }
2068
2069 }
2070
2071 }
2072
2073 }
2074
2075 }
2076
2077 }
2078
2079 }
2080
2081 }
2082
2083 }
2084
2085 }
2086
2087 }
2088
2089 }
2090
2091 }
2092
2093 }
2094
2095 }
2096
2097 }
2098
2099 }
2100
2101 }
2102
2103 }
2104
2105 }
2106
2107 }
2108
2109 }
2110
2111 }
2112
2113 }
2114
2115 }
2116
2117 }
2118
2119 }
2120
2121 }
2122
2123 }
2124
2125 }
2126
2127 }
2128
2129 }
2130
2131 }
2132
2133 }
2134
2135 }
2136
2137 }
2138
2139 }
2140
2141 }
2142
2143 }
2144
2145 }
2146
2147 }
2148
2149 }
2150
2151 }
2152
2153 }
2154
2155 }
2156
2157 }
2158
2159 }
2160
2161 }
2162
2163 }
2164
2165 }
2166
2167 }
2168
2169 }
2170
2171 }
2172
2173 }
2174
2175 }
2176
2177 }
2178
2179 }
2180
2181 }
2182
2183 }
2184
2185 }
2186
2187 }
2188
2189 }
2190
2191 }
2192
2193 }
2194
2195 }
2196
2197 }
2198
2199 }
2200
2201 }
2202
2203 }
2204
2205 }
2206
2207 }
2208
2209 }
2210
2211 }
2212
2213 }
2214
2215 }
2216
2217 }
2218
2219 }
2220
2221 }
2222
2223 }
2224
2225 }
2226
2227 }
2228
2229 }
2230
2231 }
2232
2233 }
2234
2235 }
2236
2237 }
2238
2239 }
2240
2241 }
2242
2243 }
2244
2245 }
2246
2247 }
2248
2249 }
2250
2251 }
2252
2253 }
2254
2255 }
2256
2257 }
2258
2259 }
2260
2261 }
2262
2263 }
2264
2265 }
2266
2267 }
2268
2269 }
2270
2271 }
2272
2273 }
2274
2275 }
2276
2277 }
2278
2279 }
2280
2281 }
2282
2283 }
2284
2285 }
2286
2287 }
2288
2289 }
2290
2291 }
2292
2293 }
2294
2295 }
2296
2297 }
2298
2299 }
2300
2301 }
2302
2303 }
2304
2305 }
2306
2307 }
2308
2309 }
2310
2311 }
2312
2313 }
2314
2315 }
2316
2317 }
2318
2319 }
2320
2321 }
2322
2323 }
2324
2325 }
2326
2327 }
2328
2329 }
2330
2331 }
2332
2333 }
2334
2335 }
2336
2337 }
2338
2339 }
2340
2341 }
2342
2343 }
2344
2345 }
2346
2347 }
2348
2349 }
2350
2351 }
2352
2353 }
2354
2355 }
2356
2357 }
2358
2359 }
2360
2361 }
2362
2363 }
2364
2365 }
2366
2367 }
2368
2369 }
2370
2371 }
2372
2373 }
2374
2375 }
237
```

```

// Java program to Implement a stack
// using singly linked list
// import package
import static java.lang.System.exit;

// Driver code
class stackADK
{
    public static void main(String[] args)
    {
        // create Object of Implementing class
        StackUsingLinkedList obj
            = new StackUsingLinkedList();
        // insert Stack
        valueobj.push(11);
        obj.push(22);
        obj.push(33);
        obj.push(44);

        // print Stack
        elements
        obj.display();

        // print Top element of Stack
        System.out.printf("\nTop element is
        %d\n",
                        obj.peak());

        // Delete top
        element of Stack
        obj.pop();
        obj.pop();

        // print Stack
        elements
        obj.display();

        // print Top element of Stack
        System.out.printf("\nTop element is
        %d\n",
                        obj.peak());
    }
}

```

```

// Create Stack Using
Linked list
class
StackUsingLinkedList {

    // A
    linked
    list node
    private
    class Node
    {

        int data; // integer data
        Node link; // reference variable Node type
    }

    // create global top reference
    variable globalNode top;
    // Constructor
    StackUsingLinkedList() { this.top = null; }

    // Utility function to add an element x
    in the stack
    public void push(int x) //
    insert at the beginning
    {
        // create new node temp and
        allocate memory
        Node temp = new
        Node();

        // check if stack (heap) is full. Then inserting an
        // element would lead to
        stack overflow
        if (temp ==
        null) {
            System.out.print("\nHeap Overflow");

            return;
        }

        // initialize data into temp
        data field
        temp.data = x;

        // put top reference into
        temp link
        temp.link = top;

        // update top
        reference
        top =
        temp;
    }
}

```



```

}

// Utility function to check if the stack is empty or
// not
public boolean isEmpty() { return top == null; }

// Utility function to return top element
in a stack
public int peek()
{
    // check for
    empty stack
    if
    (!isEmpty()) {
        return top.data;
    }
    else {
        System.out.println("Stack
        is empty"); return -1;
    }
}

// Utility function to pop top element
from the stack
public void pop() // remove
at the beginning
{
    // check for stack
    underflow
    if (top
    == null) {
        System.out.print("\nStack
        Underflow"); return;
    }

    // update the top pointer to point to
    the next node
    top = (top).link;
}

public void display()
{
    // check for stack underflow
    if (top == null) {
        System.out.printf("\nStack
        Underflow"); exit(1);
    }
    else {

```

```

Node temp = top;
while (temp != null)
{

    // print node data
    System.out.print(temp.data);

    // assign temp link to temp
    temp = temp.link;
    if(temp != null)
        System.out.print(" ->
        ");
    }
}
}
}

```

- Queue ADK using Singly Linked List.

