

CEGEP VANIER COLLEGE

CENTRE FOR CONTINUING EDUCATION

Programming Algorithms and Patterns

420-930-VA

Teacher: Samir Chebbine

Lab 3

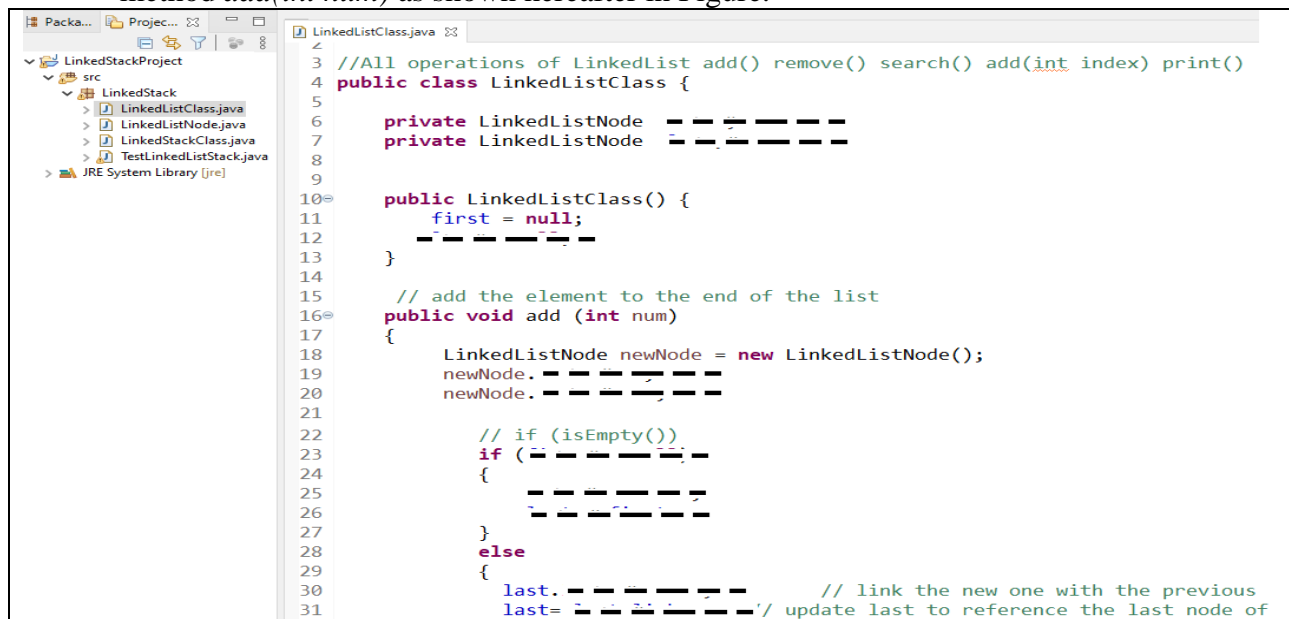
Jun 07, 2024

Lab 3: Stacks, Queues and Binary Trees

Complete all these following programs as explained during **classes**. All *missing coding statements* were provided there with explanation. **Create and Submit** a Word file **Lab3ProgramminAlgorithmsandPatternsYourName.docx** which includes **output screenshots** for every Java Project. Submit the Java projects too.

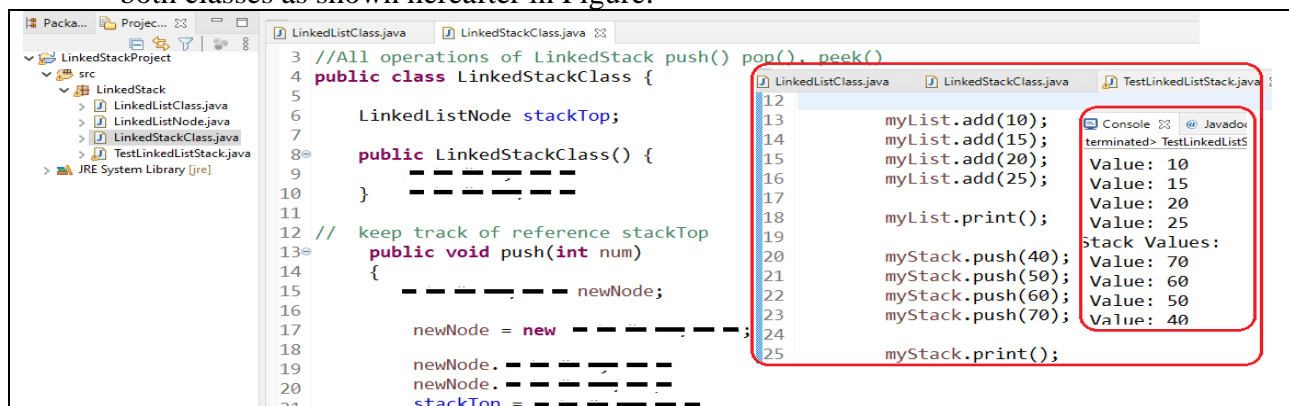
1. Stack Data Structure: Create a project named **LinkedStackProject**

- a) Create *LinkedStackProject* using Eclipse IDE for developing *user-defined methods* of Stack and linked list operations. Create *LinkedListClass.java* that includes user-defined method *add(int num)* as shown hereafter in Figure.



```
3 //All operations of LinkedList add() remove() search() add(int index) print()
4 public class LinkedListClass {
5
6     private LinkedListNode [] nodes;
7     private LinkedListNode first;
8
9
10    public LinkedListClass() {
11        first = null;
12    }
13
14    // add the element to the end of the list
15    public void add (int num)
16    {
17        LinkedListNode newNode = new LinkedListNode();
18        newNode.data = num;
19        newNode.next = null;
20
21        // if (isEmpty())
22        if (first == null)
23        {
24            first = newNode;
25        }
26        else
27        {
28            last.next = newNode; // link the new one with the previous
29            last = newNode; // update last to reference the last node of
30        }
31    }
```

