

Task 1:

My solution for this task can be found in Activity.java.

Task 2:

```
1 usage
3 public class Meeting extends Activity {
4     2 usages
5     String agenda;
6     1 usage
7     > public Meeting(String title) { super(title); }
8
9
10    /**
11     * returns the agenda of the meeting
12     * @return the agenda of the meeting
13     */
14    no usages
15    > public String getAgenda() { return this.agenda; }
16
17
18    /**
19     * returns a string representation of the meeting.
20     * @return a string representation of the meeting
21     */
22    Ⓢ public String toString() {
23        return "Title: " + this.title + " - Duration: " + this.duration
24            + " - Agenda: " + this.agenda + "\n";
25    }
26 } ⚡
```

Memory is allocated for 3 instances when a meeting is created. The three are: agenda which is directly in the meeting class, Title and duration which are inherited from the activity class.

Task 3:

My solution for this task can be found in SuggestedActivity.java.

#### Task 4:

```
1 usage
3 public class SimpleDecider implements TimeSlotDecider {
    1 usage
4     @Override
5     @
6     public void decideTimeSlot(SuggestedActivity a) {
7         TimeSlot bestTimeSlot = null;
8         int bestVotes = 0;
9         for (TimeSlot t : a.getPossibleTimeSlots()) {
10            int votes = a.getVotes(t).size();
11            if (votes > bestVotes) {
12                bestTimeSlot = t;
13                bestVotes = votes;
14            }
15        }
16        a.setTimeSlot(bestTimeSlot);
17    }
}
```

I have implemented this decideTimeSlot method such that it finds the timeSlot that has the most votes in cases of multiple timeSlots with equal amounts of votes it simply selects the first timeSlot it finds with that amount of votes.

#### Task 5:

My solution to this task can be found in Participant.java

#### Task 6:

In the TimeSlot class I have overwritten the toString method inherited from object

```
42     @Override
43     @
44     public String toString() {
45         return start.toString() + " - " + end.toString();
46     }
47 }
```

I have done this to be able to easily print out timeSlots in my application.

