



[← Return to Classroom](#)

UdaSecurity

REVIEW

HISTORY

Unable to review

Key parts of your submission were the same as another student's submission or an online source. Please resubmit after you address the issue noted below by the reviewer.

POTENTIAL SOURCE URL: <https://github.com/Sanjay93/udasecurityparent> | <https://github.com/Eduguimar/Udasecurity>

Good Morning

I've checked your code and I could see that you've violated the:

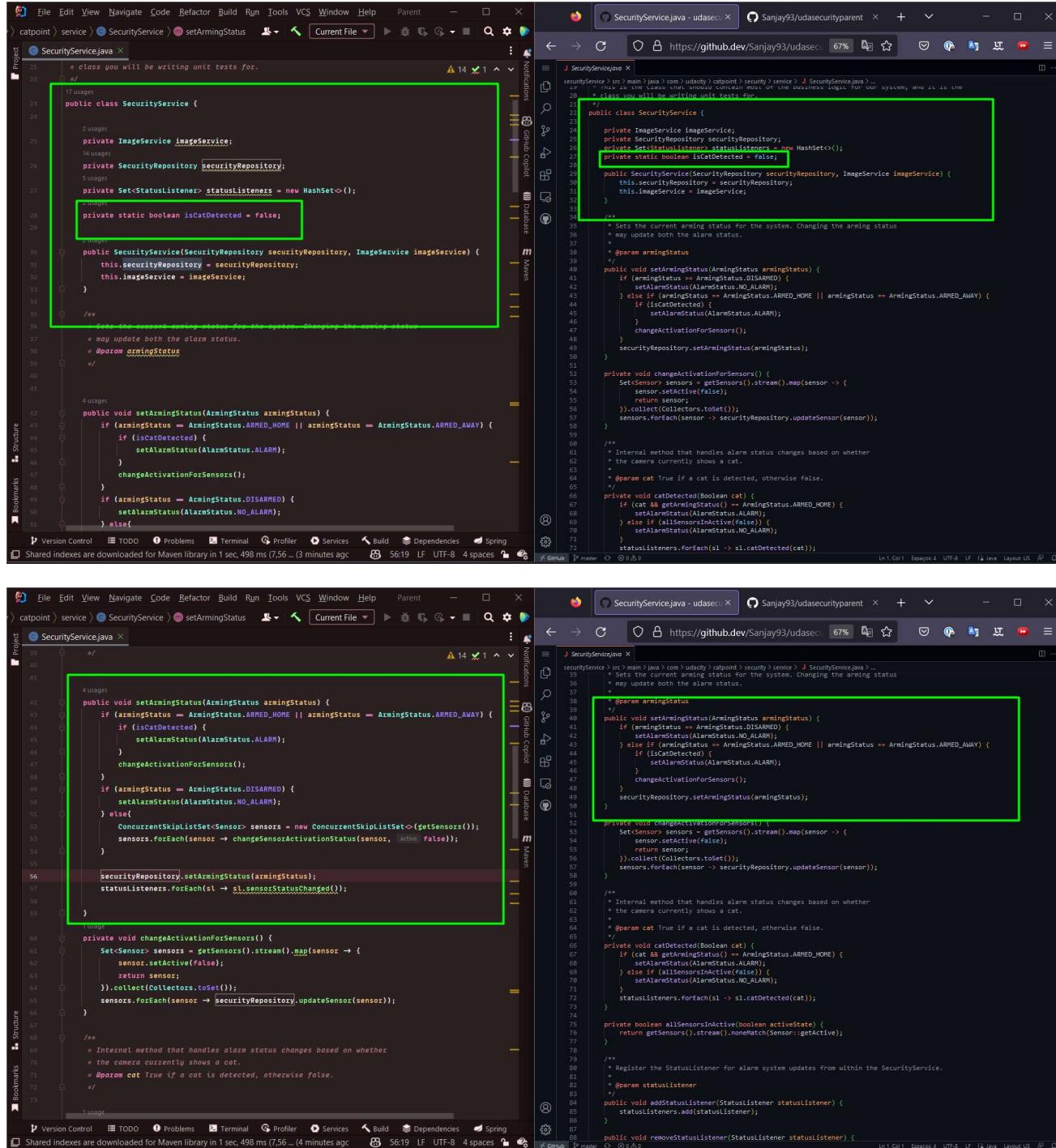
- Udacity's no-plagiarism honor code

More specifically the items:

- Copying someone's code exactly, in whole or part (verbatim)
- Copying someone's code, in whole or part, and making changes (e.g. changing variable or function names, comments, order of function definition)
- Combining code copied verbatim from multiple sources.