- b) Create *LinkedStackClass.java* that includes user-defined method *push(int num)* and test both classes as shown hereafter in Figure.



```
3 //All operations of LinkedStack push() pop() peek()
4 public class LinkedStackClass {
5
6     LinkedListNode stackTop;
7
8     public LinkedStackClass() {
9         stackTop = null;
10    }
11
12    // keep track of reference stackTop
13    public void push(int num)
14    {
15        LinkedListNode newNode;
16
17        newNode = new LinkedListNode();
18
19        newNode.data = num;
20        newNode.next = stackTop;
21        stackTop = newNode;
22    }
23
24    public void print()
25    {
26        // print the stack
27        LinkedListNode current = stackTop;
28        while (current != null)
29        {
30            System.out.print(current.data + " ");
31            current = current.next;
32        }
33        System.out.println();
34    }
35
36    public void pop()
37    {
38        // pop the stack
39        if (stackTop != null)
40        {
41            stackTop = stackTop.next;
42        }
43    }
44
45    public int peek()
46    {
47        // peek the stack
48        if (stackTop != null)
49        {
50            return stackTop.data;
51        }
52        return -1;
53    }
54
55    public boolean isEmpty()
56    {
57        return stackTop == null;
58    }
59
60    public boolean isFull()
61    {
62        return false;
63    }
64
65    public void clear()
66    {
67        stackTop = null;
68    }
69
70    public void reset()
71    {
72        stackTop = null;
73    }
74
75    public void destroy()
76    {
77        stackTop = null;
78    }
79
80    public void printStack()
81    {
82        print();
83    }
84
85    public void popStack()
86    {
87        pop();
88    }
89
90    public int peekStack()
91    {
92        peek();
93    }
94
95    public boolean isEmptyStack()
96    {
97        isEmpty();
98    }
99
100   public boolean isFullStack()
101   {
102       isFull();
103   }
104
105   public void clearStack()
106   {
107       clear();
108   }
109
110   public void resetStack()
111   {
112       reset();
113   }
114
115   public void destroyStack()
116   {
117       destroy();
118   }
119
120   public void printStack()
121   {
122       print();
123   }
124
125   public void popStack()
126   {
127       pop();
128   }
129
130   public int peekStack()
131   {
132       peek();
133   }
134
135   public boolean isEmptyStack()
136   {
137       isEmpty();
138   }
139
140   public boolean isFullStack()
141   {
142       isFull();
143   }
144
145   public void clearStack()
146   {
147       clear();
148   }
149
150   public void resetStack()
151   {
152       reset();
153   }
154
155   public void destroyStack()
156   {
157       destroy();
158   }
159
160   public void printStack()
161   {
162       print();
163   }
164
165   public void popStack()
166   {
167       pop();
168   }
169
170   public int peekStack()
171   {
172       peek();
173   }
174
175   public boolean isEmptyStack()
176   {
177       isEmpty();
178   }
179
180   public boolean isFullStack()
181   {
182       isFull();
183   }
184
185   public void clearStack()
186   {
187       clear();
188   }
189
190   public void resetStack()
191   {
192       reset();
193   }
194
195   public void destroyStack()
196   {
197       destroy();
198   }
199
200   public void printStack()
201   {
202       print();
203   }
204
205   public void popStack()
206   {
207       pop();
208   }
209
210   public int peekStack()
211   {
212       peek();
213   }
214
215   public boolean isEmptyStack()
216   {
217       isEmpty();
218   }
219
220   public boolean isFullStack()
221   {
222       isFull();
223   }
224
225   public void clearStack()
226   {
227       clear();
228   }
229
230   public void resetStack()
231   {
232       reset();
233   }
234
235   public void destroyStack()
236   {
237       destroy();
238   }
239
240   public void printStack()
241   {
242       print();
243   }
244
245   public void popStack()
246   {
247       pop();
248   }
249
250   public int peekStack()
251   {
252       peek();
253   }
254
255   public boolean isEmptyStack()
256   {
257       isEmpty();
258   }
259
260   public boolean isFullStack()
261   {
262       isFull();
263   }
264
265   public void clearStack()
266   {
267       clear();
268   }
269
270   public void resetStack()
271   {
272       reset();
273   }
274
275   public void destroyStack()
276   {
277       destroy();
278   }
279
280   public void printStack()
281   {
282       print();
283   }
284
285   public void popStack()
286   {
287       pop();
288   }
289
290   public int peekStack()
291   {
292       peek();
293   }
294
295   public boolean isEmptyStack()
296   {
297       isEmpty();
298   }
299
300   public boolean isFullStack()
301   {
302       isFull();
303   }
304
305   public void clearStack()
306   {
307       clear();
308   }
309
310   public void resetStack()
311   {
312       reset();
313   }
314
315   public void destroyStack()
316   {
317       destroy();
318   }
319
320   public void printStack()
321   {
322       print();
323   }
324
325   public void popStack()
326   {
327       pop();
328   }
329
330   public int peekStack()
331   {
332       peek();
333   }
334
335   public boolean isEmptyStack()
336   {
337       isEmpty();
338   }
339
340   public boolean isFullStack()
341   {
342       isFull();
343   }
344
345   public void clearStack()
346   {
347       clear();
348   }
349
350   public void resetStack()
351   {
352       reset();
353   }
354
355   public void destroyStack()
356   {
357       destroy();
358   }
359
360   public void printStack()
361   {
362       print();
363   }
364
365   public void popStack()
366   {
367       pop();
368   }
369
370   public int peekStack()
371   {
372       peek();
373   }
374
375   public boolean isEmptyStack()
376   {
377       isEmpty();
378   }
379
380   public boolean isFullStack()
381   {
382       isFull();
383   }
384
385   public void clearStack()
386   {
387       clear();
388   }
389
390   public void resetStack()
391   {
392       reset();
393   }
394
395   public void destroyStack()
396   {
397       destroy();
398   }
399
400   public void printStack()
401   {
402       print();
403   }
404
405   public void popStack()
406   {
407       pop();
408   }
409
410   public int peekStack()
411   {
412       peek();
413   }
414
415   public boolean isEmptyStack()
416   {
417       isEmpty();
418   }
419
420   public boolean isFullStack()
421   {
422       isFull();
423   }
424
425   public void clearStack()
426   {
427       clear();
428   }
429
430   public void resetStack()
431   {
432       reset();
433   }
434
435   public void destroyStack()
436   {
437       destroy();
438   }
439
440   public void printStack()
441   {
442       print();
443   }
444
445   public void popStack()
446   {
447       pop();
448   }
449
450   public int peekStack()
451   {
452       peek();
453   }
454
455   public boolean isEmptyStack()
456   {
457       isEmpty();
458   }
459
460   public boolean isFullStack()
461   {
462       isFull();
463   }
464
465   public void clearStack()
466   {
467       clear();
468   }
469
470   public void resetStack()
471   {
472       reset();
473   }
474
475   public void destroyStack()
476   {
477       destroy();
478   }
479
480   public void printStack()
481   {
482       print();
483   }
484
485   public void popStack()
486   {
487       pop();
488   }
489
490   public int peekStack()
491   {
492       peek();
493   }
494
495   public boolean isEmptyStack()
496   {
497       isEmpty();
498   }
499
500   public boolean isFullStack()
501   {
502       isFull();
503   }
504
505   public void clearStack()
506   {
507       clear();
508   }
509
510   public void resetStack()
511   {
512       reset();
513   }
514
515   public void destroyStack()
516   {
517       destroy();
518   }
519
520   public void printStack()