Furthermore I also implemented a getDurationMethod this was the easiest way to let the activity know the duration when the polling is closed and a time slot has been selected for the activity.

```
35     35
36     36
37     37
38     38
39     39
40     40
41     41
42     42
43     43
44     44
45     45
46     46
47     47
48     48
49     49
50     50
51     51
52     52
53     53
54     54
55     55
56     56
57     57
58     58
59     59
60     60
61     61
62     62
63     63
64     64
65     65
66     66
67     67
68     68
69     69
70     70
71     71
72     72
73     73
74     74
75     75
76     76
77     77
78     78
79     79
80     80
81     81
82     82
83     83
84     84
85     85
86     86
87     87
88     88
89     89
90     90
91     91
92     92
93     93
94     94
95     95
96     96
97     97
98     98
99     99
100    100
101    101
102    102
103    103
104    104
105    105
106    106
107    107
108    108
109    109
110    110
111    111
112    112
113    113
114    114
115    115
116    116
117    117
118    118
119    119
120    120
121    121
122    122
123    123
124    124
125    125
126    126
127    127
128    128
129    129
130    130
131    131
132    132
133    133
134    134
135    135
136    136
137    137
138    138
139    139
140    140
141    141
142    142
143    143
144    144
145    145
146    146
147    147
148    148
149    149
150    150
151    151
152    152
153    153
154    154
155    155
156    156
157    157
158    158
159    159
160    160
161    161
162    162
163    163
164    164
165    165
166    166
167    167
168    168
169    169
170    170
171    171
172    172
173    173
174    174
175    175
176    176
177    177
178    178
179    179
180    180
181    181
182    182
183    183
184    184
185    185
186    186
187    187
188    188
189    189
190    190
191    191
192    192
193    193
194    194
195    195
196    196
197    197
198    198
199    199
200    200
201    201
202    202
203    203
204    204
205    205
206    206
207    207
208    208
209    209
210    210
211    211
212    212
213    213
214    214
215    215
216    216
217    217
218    218
219    219
220    220
221    221
222    222
223    223
224    224
225    225
226    226
227    227
228    228
229    229
230    230
231    231
232    232
233    233
234    234
235    235
236    236
237    237
238    238
239    239
240    240
241    241
242    242
243    243
244    244
245    245
246    246
247    247
248    248
249    249
250    250
251    251
252    252
253    253
254    254
255    255
256    256
257    257
258    258
259    259
260    260
261    261
262    262
263    263
264    264
265    265
266    266
267    267
268    268
269    269
270    270
271    271
272    272
273    273
274    274
275    275
276    276
277    277
278    278
279    279
280    280
281    281
282    282
283    283
284    284
285    285
286    286
287    287
288    288
289    289
290    290
291    291
292    292
293    293
294    294
295    295
296    296
297    297
298    298
299    299
300    300
301    301
302    302
303    303
304    304
305    305
306    306
307    307
308    308
309    309
310    310
311    311
312    312
313    313
314    314
315    315
316    316
317    317
318    318
319    319
320    320
321    321
322    322
323    323
324    324
325    325
326    326
327    327
328    328
329    329
330    330
331    331
332    332
333    333
334    334
335    335
336    336
337    337
338    338
339    339
340    340
341    341
342    342
343    343
344    344
345    345
346    346
347    347
348    348
349    349
350    350
351    351
352    352
353    353
354    354
355    355
356    356
357    357
358    358
359    359
360    360
361    361
362    362
363    363
364    364
365    365
366    366
367    367
368    368
369    369
370    370
371    371
372    372
373    373
374    374
375    375
376    376
377    377
378    378
379    379
380    380
381    381
382    382
383    383
384    384
385    385
386    386
387    387
388    388
389    389
390    390
391    391
392    392
393    393
394    394
395    395
396    396
397    397
398    398
399    399
400    400
401    401
402    402
403    403
404    404
405    405
406    406
407    407
408    408
409    409
410    410
411    411
412    412
413    413
414    414
415    415
416    416
417    417
418    418
419    419
420    420
421    421
422    422
423    423
424    424
425    425
426    426
427    427
428    428
429    429
430    430
431    431
432    432
433    433
434    434
435    435
436    436
437    437
438    438
439    439
440    440
441    441
442    442
443    443
444    444
445    445
446    446
447    447
448    448
449    449
450    450
451    451