This can be seen in the screenshots right below:

SCREENSHOTS -



```

 17 usages
public class SecurityService {

    2 usages
    private ImageService imageService;
    14 usages
    private SecurityRepository securityRepository;
    5 usages
    private Set<StatusListener> statusListeners = new HashSet<>();
    2 usages
    private static boolean isCatDetected = false;

    /**
     * Sets the current arming status for the system. Changing the arming status
     * may update both the alarm status.
     * @param armingStatus
     */
    4 usages
    public void setArmingStatus(ArmingStatus armingStatus) {
        if (armingStatus == ArmingStatus.ARMED_HOME || armingStatus == ArmingStatus.ARMED_AWAY) {
            if (isCatDetected) {
                setAlarmStatus(AlarmStatus.ALARM);
            }
            changeActivationForSensors();
        }
        if (armingStatus == ArmingStatus.DISARMED) {
            setAlarmStatus(AlarmStatus.NO_ALARM);
        } else{
            ConcurrentSkipListSet<Sensor> sensors = new ConcurrentSkipListSet<>(getSensors());
            sensors.forEach(sensor -> changeSensorActivationStatus(sensor, active: false));
        }

        securityRepository.setArmingStatus(armingStatus);
        statusListeners.forEach(sl -> sl.sensorStatusChanged());
    }

    private void changeActivationForSensors() {
        Set<Sensor> sensors = getSensor().stream().map(sensor -> {
            sensor.setActive(false);
            return sensor;
        }).collect(Collectors.toSet());
        sensors.forEach(sensor -> securityRepository.updateSensor(sensor));
    }

    /**
     * Internal method that handles alarm status changes based on whether
     * the camera currently shows a cat.
     * @param cat True if a cat is detected, otherwise false.
     */
    private void catDetected(Boolean cat) {
        if (cat && getArmingStatus() == ArmingStatus.ARMED_HOME) {
            setAlarmStatus(AlarmStatus.ALARM);
        } else if (allSensorsInactive(false)) {
            setAlarmStatus(AlarmStatus.NO_ALARM);
        }
        statusListeners.forEach(sl -> sl.catDetected(cat));
    }

    private boolean allSensorsInactive(boolean activeState) {
        return getSensor().stream().noneMatch(sensor -> sensor.getActive());
    }

    /**
     * Register the StatusListener for alarm system updates from within the SecurityService.
     */
    public void addStatusListener(StatusListener statusListener) {
        statusListeners.add(statusListener);
    }

    public void removeStatusListener(StatusListener statusListener) {
        statusListeners.remove(statusListener);
    }
}

```

The screenshot shows two side-by-side IDE panes. The left pane is a Java code editor for `SecurityService.java` within a project named `catpoint > service > SecurityService`. The right pane is a browser window displaying the same code on the GitHub repository page at `https://github.dev/Sanjay93/udasecurityparent`.

Left Pane (Visual Studio Code):

```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help Parent
Project SecurityService.java
56     securityRepository.setArmingStatus(armingStatus);
57     statusListeners.forEach(sl -> sl.sensorStatusChanged());
58 }
59 }
60
61 /**
62 * Usage
63 * private void changeActivationForSensors() {
64     Set<Sensor> sensors = getSensors().stream().map(sensor -> {
65         sensor.setActive(false);
66         return sensor;
67     }).collect(Collectors.toSet());
68     sensors.forEach(sensor -> securityRepository.updateSensor(sensor));
69 }
70
71 /**
72 * Internal method that handles alarm status changes based on whether
73 * the camera currently shows a cat.
74 * @param cat True if a cat is detected, otherwise false.
75 */
76
77 /**
78 * Usage
79 * private void catDetected(Boolean cat) {
80     isCatDetected = cat;
81     if(cat && getArmingStatus() == ArmingStatus.ARMED_HOME) {
82         setAlarmStatus(AlarmStatus.ALARM);
83     } else if ( !cat && getSensors().stream().allMatch(s -> s.getActive() == false)) {
84         setAlarmStatus(AlarmStatus.NO_ALARM);
85     }
86
87     statusListeners.forEach(sl -> sl.catDetected(cat));
88 }
89
90 /**
91 * Else if ( !cat && sensorsStatus(false) {
92 //getSensors().stream().noneMatch(sensor::getActive)
93 // !cat && getSensors().stream().allMatch(s -> s.getActive() == false)
94
95 /**
96 * Register the StatusListener for alarm system updates from within the SecurityService.
97 * @param statusListener
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
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
567
568
569
569
570
571
572
573
574
575
575
576
577
578
579
579
580
581
582
583
584
585
585
586
587
588
589
589
590
591
592
593
594
595
595
596
597
598
599
599
600
601
602
603
604
605
605
606
607
608
609
609
610
611
612
613
614
615
615
616
617
618
619
619
620
621
622
623
624
625
625
626
627
628
629
629
630
631
632
633
634
635
635
636
637
638
639
639
640
641
642
643
644
645
645
646
647
648
649
649
650
651
652
653
654
655
655
656
657
658
659
659
660
661
662
663
664
665
665
666
667
668
669
669
670
671
672
673
674
674
675
676
677
678
678
679
679
680
681
682
683
683
684
685
686
686
687
687
688
689
689
690
691
692
692
693
693
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
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671