521   {
522       print();
523   }
524
525   public void popStack()
526   {
527       pop();
528   }
529
530   public int peekStack()
531   {
532       peek();
533   }
534
535   public boolean isEmptyStack()
536   {
537       isEmpty();
538   }
539
540   public boolean isFullStack()
541   {
542       isFull();
543   }
544
545   public void clearStack()
546   {
547       clear();
548   }
549
550   public void resetStack()
551   {
552       reset();
553   }
554
555   public void destroyStack()
556   {
557       destroy();
558   }
559
560   public void printStack()
561   {
562       print();
563   }
564
565   public void popStack()
566   {
567       pop();
568   }
569
570   public int peekStack()
571   {
572       peek();
573   }
574
575   public boolean isEmptyStack()
576   {
577       isEmpty();
578   }
579
580   public boolean isFullStack()
581   {
582       isFull();
583   }
584
585   public void clearStack()
586   {
587       clear();
588   }
589
590   public void resetStack()
591   {
592       reset();
593   }
594
595   public void destroyStack()
596   {
597       destroy();
598   }
599
600   public void printStack()
601   {
602       print();
603   }
604
605   public void popStack()
606   {
607       pop();
608   }
609
610   public int peekStack()
611   {
612       peek();
613   }
614
615   public boolean isEmptyStack()
616   {
617       isEmpty();
618   }
619
620   public boolean isFullStack()
621   {
622       isFull();
623   }
624
625   public void clearStack()
626   {
627       clear();
628   }
629
630   public void resetStack()
631   {
632       reset();
633   }
634
635   public void destroyStack()
636   {
637       destroy();
638   }
639
640   public void printStack()
641   {
642       print();
643   }
644
645   public void popStack()
646   {
647       pop();
648   }
649
650   public int peekStack()
651   {
652       peek();
653   }
654
655   public boolean isEmptyStack()
656   {
657       isEmpty();
658   }
659
660   public boolean isFullStack()
661   {
662       isFull();
663   }
664
665   public void clearStack()
666   {
667       clear();
668   }
669
670   public void resetStack()
671   {
672       reset();
673   }
674
675   public void destroyStack()
676   {
677       destroy();
678   }
679
680   public void printStack()
681   {
682       print();
683   }
684
685   public void popStack()
686   {
687       pop();
688   }
689
690   public int peekStack()
691   {
692       peek();
693   }
694
695   public boolean isEmptyStack()
696   {
697       isEmpty();
698   }
699
700   public boolean isFullStack()
701   {
702       isFull();
703   }
704
705   public void clearStack()
706   {
707       clear();
708   }
709
710   public void resetStack()
711   {
712       reset();
713   }
714
715   public void destroyStack()
716   {
717       destroy();
718   }
719
720   public void printStack()
721   {
722       print();
723   }
724
725   public void popStack()
726   {
727       pop();
728   }
729
730   public int peekStack()
731   {
732       peek();
733   }
734
735   public boolean isEmptyStack()
736   {
737       isEmpty();
738   }
739
740   public boolean isFullStack()
741   {
742       isFull();
743   }
744
745   public void clearStack()
746   {
747       clear();
748   }
749
750   public void resetStack()
751   {
752       reset();
753   }
754
755   public void destroyStack()
756   {
757       destroy();
758   }
759
760   public void printStack()
761   {
762       print();
763   }
764
765   public void popStack()
766   {
767       pop();
768   }
769
770   public int peekStack()
771   {
772       peek();
773   }
774
775   public boolean isEmptyStack()
776   {
777       isEmpty();
778   }
779
780   public boolean isFullStack()
781   {
782       isFull();
783   }
784
785   public void clearStack()
786   {
787       clear();
788   }
789
790   public void resetStack()
791   {
792       reset();
793   }
794
795   public void destroyStack()
796   {
797       destroy();
798   }
799
800   public void printStack()
801   {
802       print();
803   }
804
805   public void popStack()
806   {
807       pop();
808   }
809
810   public int peekStack()
811   {
812       peek();
813   }
814
815   public boolean isEmptyStack()
816   {
817       isEmpty();
818   }
819
820   public boolean isFullStack()
821   {
822       isFull();
823   }
824
825   public void clearStack()
826   {
827       clear();
828   }
829
830   public void resetStack()
831   {
832       reset();
833   }
834
835   public void destroyStack()
836   {
837       destroy();
838   }
839
840   public void printStack()
841   {
842       print();
843   }
844
845   public void popStack()
846   {
847       pop();
848   }
849
850   public int peekStack()
851   {
852       peek();
853   }
854
855   public boolean isEmptyStack()
856   {
857       isEmpty();
858   }
859
860   public boolean isFullStack()
861   {
862       isFull();
863   }
864
865   public void clearStack()
866   {
867       clear();
868   }
869
870   public void resetStack()
871   {
872       reset();
873   }
874
875   public void destroyStack()
876   {
877       destroy();
878   }
879
880   public void printStack()
881   {
882       print();
883   }
884
885   public void popStack()
886   {
887       pop();
888   }
889
890   public int peekStack()
891   {
892       peek();
893   }
894
895   public boolean isEmptyStack()
896   {
897       isEmpty();
898   }
899
900   public boolean isFullStack()
901   {
902       isFull();
903   }
904
905   public void clearStack()
906   {
907       clear();
908   }
909
910   public void resetStack()
911   {
912       reset();
913   }
914
915   public void destroyStack()
916   {
917       destroy();
918   }
919
920   public void printStack()
921   {
922       print();
923   }
924
925   public void popStack()
926   {
927       pop();
928   }
929
930   public int peekStack()
931   {
932       peek();
933   }
934
935   public boolean isEmptyStack()
936   {
937       isEmpty();
938   }
939
940   public boolean isFullStack()
941   {
942       isFull();
943   }
944
945   public void clearStack()
946   {
947       clear();
948   }
949
950   public void resetStack()
951   {
952       reset();
953   }
954
955   public void destroyStack()
956   {
957       destroy();
958   }
959
960   public void printStack()
961   {
962       print();
963   }
964
965   public void popStack()
966   {
967       pop();
968   }
969
970   public int peekStack()
971   {
972       peek();
973   }
974
975   public boolean isEmptyStack()
976   {
977       isEmpty();
978   }
979
980   public boolean isFullStack()
981   {
982       isFull();
983   }
984
985   public void clearStack()
986   {
987       clear();
988   }
989
990   public void resetStack()
991   {
992       reset();
993   }
994
995   public void destroyStack()
996   {
997       destroy();
998   }
999
1000  public void printStack()
1001  {
1002      print();
1003  }
1004
1005  public void popStack()
1006  {
1007      pop();
1008  }
1009
1010  public int peekStack()
1011  {
1012      peek();
1013  }
1014
1015  public boolean isEmptyStack()
1016  {
1017      isEmpty();
1018  }
1019
1020  public boolean isFullStack()
1021  {
1022      isFull();
1023  }
1024
1025  public void clearStack()
1026  {
1027      clear();
1028  }
1029
1030  public void resetStack()
1031  {
1032      reset();
1033  }
1034
1035  public void destroyStack()
1036  {
1037      destroy();
1038  }
1039
1040  public void printStack()
1041  {
1042      print();
1043  }
1044
1045  public void popStack()
1046  {
1047      pop();
1048  }
1049
1050  public int peekStack()
1051  {
1052      peek();
1053  }
1054
1055  public boolean isEmptyStack()
1056  {
1057      isEmpty();
1058  }
1059
1060  public boolean isFullStack()
1061  {
1062      isFull();
1063  }
1064
1065  public void clearStack()
1066  {
1067      clear();
1068  }
1069
1070  public void resetStack()
1071  {
1072      reset();
1073  }
1074
1075  public void destroyStack()