452    452
453    453
454    454
455    455
456    456
457    457
458    458
459    459
460    460
461    461
462    462
463    463
464    464
465    465
466    466
467    467
468    468
469    469
470    470
471    471
472    472
473    473
474    474
475    475
476    476
477    477
478    478
479    479
480    480
481    481
482    482
483    483
484    484
485    485
486    486
487    487
488    488
489    489
490    490
491    491
492    492
493    493
494    494
495    495
496    496
497    497
498    498
499    499
500    500
501    501
502    502
503    503
504    504
505    505
506    506
507    507
508    508
509    509
510    510
511    511
512    512
513    513
514    514
515    515
516    516
517    517
518    518
519    519
520    520
521    521
522    522
523    523
524    524
525    525
526    526
527    527
528    528
529    529
530    530
531    531
532    532
533    533
534    534
535    535
536    536
537    537
538    538
539    539
540    540
541    541
542    542
543    543
544    544
545    545
546    546
547    547
548    548
549    549
550    550
551    551
552    552
553    553
554    554
555    555
556    556
557    557
558    558
559    559
560    560
561    561
562    562
563    563
564    564
565    565
566    566
567    567
568    568
569    569
570    570
571    571
572    572
573    573
574    574
575    575
576    576
577    577
578    578
579    579
580    580
581    581
582    582
583    583
584    584
585    585
586    586
587    587
588    588
589    589
590    590
591    591
592    592
593    593
594    594
595    595
596    596
597    597
598    598
599    599
600    600
601    601
602    602
603    603
604    604
605    605
606    606
607    607
608    608
609    609
610    610
611    611
612    612
613    613
614    614
615    615
616    616
617    617
618    618
619    619
620    620
621    621
622    622
623    623
624    624
625    625
626    626
627    627
628    628
629    629
630    630
631    631
632    632
633    633
634    634
635    635
636    636
637    637
638    638
639    639
640    640
641    641
642    642
643    643
644    644
645    645
646    646
647    647
648    648
649    649
650    650
651    651
652    652
653    653
654    654
655    655
656    656
657    657
658    658
659    659
660    660
661    661
662    662
663    663
664    664
665    665
666    666
667    667
668    668
669    669
670    670
671    671
672    672
673    673
674    674
675    675
676    676
677    677
678    678
679    679
680    680
681    681
682    682
683    683
684    684
685    685
686    686
687    687
688    688
689    689
690    690
691    691
692    692
693    693
694    694
695    695
696    696
697    697
698    698
699    699
700    700
701    701
702    702
703    703
704    704
705    705
706    706
707    707
708    708
709    709
710    710
711    711
712    712
713    713
714    714
715    715
716    716
717    717
718    718
719    719
720    720
721    721
722    722
723    723
724    724
725    725
726    726
727    727
728    728
729    729
730    730
731    731
732    732
733    733
734    734
735    735
736    736
737    737
738    738
739    739
740    740
741    741
742    742
743    743
744    744
745    745
746    746
747    747
748    748
749    749
750    750
751    751
752    752
753    753
754    754
755    755
756    756
757    757
758    758
759    759
760    760
761    761
762    762
763    763
764    764
765    765
766    766
767    767
768    768
769    769
770    770
771    771
772    772
773    773
774    774
775    775
776    776
777    777
778    778
779    779
780    780
781    781
782    782
783    783
784    784
785    785
786    786
787    787
788    788
789    789
790    790
791    791
792    792
793    793
794    794
795    795
796    796
797    797
798    798
799    799
800    800
801    801
802    802
803    803
804    804
805    805
806    806
807    807
808    808
809    809
810    810
811    811
812    812
813    813
814    814
815    815
816    816
817    817
818    818
819    819
820    820
821    821
822    822
823    823
824    824
825    825
826    826
827    827
828    828
829    829
830    830
831    831
832    832
833    833
834    834
835    835
836    836
837    837
838    838
839    839
840    840
841    841
842    842
843    843
844    844
845    845
846    846
847    847
848    848
849    849
850    850
851    851
852    852
853    853
854    854
855    855
856    856
857    857
858    858
859    859
860    860
861    861
862    862
863    863
864    864
865    865
866    866
867    867
868    868
869    869
870    870
871    871
872    872
873    873
874    874
875    875
876    876
877    877
878    878
879    879
880    880
881    881