```

The screenshot shows two side-by-side code editors. The left editor is in 'catpoint' service mode, displaying `SecurityService.java`. The right editor is in 'udasecparent' mode, displaying the same file. A green box highlights a section of code in the left editor, and a green arrow points from this box to the corresponding code in the right editor, indicating a code change or comparison.

```
case ALARM → setAlarmStatus(AlarmStatus.PENDING_ALARM);
```

```
    /**
     * Change the activation status for the specified sensor and update alarm status if necessary.
     * @param sensor
     * @param active
     */
    public void changeSensorActivationStatus(Sensor sensor, Boolean active) {
        AlarmStatus alarmStatus = securityRepository.setAlarmStatus();
        if (alarmStatus == AlarmStatus.ALARM) {
            if (sensor.getActivate() && active) {
                handleSensorActivated();
            } else if (sensor.getActivate() && !active) {
                handleSensorDeactivated();
            } else if (sensor.getActivate() && active) {
                handleSensorActivated();
            }
        }
        sensor.setActive(active);
        securityRepository.updateSensor(sensor);
    }

    /**
     * Send an image to the SecurityService for processing. The securityService will use its provided
     * ImageService to analyze the image for cats and update the alarm status accordingly.
     * @param currentCameraImage
     */
    public void processImage(BufferedImage currentCameraImage) {
        catDetected = imageService.imageContainsCat(currentCameraImage, confidenceThreshold .50.0f);
    }
}
```

```
securityService.setAlarmStatus();
if (securityRepository.getAlarmStatus() == AlarmStatus.PENDING_ALARM) {
    setAlarmStatus(AlarmStatus.NO_ALARM);
}

/**
 * Change the activation status for the specified sensor and update alarm status if necessary.
 * @param sensor
 * @param active
 */
public void changeSensorActivationStatus(Sensor sensor, Boolean active) {
    if (sensor.getActivate() && active) {
        handleSensorActivated();
    } else if (sensor.getActivate() && !active) {
        handleSensorDeactivated();
    } else if (sensor.getActivate() || active) {
        handleSensorActivated();
    }
    sensor.setActive(active);
    securityRepository.updateSensor(sensor);
}

/**
 * Send an image to the SecurityService for processing. The securityService will use its provided
 * ImageService to analyze the image for cats and update the alarm status accordingly.
 * @param currentCameraImage
 */
public void processImage(BufferedImage currentCameraImage) {
    catDetected = imageService.imageContainsCat(currentCameraImage, .50.0f);
}

public AlarmStatus getAlarmStatus() {
    return securityRepository.getAlarmStatus();
}

public Set<Sensor> getSensors() {
    return securityRepository.getSensors();
}

public void addSensor(Sensor sensor) {
    securityRepository.addSensor(sensor);
}

public void removeSensor(Sensor sensor) {
    securityRepository.removeSensor(sensor);
}

public AlarmStatus getArmingStatus() {
}
```

```

    /*
     * Sets the current arming status for the system. Changing the arming status
     * may update both the alarm status.
     * @param armingStatus
     */
    public void setArmingStatus(ArmingStatus armingStatus) {
        if (armingStatus == ArmingStatus.ARMED_HOME || armingStatus == ArmingStatus.ARMED_AWAY) {
            if (isCatDetected()) {
                setAlarmStatus(AlarmStatus.ALARM);
            }
            changeActivationForSensors();
        }
        if (armingStatus == ArmingStatus.DISARMED) {
            setAlarmStatus(AlarmStatus.NO_ALARM);
        } else {
            ConcurrentSkipListSet<Sensor> sensors = new ConcurrentSkipListSet<>(getSensors());
            sensors.forEach(sensor -> changeSensorActivationStatus(sensor, active: false));
        }
        securityRepository.setArmingStatus(armingStatus);
        statusListeners.forEach(sl -> sl.sensorStatusChanged());
    }

    private void changeActivationForSensors() {
        Set<Sensor> sensors = getSensors().stream().map(sensor -> {
            sensor.setActive(false);
            return sensor;
        }).collect(Collectors.toSet());
        sensors.forEach(sensor -> securityRepository.updateSensor(sensor));
    }

    /**
     * Internal method that handles alarm status changes based on whether
     * the camera currently shows a cat.
     */
    @param status' tag description is missing
  