1076  {
1077      destroy();
1078  }
1079
1080  public void printStack()
1081  {
1082      print();
1083  }
1084
1085  public void popStack()
1086  {
1087      pop();
1088  }
1089
1090  public int peekStack()
1091  {
1092      peek();
1093  }
1094
1095  public boolean isEmptyStack()
1096  {
1097      isEmpty();
1098  }
1099
1100  public boolean isFullStack()
1101  {
1102      isFull();
1103  }
1104
1105  public void clearStack()
1106  {
1107      clear();
1108  }
1109
1110  public void resetStack()
1111  {
1112      reset();
1113  }
1114
1115  public void destroyStack()
1116  {
1117      destroy();
1118  }
1119
1120  public void printStack()
1121  {
1122      print();
1123  }
1124
1125  public void popStack()
1126  {
1127      pop();
1128  }
1129
1130  public int peekStack()
1131  {
1132      peek();
1133  }
1134
1135  public boolean isEmptyStack()
1136  {
1137      isEmpty();
1138  }
1139
1140  public boolean isFullStack()
1141  {
1142      isFull();
1143  }
1144
1145  public void clearStack()
1146  {
1147      clear();
1148  }
1149
1150  public void resetStack()
1151  {
1152      reset();
1153  }
1154
1155  public void destroyStack()
1156  {
1157      destroy();
1158  }
1159
1160  public void printStack()
1161  {
1162      print();
1163  }
1164
1165  public void popStack()
1166  {
1167      pop();
1168  }
1169
1170  public int peekStack()
1171  {
1172      peek();
1173  }
1174
1175  public boolean isEmptyStack()
1176  {
1177      isEmpty();
1178  }
1179
1180  public boolean isFullStack()
1181  {
1182      isFull();
1183  }
1184
1185  public void clearStack()
1186  {
1187      clear();
1188  }
1189
1190  public void resetStack()
1191  {
1192      reset();
1193  }
1194
1195  public void destroyStack()
1196  {
1197      destroy();
1198  }
1199
1200  public void printStack()
1201  {
1202      print();
1203  }
1204
1205  public void popStack()
1206  {
1207      pop();
1208  }
1209
1210  public int peekStack()
1211  {
1212      peek();
1213  }
1214
1215  public boolean isEmptyStack()
1216  {
1217      isEmpty();
1218  }
1219
1220  public boolean isFullStack()
1221  {
1222      isFull();
1223  }
1224
1225  public void clearStack()
1226  {
1227      clear();
1228  }
1229
1230  public void resetStack()
1231  {
1232      reset();
1233  }
1234
1235  public void destroyStack()
1236  {
1237      destroy();
1238  }
1239
1240  public void printStack()
1241  {
1242      print();
1243  }
1244
1245  public void popStack()
1246  {
1247      pop();
1248  }
1249
1250  public int peekStack()
1251  {
1252      peek();
1253  }
1254
1255  public boolean isEmptyStack()
1256  {
1257      isEmpty();
1258  }
1259
1260  public boolean isFullStack()
1261  {
1262      isFull();
1263  }
1264
1265  public void clearStack()
1266  {
1267      clear();
1268  }
1269
1270  public void resetStack()
1271  {
1272      reset();
1273  }
1274
1275  public void destroyStack()
1276  {
1277      destroy();
1278  }
1279
1280  public void printStack()
1281  {
1282      print();
1283  }
1284
1285  public void popStack()
1286  {
1287      pop();
1288  }
1289
1290  public int peekStack()
1291  {
1292      peek();
1293  }
1294
1295  public boolean isEmptyStack()
1296  {
1297      isEmpty();
1298  }
1299
1300  public boolean isFullStack()
1301  {
1302      isFull();
1303  }
1304
1305  public void clearStack()
1306  {
1307      clear();
1308  }
1309
1310  public void resetStack()
1311  {
1312      reset();
1313  }
1314
1315  public void destroyStack()
1316  {
1317      destroy();
1318  }
1319
1320  public void printStack()
1321  {
1322      print();
1323  }
1324
1325  public void popStack()
1326  {
1327      pop();
1328  }
1329
1330  public int peekStack()
1331  {
1332      peek();
1333  }
1334
1335  public boolean isEmptyStack()
1336  {
1337      isEmpty();
1338  }
1339
1340  public boolean isFullStack()
1341  {
1342      isFull();
1343  }
1344
1345  public void clearStack()
1346  {
1347      clear();
1348  }
1349
1350  public void resetStack()
1351  {
1352      reset();
1353  }
1354
1355  public void destroyStack()
1356  {
1357      destroy();
1358  }
1359
1360  public void printStack()
1361  {
1362      print();
1363  }
1364
1365  public void popStack()
1366  {
1367      pop();
1368  }
1369
1370  public int peekStack()
1371  {
1372      peek();
1373  }
1374
1375  public boolean isEmptyStack()
1376  {
1377      isEmpty();
1378  }
1379
1380  public boolean isFullStack()
1381  {
1382      isFull();
1383  }
1384
1385  public void clearStack()
1386  {
1387      clear();
1388  }
1389
1390  public void resetStack()
1391  {
1392      reset();
1393  }
1394
1395  public void destroyStack()
1396  {
1397      destroy();
1398  }
1399
1400  public void printStack()
1401  {
1402      print();
1403  }
1404
1405  public void popStack()
1406  {
1407      pop();
1408  }
1409
1410  public int peekStack()
1411  {
1412      peek();
1413  }
1414
1415  public boolean isEmptyStack()
1416  {
1417      isEmpty();
1418  }
1419
1420  public boolean isFullStack()
1421  {
1422      isFull();
1423  }
1424
1425  public void clearStack()
1426  {
1427      clear();
1428  }
1429
1430  public void resetStack()
1431  {
1432      reset();
1433  }
1434
1435  public void destroyStack()
1436  {
1437      destroy();
1438  }
1439
1440  public void printStack()
1441  {
1442      print();
1443  }
1444
1445  public void popStack()
1446  {
1447      pop();
1448  }
1449
1450  public int peekStack()
1451  {
1452      peek();
1453  }
1454
1455  public boolean isEmptyStack()
1456  {
1457      isEmpty();
1458  }
1459
1460  public boolean isFullStack()
1461  {
1462      isFull();
1463  }
1464
1465  public void clearStack()
1466  {
1467      clear();
1468  }
1469
1470  public void resetStack()
1471  {
1472      reset();
1473  }
1474
1475  public void destroyStack()
1476  {
1477      destroy();
1478  }
1479
1480  public void printStack()
1481  {
1482      print();
1483  }
1484
1485  public void popStack()
1486  {
1487      pop();
1488  }
1489
1490  public int peekStack()
1491  {
1492      peek();
1493  }
1494
1495  public boolean isEmptyStack()
1496  {
1497      isEmpty();
1498  }
1499
1500  public boolean isFullStack()
1501  {
1502      isFull();
1503  }
1504
1505  public void clearStack()
1506  {
1507      clear();
1508  }
1509
1510  public void resetStack()
1511  {
1512      reset();
1513  }
1514
1515  public void destroyStack()
1516  {
1517      destroy();
1518  }
1519
1520  public void printStack()
1521  {
1522      print();
1523  }
1524
1525  public void popStack()
1526  {
1527      pop();
1528  }
1529
1530  public int peekStack()
1531  {
1532      peek();
1533  }
1534
1535  public boolean isEmptyStack()
1536  {
1537      isEmpty();
1538  }
1539
1540  public boolean isFullStack()
1541  {
1542      isFull();
1543  }
1544
1545  public void clearStack()
1546  {
1547      clear();
1548  }
1549
1550  public void resetStack()
1551  {
1552      reset();
1553  }
1554
1555  public void destroyStack()
1556  {
1557      destroy();
1558  }
1559
1560  public void printStack()
1561  {
1562      print();
1563  }
1564
1565  public void popStack()
1566  {
1567      pop();
1568  }
1569
1570  public int peekStack()
1571  {
1572      peek();
1573  }
1574
1575  public boolean isEmptyStack()
1576  {
1577      isEmpty();
1578  }
1579
1580  public boolean isFullStack()
1581  {
1582      isFull();
1583  }
1584
1585  public void clearStack()
1586  {
1587      clear();
1588  }
1589
1590  public void resetStack()
1591  {
1592      reset();
1593  }
1594
1595  public void destroyStack()
1596  {
1597      destroy();
1598  }
1599
1600  public void printStack()
1601  {
1602      print();
1603  }
1604
1605  public void popStack()
1606  {
1607      pop();
1608  }
1609
1610  public int peekStack()
1611  {
1612      peek();
1613  }
1614
1615  public boolean isEmptyStack()
1616  {
1617      isEmpty();
1618  }
1619
1620  public boolean isFullStack()
1621  {
1622      isFull();
1623  }
16
```