882    882
883    883
884    884
885    885
886    886
887    887
888    888
889    889
890    890
891    891
892    892
893    893
894    894
895    895
896    896
897    897
898    898
899    899
900    900
901    901
902    902
903    903
904    904
905    905
906    906
907    907
908    908
909    909
910    910
911    911
912    912
913    913
914    914
915    915
916    916
917    917
918    918
919    919
920    920
921    921
922    922
923    923
924    924
925    925
926    926
927    927
928    928
929    929
930    930
931    931
932    932
933    933
934    934
935    935
936    936
937    937
938    938
939    939
940    940
941    941
942    942
943    943
944    944
945    945
946    946
947    947
948    948
949    949
950    950
951    951
952    952
953    953
954    954
955    955
956    956
957    957
958    958
959    959
960    960
961    961
962    962
963    963
964    964
965    965
966    966
967    967
968    968
969    969
970    970
971    971
972    972
973    973
974    974
975    975
976    976
977    977
978    978
979    979
980    980
981    981
982    982
983    983
984    984
985    985
986    986
987    987
988    988
989    989
990    990
991    991
992    992
993    993
994    994
995    995
996    996
997    997
998    998
999    999
1000   1000
1001   1001
1002   1002
1003   1003
1004   1004
1005   1005
1006   1006
1007   1007
1008   1008
1009   1009
1010   1010
1011   1011
1012   1012
1013   1013
1014   1014
1015   1015
1016   1016
1017   1017
1018   1018
1019   1019
1020   1020
1021   1021
1022   1022
1023   1023
1024   1024
1025   1025
1026   1026
1027   1027
1028   1028
1029   1029
1030   1030
1031   1031
1032   1032
1033   1033
1034   1034
1035   1035
1036   1036
1037   1037
1038   1038
1039   1039
1040   1040
1041   1041
1042   1042
1043   1043
1044   1044
1045   1045
1046   1046
1047   1047
1048   1048
1049   1049
1050   1050
1051   1051
1052   1052
1053   1053
1054   1054
1055   1055
1056   1056
1057   1057
1058   1058
1059   1059
1060   1060
1061   1061
1062   1062
1063   1063
1064   1064
1065   1065
1066   1066
1067   1067
1068   1068
1069   1069
1070   1070
1071   1071
1072   1072
1073   1073
1074   1074
1075   1075
1076   1076
1077   1077
1078   1078
1079   1079
1080   1080
1081   1081
1082   1082
1083   1083
1084   1084
1085   1085
1086   1086
1087   1087
1088   1088
1089   1089
1090   1090
1091   1091
1092   1092
1093   1093
1094   1094
1095   1095
1096   1096
1097   1097
1098   1098
1099   1099
1100   1100
1101   1101
1102   1102
1103   1103
1104   1104
1105   1105
1106   1106
1107   1107
1108   1108
1109   1109
1110   1110
1111   1111
1112   1112
1113   1113
1114   1114
1115   1115
1116   1116
1117   1117
1118   1118
1119   1119
1120   1120
1121   1121
1122   1122
1123   1123
1124   1124
1125   1125
1126   1126
1127   1127
1128   1128
1129   1129
1130   1130
1131   1131
1132   1132
1133   1133
1134   1134
1135   1135
1136   1136
1137   1137
1138   1138
1139   1139
1140   1140
1141   1141
1142   1142
1143   1143
1144   1144
1145   1145
1146   1146
1147   1147
1148   1148
1149   1149
1150   1150
1151   1151
1152   1152
1153   1153
1154   1154
1155   1155
1156   1156
1157   1157
1158   1158
1159   1159
1160   1160
1161   1161
1162   1162
1163   1163
1164   1164
1165   1165
1166   1166
1167   1167
1168   1168
1169   1169
1170   1170
1171   1171
1172   1172
1173   1173
1174   1174
1175   1175
1176   1176
1177   1177
1178   1178
1179   1179
1180   1180
1181   1181
1182   1182
1183   1183
1184   1184
1185   1185
1186   1186
1187   1187
1188   1188
1189   1189
1190   1190
1191   1191
1192   1192
1193   1193
1194   1194
1195   1195
1196   1196
1197   1197
1198   1198
1199   1199
1200   1200
1201   1201
1202   1202
1203   1203
1204   1204
1205   1205
1206   1206
1207   1207
1208   1208
1209   1209
1210   1210
1211   1211
1212   1212
1213   1213
1214   1214
1215   1215
1216   1216
1217   1217
1218   1218
1219   1219
1220   1220
1221   1221
1222   1222
1223   1223
1224   1224
1225   1225
1226   1226
1227   1227
1228   1228
1229   1229
1230   1230
1231   1231
1232   1232
1233   1233
1234   1234
1235   1235
1236   1236
1237   1237
1238   1238
1239   1239
1240   1240
1241   1241
1242   1242
1243   1243
1244   1244
1245   1245
1246   1246
1247   1247
1248   1248
1249   1249
1250   1250
1251   1251
1252   1252
1253   1253
1254   1254
1255   1255
1256   1256
1257   1257
1258   1258
1259   1259
1260   1260
1261   1261
1262   1262
1263   1263
1264   1264
1265   1265
1266   1266
1267   1267
1268   1268
1269   1269
1270   1270
1271   1271
1272   1272
1273   1273
1274   1274
1275   1275
1276   1276
1277   1277
1278   1278
1279   1279
1280   1280
1281   1281
1282   1282
1283   1283
1284   1284
1
```