```

```

    /*
     * Internal method that handles alarm status changes based on whether
     * the camera currently shows a cat.
     * @param cat True if a cat is detected, otherwise false.
     */
    private void catDetected(Boolean cat) {
        isCatDetected = cat;
        if(cat && getArmingStatus() == ArmingStatus.ARMED_HOME) {
            setAlarmStatus(AlarmStatus.ALARM);
        } else if (cat && getSensors().stream().allMatch(s -> s.getActive() == false)) {
            setAlarmStatus(AlarmStatus.NO_ALARM);
        }
        statusListeners.forEach(sl -> sl.catDetected(cat));
    }

    //getSensors().stream().noneMatch(sensor::getActive)
    // !cat && getSensors().stream().allMatch(s -> s.getActive() == false)

    /**
     * Register the StatusListener for alarm system updates from within the SecurityService.
     * @param statusListener
     */
    3 usages
    public void addStatusListener(StatusListener statusListener) { statusListeners.add(statusListener); }

    public void removeStatusListener(StatusListener statusListener) { statusListeners.remove(statusListener); }

    /**
     * Change the alarm status of the system and notify all listeners.
     * @param status
     */
    8 usages
    public void setAlarmStatus(AlarmStatus status) {
        securityRepository.setAlarmStatus(status);
        statusListeners.forEach(sl -> sl.notify(status));
    }

    @param status' tag description is missing
  
```

```

134     * Change the activation status for the specified sensor and update alarm status
135     * @param sensor
136     * @param active
137
138     public void changeSensorActivationStatus(Sensor sensor, Boolean active) {
139
140         AlarmStatus alarmStatus = securityRepository.getAlarmStatus();
141
142         if (alarmStatus == AlarmStatus.ALARM) {
143             if(sensor.getActive() == active) {
144                 handleSensorActivated();
145             } else if (sensor.getActive() && !active) {
146                 handleSensorDeactivated();
147             } else if (!sensor.getActive() && active) {
148                 handleSensorActivated();
149             }
150
151             sensor.setActive(active);
152             securityRepository.updateSensor(sensor);
153
154         }
155
156         /* A task for the SecurityService job: periodically, the SecurityService will use the ImageService to analyze the image for cats and update the alarm status accordingly.
157         * @param currentCameraImage
158         */
159
160         public void processImage(BufferedImage currentCameraImage) {
161             catDetected(imageService.imageContainsCat(currentCameraImage, confidenceThreshold: 50.0f));
162         }
163
164         public AlarmStatus getAlarmStatus() { return securityRepository.getAlarmStatus(); }
165
166         public Set<Sensor> getSensors() { return securityRepository.getSensors(); }
167
168     }
169
170     /**
171      * A task for the SecurityService job: periodically, the SecurityService will use its provided
172      * ImageService to analyze the image for cats and update the alarm status accordingly.
173      * @param currentCameraImage
174      */
175
176     public void processImage(BufferedImage currentCameraImage) {
177         catDetected(imageService.imageContainsCat(currentCameraImage, confidenceThreshold: 50.0f));
178     }
179
180     public AlarmStatus getAlarmStatus() { return securityRepository.getAlarmStatus(); }
181
182     public Set<Sensor> getSensors() { return securityRepository.getSensors(); }
183
184     public void addSensor(Sensor sensor) { securityRepository.addSensor(sensor); }
185
186     public void removeSensor(Sensor sensor) { securityRepository.removeSensor(sensor); }
187
188     public ArmingStatus getArmingStatus() { return securityRepository.getArmingStatus(); }
189
190 }