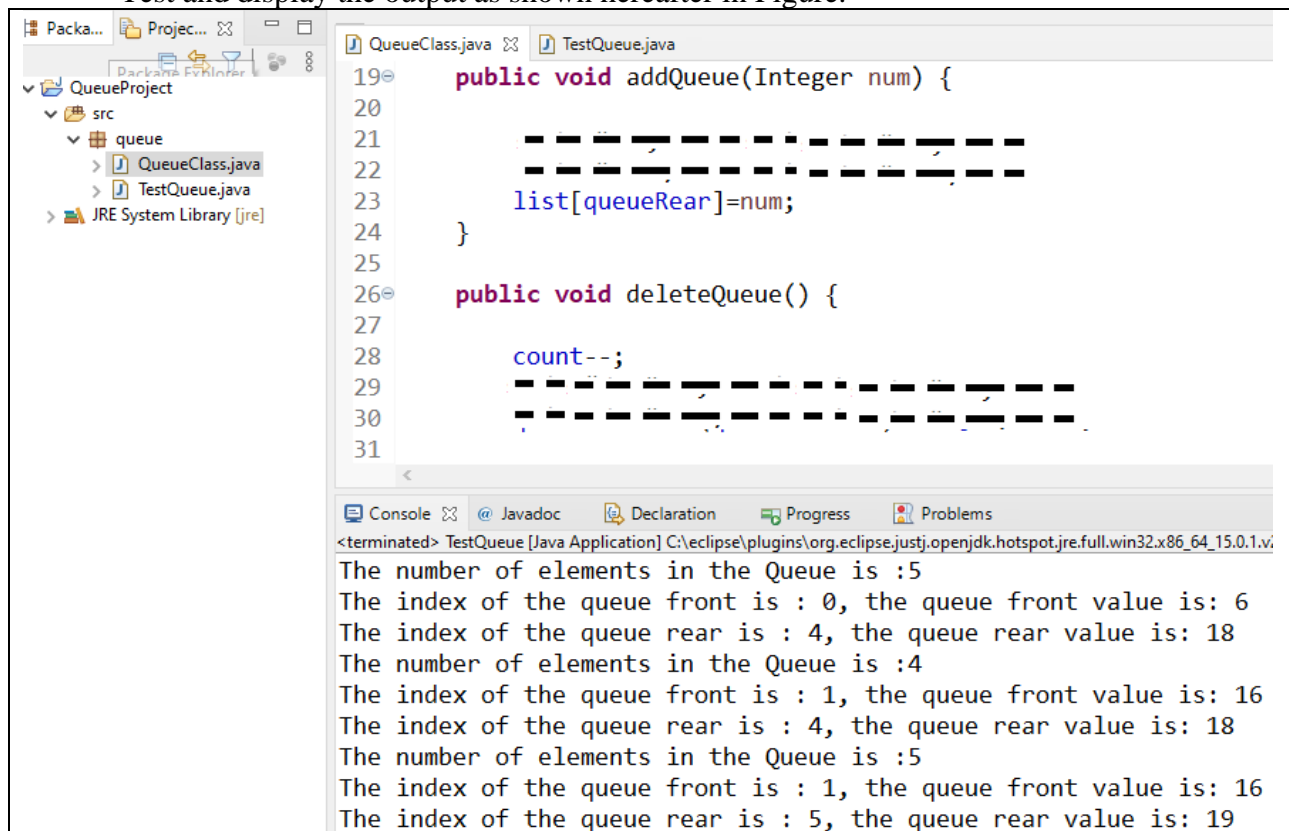
2. Queue Data Structure: Create a Project named QueueProject