```

1  class Node {
2      private static int front, rear, capacity;
3      private static int Node[];
4
5      Node(int size) {
6          front = rear = 0;
7          capacity = size;
8          Node = new int[capacity];
9      }
10
11     // insert an element into the Node
12     static void NodeEnNode(int item) {
13         // check if the Node is full
14         if (capacity == rear) {
15             System.out.printf("\nNode is full\n");
16             return;
17         }
18
19         // insert element at the rear
20         else {
21             Node[rear] = item;
22             rear++;
23         }
24         return;
25     }
26
27     //remove an element from the Node
28     static void NodeDeNode() {
29         // check if Node is empty
30         if (front == rear) {
31             System.out.printf("\nNode is empty\n");
32             return;
33         }
34
35         // shift elements to the right by one place uptil rear
36         else {
37             for (int i = 0; i < rear - 1; i++) {
38                 Node[i] = Node[i + 1];
39             }
40
41             // set Node[rear] to 0
42             if (rear < capacity)
43                 Node[rear] = 0;
44
45             // decrement rear
46             rear--;
47         }
48         return;
49     }
50
51     // print Node elements
52     static void NodeDisplay()
53     {
54         int i;
55         if (front == rear) {
56             System.out.printf("Node is Empty\n");
57             return;
58         }
59
60         // traverse front to rear and print elements
61         for (i = front; i < rear; i++) {
62             System.out.printf(" %d = ", Node[i]);
63         }
64         return;
65     }
66
67     // print front of Node
68     static void NodeFront()
69     {
70         if (front == rear) {
71             System.out.printf("Node is Empty\n");
72             return;
73         }
74         System.out.printf("\nFront Element of the Node: %d", Node[front]);
75         return;
76     }
77 }
78
79 public class Main {
80     public static void main(String[] args) {
81         // Create a Node of capacity 4
82         Node q = new Node(4);
83
84         System.out.println("Initial Node:");
85         // print Node elements
86         q.NodeDisplay();
87
88         // Inserting elements in the Node
89         q.NodeEnNode(10);
90         q.NodeEnNode(30);
91         q.NodeEnNode(50);
92         q.NodeEnNode(70);
93
94         // print Node elements
95         System.out.println("Node after EnNode Operation:");
96         q.NodeDisplay();
97
98         // print front of the Node
99         q.NodeFront();
100
101         // Insert element in the Node
102         q.NodeEnNode(90);
103
104         // print Node elements
105     }

```

```

106     q.NodeDisplay();
107
108     q.NodeDeNode();
109     q.NodeDeNode();
110     System.out.printf("\nNode after two delNode operations:");
111
112     // print Node elements
113     q.NodeDisplay();
114
115     // print front of the Node
116     q.NodeFront();
117 }
118 }

```

CPU Time: 0.16 sec(s), Memory: 33652 kilobyte(s)

compiled and executed in 0.969 sec(s)

```

Initial Node:
Node is Empty
Node after EnNode Operation:
10 = 30 = 50 = 70 =
Front Element of the Node: 10
Node is full
10 = 30 = 50 = 70 =
Node after two delNode operations: 50 = 70 =
Front Element of the Node: 50

```

Code:

```

class Node {
    private static int front, rear,
    capacity; private static int Node[];

    Node(int size) {
        front = rear =
        0; capacity =
        size;
        Node = new int[capacity];
    }

    // insert an element into the Node
    static void NodeEnNode(int item) {
        // check if the Node is full
        if (capacity == rear) {
            System.out.printf("\nNode is
            full\n"); return;
        }

        // insert element at the rear
        else {
            Node[rear] =
            item; rear++;
        }
        return;
    }

    //remove an element from the Node
    static void NodeDeNode() {
        // check if Node is empty
        if (front == rear) {

```

```

        System.out.printf("\nNode is
        empty\n");return;
    }

    // shift elements to the right by one place
    upto rear else {
        for (int i = 0; i < rear
            - 1; i++) {Node[i] =
            Node[i + 1];
        }

    // set Node[rear] to 0
        if (rear
            <
            capac
            ity)
            Node[
            rear]
            = 0;

        //
        decrem
        ent
        rear
        rear--
        ;
    }
    return;
}

// print Node
elements
static void
NodeDisplay()
{
    int i;
    if (front == rear) {
        System.out.printf("Node is
        Empty\n");return;
    }
}

```

```

        // traverse front to rear and print
        elementsfor (i = front; i < rear;
i++) {
            System.out.printf(" %d = ", Node[i]);
        }
        return;
    }

    // print
    front of
    Node static
    void
    NodeFront()
    {
        if (front == rear) {
            System.out.printf("Node is
            Empty\n");return;
        }
        System.out.printf("\nFront Element of the Node: %d", Node[front]);

        return;
    }
}

public class Main {
    public static void main(String[] args) {
        // Create a Node of
        capacity 4Node q =
        new Node(4);

        System.out.println("Initial Node:");
        // print
        Node
        elements
        q.NodeDisp
        lay();

        // inserting elements
        in the Node
        q.NodeEnNode(10);
        q.NodeEnNode(30);
        q.NodeEnNode(50);
        q.NodeEnNode(70);

        // print Node elements
    }
}

```

```

        System.out.println("Node after EnNode
        Operation:");q.NodeDisplay();

        // print front of
        the Node
        q.NodeFront();

        // insert element in
        the Node
        q.NodeEnNode(90);

        // print
        Node
        elements
        q.NodeDispl
        ay();

        q.No
        deDe
        Node
        ();
        q.No
        deDe
        Node
        ();
        System.out.printf("\nNode after two deNode operations:");

        // print
        Node
        elements
        q.NodeDispl
        ay();

        // print front of
        the Node
        q.NodeFront();
    }
}

```