```

```

29     * @ExtendWith(MockitoExtension.class)
30     public class SecurityServiceTest {
31
32         /* Rigorous Test :-)
33
34
35         @Test
36         private SecurityService securityService;
37
38         @Mock
39         private Sensor sensor;
40
41         @Mock
42         private ImageService imageService;
43
44         @Mock
45         private SecurityRepository securityRepository;
46
47         @Test
48         public void shouldAnswerWithTrue() { assertEquals( condition: true ); }
49
50     }
51
52     @BeforeEach
53     void init() {
54         securityService = new SecurityService(securityRepository, imageService);
55         sensor = new Sensor(random, SensorType.DOOR);
56     }
57
58     /**
59      * When alarm is armed and sensor becomes activated, put the system into pending alarm status
60      */
61
62     @Test
63     void IfAlarmsarmedandsensorbecomesactivatedputthesystemintopendingalarmstatus(){
64
65         when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_HOME);
66         when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.NO_ALARM);
67         securityService.changeSensorActivationStatus(sensor, active: true);
68         //then
69         verify(securityRepository, times( wantedNumberOfInvocations: 1)).setAlarmStatus(AlarmStatus.PENDING_ALARM);
70     }
71
72 }

```

```

198     * @ExtendWith(MockitoExtension.class)
199     public class SecurityServiceTest {
200
201         import static java.util.UUID.randomUUID;
202
203         import static org.junit.jupiter.api.Assertions.assertFalse;
204         import static org.mockito.Mockito.*;
205
206         private SecurityService securityService;
207
208         private Sensor sensor;
209
210         private final String random = UUID.randomUUID().toString();
211
212         @Mock
213         private StatusListener statusListener;
214
215         @Mock
216         private ImageService imageService;
217
218         @Mock
219         private SecurityRepository securityRepository;
220
221         @Test
222         void getNewSensor() {
223             assertEquals( new Sensor(random, SensorType.DOOR), securityService.getNewSensor() );
224
225             private Set<Sensor> getAllSensors(int count, boolean status) {
226                 Set<Sensor> sensors = new HashSet<>();
227                 for (int i = 0; i < count; i++) {
228                     sensors.add(new Sensor(random, SensorType.DOOR));
229                 }
230                 sensors.forEach(sensor -> sensor.setActive(status));
231
232                 return sensors;
233             }
234
235             @BeforeEach
236             void init() {
237                 securityService = new SecurityService(securityRepository, imageService);
238                 sensor = getNewSensor();
239             }
240
241         }
242
243         @Test
244         void ifSystemArmedAndSensorActivated_changeStatusToPending() {
245             when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_HOME);
246             when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.NO_ALARM);
247             securityService.changeSensorActivationStatus(sensor, true);
248
249             verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.PENDING_ALARM);
250         }
251
252     }
253
254 }

```

```

    securityService = new SecurityService(securityRepository, imageService);
    sensor = new Sensor(random, SensorType.DOOR);
    when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_HOME);
    when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.NO_ALARM);
    securityService.changeSensorActivationStatus(sensor, active: true);
    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.PENDING_ALARM);
}

//2. If alarm is armed and a sensor becomes activated and the system is already pending alarm, set the alarm status
@Test
void Ifalarmisarmedandasensorbecomesactivatedandthesystemisalreadypendingalarmsetthealarmstatustoalarm(){
    when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_HOME);
    when(securityRepository.setAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
    securityService.changeSensorActivationStatus(sensor, active: true);

    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
}

//3. If pending alarm and all sensors are inactive, return to no alarm state.
@Test
void Ifpendingalarmandsensorsareinactivereturntonoalarmstate(){
    when(securityService.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
    sensor.setActive(true);

    securityService.changeSensorActivationStatus(sensor, active: false);
    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
}

```

```

    void Ifalarmisarmedandasensorbecomesactivatedandthesystemisalreadypendingalarmsetthealarmstatustoalarm(){
        when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_HOME);
        when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
        securityService.changeSensorActivationStatus(sensor, active: true);

        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
    }

    //3. If pending alarm and all sensors are inactive, return to no alarm state.
    @Test
    void Ifpendingalarmandsensorsareinactivereturntonoalarmstate(){
        when(securityService.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
        sensor.setActive(true);

        securityService.changeSensorActivationStatus(sensor, active: false);
        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
    }

    //4. If alarm is active, change in sensor state should not affect the alarm state.
    @ParameterizedTest
    @ValueSource booleans = {true, false}
    void Ifalarmingactivechangesensorstatebutshouldnotaffectthealarmstate(boolean status){
        when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.ALARM);
        securityService.changeSensorActivationStatus(sensor, status);

        verify(securityRepository, never()).setAlarmStatus(any(AlarmStatus.class));
    }

    //5. If a sensor is activated while already active and the system is in pending state, change it to alarm state.
    @Test
    void Ifsensorisactivatedwhilealreadyactiveandthesystemisinpendingstatechangeittoalarmstate(){