Create a Java Project named QueueProject to add element to a given queue and removing an element from a queue.

- Create a class called *QueueClass* which contains the following data members:

```
public int queueFront;    // keeps track of the first element
public int queueRear;     // keeps track of the last element
public int maxQueueSize;  // specifies the maximum size of the queues
public int count;         // number of element in the queue
public Integer[] list;
```

- Add a constructor to *QueueClass* in order to initialize the above data members accordingly.
- Add Java method *addQueue(Integer num)* into *QueueClass* to add an element in the queue.
- Add Java method *deleteQueue()* into *QueueClass* to delete an element in the queue.
- Test and display the output as shown hereafter in Figure.

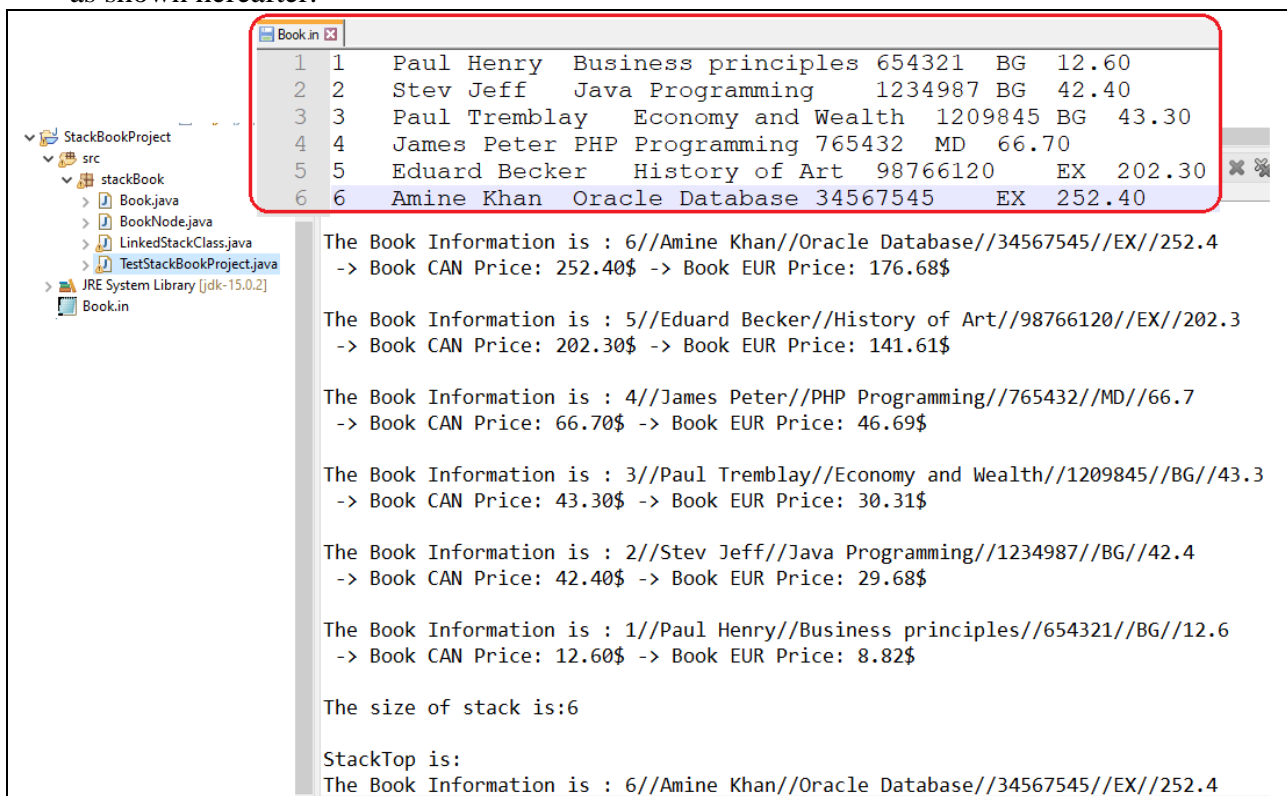


3. Stack List for Book records

- a) Create a Java Project *StackBookProject* to assign the instance data of all *records* read from the input file *Book.in* into a *stack List* to print records in the descending order with respect to the amount of book price. You should use user-defined method *push()* implemented in the user-defined Linked Stack class.
- b) Create a class to define data structure type, called *Book*, which is designed to group data and functions into a single unit that *represents* a template of the fields used in *Book.in* as shown in the following Figure.
- c) Each line within *Book.in* represents a book's record stored in book price ascending order with the following fields: (b_id (int), b_author(string), b_title(string), b_isbn(string),

```
b_type(string), b_price(double)).
```