### Task 7:

The entire code for the ActivityOrganizer class can be found in ActivityOrganizer.java

### Task 8:

```
11 > public static void main(String[] args) {
12     for (int i = 0; i < 5; i++) {
13         System.out.println("\nFirst you need to create an activity by suggesting it.");
14         suggestActivity();
15         String input = "y";
16         System.out.println("\nNow you need to register what participant will be available for the activity and vote for the times");
17         while (input.equals("y")) {
18             registerAvailability();
19             input = InputSystem.TakeStringInput("Do you want to register another participant? (y/n)");
20         }
21
22         System.out.println("\nFinally you need to schedule the activity");
23         scheduleActivity();
24         System.out.println("\nHere is a list of all the activities so far");
25         displaySuggestedActivities();
26     }
27 }
```

This main method takes the user through five iterations of suggesting an activity and the choosing or creating participants to vote. Then finally it schedules the activity and then prints a list of all the activities created so far.

### Task 9:

```
84     /**
85      * schedules a suggested activity. The method allows the user to
86      * choose any of the activities that have been suggested so far.
87      */
88     1 usage
89     public static void scheduleActivity() {
90         SuggestedActivity suggestedActivity = InputSystem.TakeSuggestedActivityInput(suggestedActivities);
91         suggestedActivity.closePolling(new SimpleDecider());
92     }
```

SuggestedActivity is defined with early binding as it is evaluated in compile time as the class SuggestedActivity. The SimpleDecider on the other hand uses late binding since it is bound to a SimpleDecider in run Time.

## Task 10:

```
60      /**
61       * the interested participant p and their
62       * votes for time slots ts are registered.
63       * @param p the participant
64       * @param ts the time slots
65       */
66      @1 usage
67      public void vote(Participant p, Collection<TimeSlot> ts) {
68          this.participants.add(p);
69          for (TimeSlot t : ts) {
70              this.timeSlotVotes.get(t).add(p);
71          }
72      }
```

In the suggestedActivity class the observer pattern is almost fulfilled firstly in the vote method the Participant is registered in participants, later they are then notified with the notifyAllParticipants method. If the participants here are considered observers, they then register for events by voting, it is as of now not possible to unregister from the event again. This could easily be added such that the participants could unregister from the event and then fulfill the observer pattern.

```
88      /**
89       * notifies all the interested participants when a time slot for
90       * the activity has been chosen.
91       */
92      @1 usage
93      public void notifyAllParticipants() {
94          for (Participant p : this.participants) {
95              p.notify(a: this);
96          }
97      }
```

## Task 11:

```
1 usage
21 @ public boolean overlaps(TimeSlot other) {
22     //If other is within this
23     if(this.start.isBefore(other.start) && this.end.isAfter(other.start)){
24         return true;
25     }
26
27     //If this is within other
28     if(this.start.isAfter(other.start) && this.start.isBefore(other.end)){
29         return true;
30     }
31
32     //If this ends after other starts and ends before other ends
33     if(this.end.isAfter(other.start) && this.end.isBefore(other.end)){
34         return true;
35     }
36
37     //If this ends after other and starts before other starts
38     return this.end.isAfter(other.end) && this.start.isBefore(other.start);
39 }
```

The four cases where the activity overlaps with another activity in the calendar are shown in the figure above. Lets call them t1 and t2 the two timeslots we compare for simplicity. The four cases are if:

- t2 is within t1,
- t1 is within t2.
- t1 start after t2 starts and ends after t2 ends
- t1 ends after t2 ends and starts before t2 starts

All these 4 cases are covered in the image above and thus it should ensure no overlap, there is one case though where this check isn't sufficient, and this is the case where t2 is equal to t1 in both start and end. Then these checks would return false due to the isBefore and isAfter methods returning false for cases that are equal (because it is not before or after as it is the same).

## Task 12

```
7
14 usages
8 public class SuggestedActivity {
    2 usages
9     private boolean pollingOpen = true;
    5 usages
10    private final Activity activity;
    4 usages
11    private final Map<TimeSlot, Collection<Participant>> timeSlotVotes = new HashMap<>();
    2 usages
12    private final Collection<Participant> participants = new ArrayList<>();
    5 usages
13    private TimeSlot timeSlot;
14
```

All variables in the SuggestedActivity are encapsulated, all are private and thus accessed (and changed) with methods within the class. This also adds a level of abstraction to the class since the user of this class has no need to know how the data is saved and only needs to retrieve data and have it stored.

Furthermore abstraction is also used on the Activity parameter as it is an abstract class the actual class of the parameter is evaluated in runTime when e.g I run .toString on activity that is method is overwritten by the runtime classes method. Since only meeting is an inheritor this will in the current version of the application be the class of activity.

```
99
100 /**
101  * returns a string representation of the suggested activity.
102  * @return a string representation of the suggested activity
103  */
103 @
104 public String toString() {
105     if(pollingOpen){
106         return this.activity.toString();
107     }
108     return this.timeSlot.getStartDate().toString() + " - " + this.timeSlot.getEndDate().toString() + ": " +
109         this.activity.toString();
110 }
```