```

```

    securityService.changeSensorActivationStatus(sensor, false);
    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);

    //3.If pending alarm and all sensors are inactive, return to no alarm state.
    @Test
    void IfPendingalarmandalloensorsareinactiveReturnnoalarmstate() {
        when(securityService.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
        sensor.setActive(true);

        securityService.changeSensorActivationStatus(sensor, false);
        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
    }

    //4.If alarm is active, change in sensor state should not affect the alarm state.
    @ParameterizedTest
    @ValueSource booleans = {true, false}
    void Ifalarmisactivechangeinsensorstateshouldnotaffectthealarmstate(boolean status) {
        when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.ALARM);
        securityService.changeSensorActivationStatus(sensor, status);

        verify(securityRepository, never()).setAlarmStatus(any(AlarmStatus.class));
    }

    //5.If a sensor is activated while already active and the system is in pending state, change it to alarm state.
    @Test
    void Ifsensorisactivatedwhilealreadyactiveandthesystemisinpendingstatechangetoalarmstate() {
        when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
        sensor.setActive(true);
        securityService.changeSensorActivationStatus(sensor, true);
        ArgumentCaptor<AlarmStatus> captor = ArgumentCaptor.ofClass(AlarmStatus.class);
        verify(securityRepository, atMostOnce()).setAlarmStatus(captor.capture());
        assertEquals(captor.getValue(), AlarmStatus.ALARM);
    }

    //6.If a sensor is deactivated while already inactive, make no changes to the alarm state.
    @Test

```

// Test 3
@ParameterizedTest
@ValueSource booleans = {true, false}
void IfPendingalarmandalloensorsareinactiveReturnnoalarmstate() {
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
 sensor.setActive(false);
 securityService.changeSensorActivationStatus(sensor);
 verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
}
// Test 4
@ParameterizedTest
@ValueSource booleans = {true, false}
void IfalarmisactivechangeinsensorstateShouldNotAffectAlarmState(boolean status) {
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.ALARM);
 securityService.changeSensorActivationStatus(sensor, status);
 verify(securityRepository, never()).setAlarmStatus(any(AlarmStatus.class));
}
// Test 5
@Test
void IfsensorActivatedWhileInactive_noChangesToAlarm() {
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
 sensor.setActive(true);
 securityService.changeSensorActivationStatus(sensor, true);
 verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
}
// Test 6
@ParameterizedTest
@EnumSource(AlarmStatus.class, names = {"NO_ALARM", "PENDING_ALARM", "ALARM"})
void IfsensorActivatedWhileInactive_noChangesToAlarmState(AlarmStatus status) {
 when(securityRepository.getAlarmStatus()).thenReturn(status);
 sensor.setActive(false);
 securityService.changeSensorActivationStatus(sensor, false);
 verify(securityRepository, never()).setAlarmStatus(any(AlarmStatus.class));
}
// Test 7
void IfImageServiceIdentifiesCatWhileArmedHome_changeStatusToAlarm() {
 BufferedImage catImage = new BufferedImage(256, 256, BufferedImage.TYPE_INT_RGB);
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.ARMED_HOME);
 when(imageService.imageContainsCat(any(), ArgumentMatchers.anyFloat())).thenReturn(true);
 securityService.processImage(catImage);
 verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
}
// Test 8
@Test
void IfImageServiceIdentifiesNoCatImage_changeStatusToNoAlarmAlongSensorsNotActive() {
 Set<SecuritySensor> getSensors = new HashSet<SecuritySensor>();
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.NO_ALARM);
 when(imageService.imageContainsCat(any(), ArgumentMatchers.anyFloat())).thenReturn(false);
 securityService.processImage(mock(BufferedImage.class));
 verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
}

```

    //4.If alarm is active, change in sensor state should not affect the alarm state.
    @ParameterizedTest
    @ValueSource booleans = {true, false}
    void IfalarmisactivechangeinsensorstateShouldNotAffectAlarmState(boolean status) {
        when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.ALARM);
        securityService.changeSensorActivationStatus(sensor, status);

        verify(securityRepository, never()).setAlarmStatus(any(AlarmStatus.class));
    }

    //5.If a sensor is activated while already active and the system is in pending state, change it to alarm state.
    @Test
    void Ifsensorisactivatedwhilealreadyactiveandthesystemisinpendingstatechangetoalarmstate() {
        when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
        sensor.setActive(true);
        securityService.changeSensorActivationStatus(sensor, true);
        ArgumentCaptor<AlarmStatus> captor = ArgumentCaptor.ofClass(AlarmStatus.class);
        verify(securityRepository, atMostOnce()).setAlarmStatus(captor.capture());
        assertEquals(captor.getValue(), AlarmStatus.ALARM);
    }

    //6.If a sensor is deactivated while already inactive, make no changes to the alarm state.
    @Test
    void Ifsensorisdeactivatedwhilealreadyinactive makenochanges to the alarm state() {
        sensor.setActive(false);
        securityService.changeSensorActivationStatus(sensor, false);

        verify(securityRepository, never()).setAlarmStatus(AlarmStatus.ALARM);
        verify(securityRepository, never()).setAlarmStatus(AlarmStatus.PENDING_ALARM);
        verify(securityRepository, never()).setAlarmStatus(AlarmStatus.NO_ALARM);
    }

    //7.If the image service identifies an image containing a cat while the system is armed-home, put the system into a
    @Test
    void Iftheimageserviceidentifiesanimagecontainingacatwhilethesystemisarmedhomeputthesystemtoalarmstatus() {
        BufferedImage catImage = new BufferedImage(256, 256, BufferedImage.TYPE_INT_RGB);
        when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.ARMED_HOME);
    }