- d) Add **default constructor** (`b_id=0, b_pid=0, b_author="", b_title="", b_isbn="", b_type="", b_price=0`) and **constructor with parameters** within the Book class in order to initialize the data members (`b_id, b_pid, b_author, b_title, b_isbn, b_type, b_price`) of every object.
- e) Add public **Mutator (setter)** methods (`setBook_id(), setB_Author(), setB_Title(), setB_Isbn(), setB_Type(), setB_Price()`) in Book class to modify the values of private members.
- f) Add public **Accessor (getter)** methods (`getBook_id(), getB_Author(), getB_Title(), getB_Isbn(), getB_Type(), getB_Price()`) in Book class to read the values of private members.
- g) Add a method called public String `toString()` in Book class to print the Book information in the form of "The Book Information is : " + `b_id` + "/" + `b_author` + "/" + `b_title` + "/" + `b_isbn` + "/" + `b_type` + "/" + `b_price`
- h) Add a method called public `calculate_Price_Euro()` in Book class to calculate the price in euro using the following formula: `b_price*0.7`
- i) You need to instantiate in the main method objects of Book class type, and display the output as shown hereafter.



```
Book.in
1 1 Paul Henry Business principles 654321 BG 12.60
2 2 Stev Jeff Java Programming 1234987 BG 42.40
3 3 Paul Tremblay Economy and Wealth 1209845 BG 43.30
4 4 James Peter PHP Programming 765432 MD 66.70
5 5 Eduard Becker History of Art 98766120 EX 202.30
6 6 Amine Khan Oracle Database 34567545 EX 252.40

The Book Information is : 6//Amine Khan//Oracle Database//34567545//EX//252.4
-> Book CAN Price: 252.40$ -> Book EUR Price: 176.68$

The Book Information is : 5//Eduard Becker//History of Art//98766120//EX//202.3
-> Book CAN Price: 202.30$ -> Book EUR Price: 141.61$

The Book Information is : 4//James Peter//PHP Programming//765432//MD//66.7
-> Book CAN Price: 66.70$ -> Book EUR Price: 46.69$

The Book Information is : 3//Paul Tremblay//Economy and Wealth//1209845//BG//43.3
-> Book CAN Price: 43.30$ -> Book EUR Price: 30.31$

The Book Information is : 2//Stev Jeff//Java Programming//1234987//BG//42.4
-> Book CAN Price: 42.40$ -> Book EUR Price: 29.68$

The Book Information is : 1//Paul Henry//Business principles//654321//BG//12.6
-> Book CAN Price: 12.60$ -> Book EUR Price: 8.82$

The size of stack is:6

StackTop is:
The Book Information is : 6//Amine Khan//Oracle Database//34567545//EX//252.4
```

4. Queue with two lines

Create a Java Project named *MultipleQueueProject* in order to implement a new data structure called *MultipleQueueClass* for maintaining two queues *A* and *B* at the same time (in case you have two counter desks). You need to take into account the following requirements:

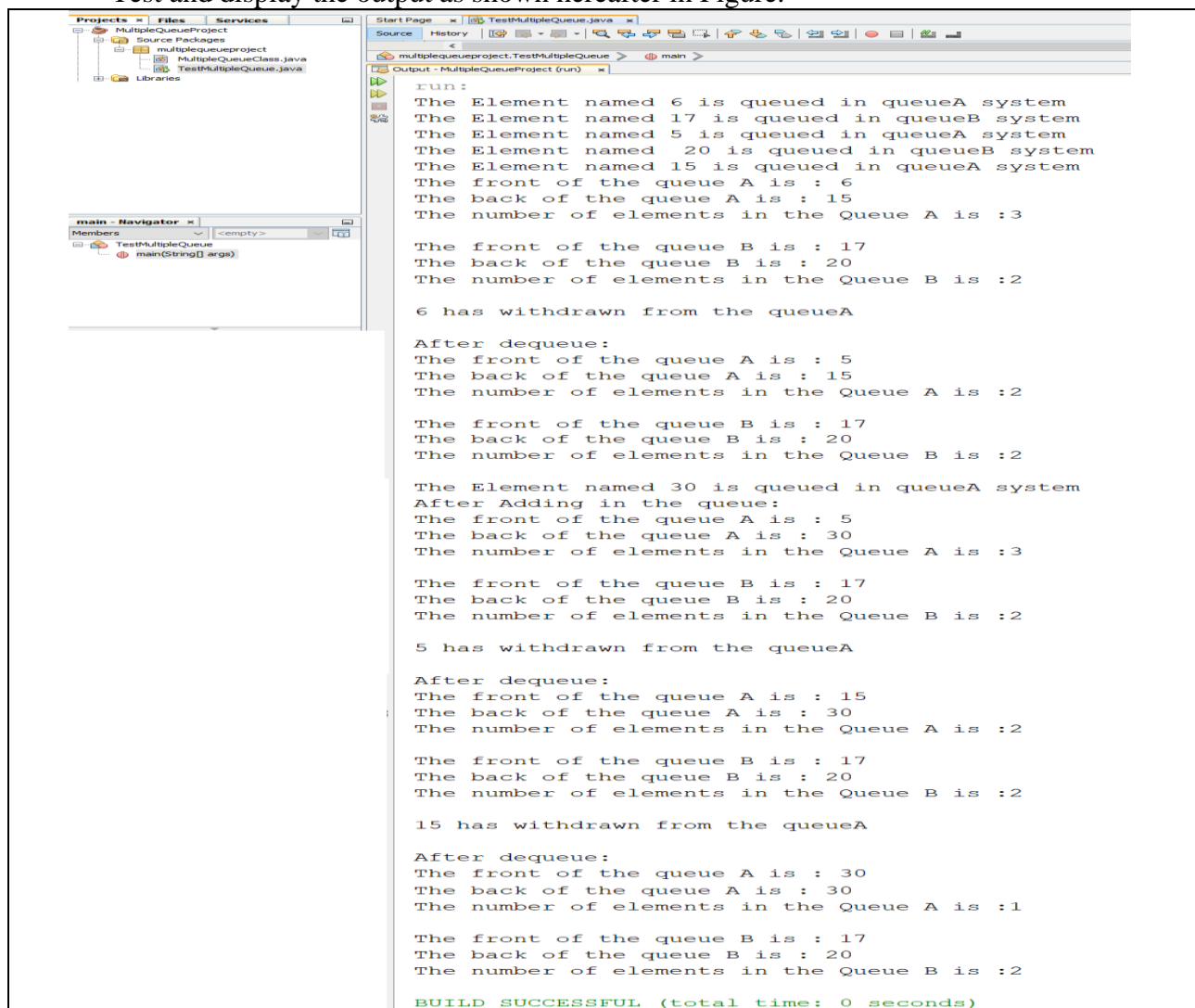
- The first element is inserted in the first queue *A*.
- A scenario when an element is already inserted in the first queue *A*, the second element *has* to be inserted in the second queue *B*.