```

// Test 4
@ParameterizedTest
@ValueSource booleans = {true, false}
void IfalarmisactivechangeinsensorstateShouldNotAffectAlarmState(boolean status) {
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.ALARM);
 securityService.changeSensorActivationStatus(sensor, status);
 verify(securityRepository, never()).setAlarmStatus(any(AlarmStatus.class));
}
// Test 5
@Test
void IfsensorActivatedWhileInactive_noChangesToAlarm() {
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.PENDING_ALARM);
 sensor.setActive(true);
 securityService.changeSensorActivationStatus(sensor, true);
 verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
}
// Test 6
@ParameterizedTest
@EnumSource(AlarmStatus.class, names = {"NO_ALARM", "PENDING_ALARM", "ALARM"})
void IfsensorActivatedWhileInactive_noChangesToAlarmState(AlarmStatus status) {
 when(securityRepository.getAlarmStatus()).thenReturn(status);
 sensor.setActive(false);
 securityService.changeSensorActivationStatus(sensor, false);
 verify(securityRepository, never()).setAlarmStatus(any(AlarmStatus.class));
}
// Test 7
@Test
void IfImageServiceIdentifiesCatWhileArmedHome_changeStatusToAlarm() {
 BufferedImage catImage = new BufferedImage(256, 256, BufferedImage.TYPE_INT_RGB);
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.ARMED_HOME);
 when(imageService.imageContainsCat(any(), ArgumentMatchers.anyFloat())).thenReturn(true);
 securityService.processImage(catImage);
 verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
}
// Test 8
@Test
void IfImageServiceIdentifiesNoCatImage_changeStatusToNoAlarmAlongSensorsNotActive() {
 Set<SecuritySensor> getSensors = new HashSet<SecuritySensor>();
 when(securityRepository.getAlarmStatus()).thenReturn(AlarmStatus.NO_ALARM);
 when(imageService.imageContainsCat(any(), ArgumentMatchers.anyFloat())).thenReturn(false);
 securityService.processImage(mock(BufferedImage.class));
 verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
}

```

    // If a sensor is deactivated while already inactive, make no changes to the alarm state.
    @Test
    void IfASensorIsDeactivatedWhileAlreadyInactiveMakesNoChangesToTheAlarmState() {
        sensor.setActive(false);
        securityService.changeSensorActivationStatus(sensor, false);

        verify(securityRepository, never()).setAlarmStatus(AlarmStatus.ALARM);
        verify(securityRepository, never()).setAlarmStatus(AlarmStatus.PENDING_ALARM);
        verify(securityRepository, never()).setAlarmStatus(AlarmStatus.NO_ALARM);
    }

    // If the image service identifies an image containing a cat while the system is armed-home, put the system into a
    @Test
    void IfTheImageServiceIdentifiesAnImageContainingACatWhileTheSystemIsArmedHomePutsTheSystemIntoAlarmStatus() {
        BufferedImage catImage = new BufferedImage(256, height, TYPE_INT_RGB);
        when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_HOME);
        when(imageService.imageContainsCat(any(), ArgumentMatchers.anyFloat())).thenReturn(true);
        securityService.processImage(catImage);

        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
    }

    // If the image service identifies an image that does not contain a cat, change the status to no alarm as long as
    @Test
    void IfTheImageServiceIdentifiesAnImageThatDoesNotContainACatChangesTheStatusToNoAlarmAsLongAsTheSensorsAreInactive() {
        when(imageService.imageContainsCat(any(BufferedImage.class), anyFloat())).thenReturn(false);
        securityService.processImage(mock(BufferedImage.class));

        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
    }

    // If the system is disarmed, set the status to no alarm
    @Test
    void IfTheSystemIsDisarmedSetsTheStatusToNoAlarm() {
        securityService.setArmingStatus(ArmingStatus.DISARMED);
        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
    }

    // If the system is armed, reset all sensors to inactive
    @ParameterizedTest
    @EnumSource(value = ArmingStatus.class, names = {"ARMED_HOME", "ARMED_AWAY"})
    void IfTheSystemIsArmedResetsAllSensorsToInactive(ArmingStatus status) {
        securityService.setArmingStatus(status);
        securityService.getSensor().forEach(sensor -> {
            assert Boolean.FALSE.equals(sensor.getActive());
        });
    }

    // If the system is armed-home while the camera shows a cat, set the alarm status to alarm.
    @Test
    void IfTheSystemIsArmedHomeWhileTheCameraShowsACatSetsTheAlarmStatusToAlarm() {
        assertBoolean.TRUE.equals(Captor.ofValue(), AlarmStatus.ALARM);
    }
}