- A new element is inserted in queue A if number of elements is the same in both queues.
- You need to delete an element from a given queue which has higher size of elements. You delete an element from the queue A if the size is the same in the two queues.
- You need to maintain the front and the rear of each queue.
- Maintain the elements of the two queues in two dimensional array *list[][]*

Implement *MultipleQueueClass* using arrays to simulate two lines of queues in Counter waiting line. *MultipleQueueClass* should contains the following data members:

```
public int queueFrontA; // keeps track of the first element in the first queue
public int queueRearA; // keeps track of the last element in the first queue
public int queueFrontB; // keeps track of the first element in the second queue
public int queueRearB; // keeps track of the last element in the second queue
public int maxQueueSize; // specifies the maximum size of the queues
public int countA ; // number of element in the first queues
public int countB ; // number of element in the second queues
public Integer[][] list;
```

- Add a constructor to *QueueClass* in order to initialize the above data members accordingly.
- Add Java method *addQueue(Integer num)* into *MultipleQueueClass* to add an element in one of the two queues accordingly.
- Add Java method *deleteQueue()* into *MultipleQueueClass* to delete an element in one of the two queues accordingly.
- Test and display the output as shown hereafter in Figure.



5. Building Binary Tree class

Create project *BinaryTreeProject* to implement Binary search tree. Create a class called *BinaryTreeClass* which includes an attribute *rootTree* of *BinaryTreeNode* class type as done during Zoom synchronous class.

- Add a constructor to *BinaryTreeClass* in order to initialise *rootTree* to null.
- Add Java method *put()* into *BinaryTreeClass* to insert an element *num* in the Binary Search Tree.
- Add Java method *search()* into *BinaryTreeClass* to search an element *num* in the Binary Search Tree.
- Add Java method *InorderTraversal()* into *BinaryTreeClass* to print all elements of Binary Search Tree.
- Add Java method *PreorderTraversal()* into *BinaryTreeClass* to print all elements of Binary Search Tree.
- Add Java method *PostorderTraversal()* into *BinaryTreeClass* to print all elements of Binary Search Tree.
- Test and display the output as shown hereafter in Figure.

```
2 public class BinaryTreeClass {
3     BinaryTreeNode rootTree;
4
5     public BinaryTreeClass() {
6         rootTree = null;
7     }
8
9     public void put(int num)
10    {
11        //add num to tree
12        BinaryTreeNode myNodeTree = new BinaryTreeNode(num);
13        myNodeTree.info = num;
14        myNodeTree.left = null;
15        myNodeTree.right = null;
16
17        BinaryTreeNode currentNode;
18
19        if (rootTree == null)
20        {
21            myNodeTree.info = num;
22            myNodeTree.left = null;
23            myNodeTree.right = null;
24            rootTree = myNodeTree;
25        }
26        else
27        {
28            currentNode = rootTree;
29            while (currentNode != null)
30            {
31                if (num < currentNode.info)
32                {
33                    if (currentNode.left == null)
34                    {
35                        myNodeTree.info = num;
36                        myNodeTree.left = null;
37                        myNodeTree.right = null;
38                        currentNode.left = myNodeTree;
39                        break;
40                    }
41                    else
42                    {
43                        currentNode = currentNode.left;
44                    }
45                }
46                else
47                {
48                    if (currentNode.right == null)
49                    {
50                        myNodeTree.info = num;
51                        myNodeTree.left = null;
52                        myNodeTree.right = null;
53                        currentNode.right = myNodeTree;
54                        break;
55                    }
56                    else
57                    {
58                        currentNode = currentNode.right;
59                    }
60                }
61            }
62        }
63    }
64
65    public void search(int num)
66    {
67        BinaryTreeNode currentNode = rootTree;
68        while (currentNode != null)
69        {
70            if (num == currentNode.info)
71            {
72                System.out.println("Value " + num + " exists in the tree");
73                return;
74            }
75            else if (num < currentNode.info)
76            {
77                currentNode = currentNode.left;
78            }
79            else
80            {
81                currentNode = currentNode.right;
82            }
83        }
84        System.out.println("Value " + num + " does not exist in the tree");
85    }
86
87    public void InorderTraversal()
88    {
89        InorderTraversal(rootTree);
90    }
91
92    public void PreorderTraversal()
93    {
94        PreorderTraversal(rootTree);
95    }
96
97    public void PostorderTraversal()
98    {
99        PostorderTraversal(rootTree);
100    }
101}
```

```
<terminated> TestBinaryTree [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full\jre\bin\java.exe
Printing Inorder Tree Values:
-----40-----
-----50-----
-----60-----
-----70-----
-----80-----

Printing Preorder Tree Values:
-----60-----
-----50-----
-----40-----
-----70-----
-----80-----

Printing Postorder Tree Values:
-----40-----
-----50-----
-----80-----
-----70-----
-----60-----

Please enter number for search within Binary Tree
40
value 40 exists in the tree
```

6. Binary Tree for Employee records

Create a Java Project *EmployeeTreeProject* to assign the instance data of all *records* read from the input file *Employee.in* into a *Binary Search Tree* such that *emp_salary* attribute in *each node* of *Binary Search Tree* is **Larger** than the *emp_salary* in its left child and **Smaller** than the *emp_salary* in its right child as shown in Figure.

- Create class *Employee* that defines data attributes of every employee record (*emp_id*, *emp_name*, *emp_salary*).
- Create class *BinaryEmployeeTreeNode* that defines every node in tree where data info is of *Employee* Data type.
- Create class *BinaryEmployeeTreeClass* that defines *put* (*BinaryEmployeeTreeNode obj*) and *InorderTraversal()* operations related to binary tree.
- Test and display the output as shown hereafter in Figure.

```
30
31
32 System.out.println("Displaying the components of the Tree list stored
33 + "\nfrom input file Employee.in as Inorder Traversal");
34 employeefileTree.InorderTraversal();
```

```
Employee Id: 810 Employee Name: Fanel Yun Employee Salary: 30000.00
Employee Id: 910 Employee Name: Simson Sarah Employee Salary: 50000.00
Employee Id: 310 Employee Name: Emmanuel Jessica Employee Salary: 55000.00
Employee Id: 110 Employee Name: Rantson Ali Employee Salary: 60000.00
Employee Id: 370 Employee Name: Siroko Ayo Employee Salary: 95000.00
Employee Id: 210 Employee Name: Rollen Bob Employee Salary: 98000.00
Employee Id: 340 Employee Name: Mendy Anais Employee Salary: 99000.00
```