```

```

    // If the image service identifies an image containing a cat while the system is armed-home, put the system into a
    @Test
    void IfTheImageServiceIdentifiesAnImageContainingACatWhileTheSystemIsArmedHomePutsTheSystemIntoAlarmStatus() {
        BufferedImage catImage = new BufferedImage(256, height, TYPE_INT_RGB);
        when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_HOME);
        when(imageService.imageContainsCat(any(), ArgumentMatchers.anyFloat())).thenReturn(true);
        securityService.processImage(catImage);

        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
    }

    // If the image service identifies an image that does not contain a cat, change the status to no alarm as long as
    @Test
    void IfTheImageServiceIdentifiesAnImageThatDoesNotContainACatChangesTheStatusToNoAlarmAsLongAsTheSensorsAreInactive() {
        when(imageService.imageContainsCat(any(BufferedImage.class), anyFloat())).thenReturn(false);
        securityService.processImage(mock(BufferedImage.class));

        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
    }

    // If the system is disarmed, set the status to no alarm
    @Test
    void IfTheSystemIsDisarmedSetsTheStatusToNoAlarm() {
        securityService.setArmingStatus(ArmingStatus.DISARMED);
        verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
    }

    // If the system is armed, reset all sensors to inactive
    @Test
    void IfTheSystemIsArmedResetsAllSensorsToInactive() {
        ArmingStatus value = ArmingStatus.class, names = {"ARMED_HOME", "ARMED_AWAY"};
        void IfTheSystemIsArmedResetsAllSensorsToInactive(ArmingStatus status) {
            securityService.setArmingStatus(status);
            securityService.getSensor().forEach(sensor -> {
                assert Boolean.FALSE.equals(sensor.getActive());
            });
        }

        // If the system is armed-home while the camera shows a cat, set the alarm status to alarm.
        @Test
        void IfTheSystemIsArmedHomeWhileTheCameraShowsACatSetsTheAlarmStatusToAlarm() {
            assertBoolean.TRUE.equals(Captor.ofValue(), AlarmStatus.ALARM);
        }
}

```

```

    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
}

// If the image service identifies an image that does not contain a cat, change the status to no alarm as long as the system is armed
@Test
void IfTheImageServiceIdentifiesCatThenDoNotSetTheStatusToAlarmIfTheSensorsAreNotActive() {
    when(imageService.imageContainsCat(any(BufferedImage.class), anyFloat())).thenReturn(false);

    securityService.processImage(mock(BufferedImage.class));

    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
}

// If the system is armed, reset all sensors to inactive
@ParameterizedTest
@EnumSource(value = ArmingStatus.class, names = {"ARMED_HOME", "ARMED_AWAY"})
void IfTheSystemIsArmedSetAllSensorsToInactive(ArmingStatus status) {
    securityService.setArmingStatus(status);
    securityService.getSensor().forEach(sensor -> {
        assertFalse(sensor.getActive());
    });
}

// If the system is armed-home while the camera shows a cat, set the alarm status to alarm.
@Test
void IfTheSystemIsArmedHomeWhileTheCameraShowsACatSetTheAlarmStatusToAlarm() {
    BufferedImage catImage = new BufferedImage(100, 100, TYPE_INT_RGB);

    when(imageService.imageContainsCat(any(BufferedImage.class), anyFloat())).thenReturn(true);
    when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.DISARMED);
    securityService.processImage(catImage);
    securityService.setArmingStatus(ArmingStatus.ARMED_HOME);

    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
}

```

```

    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
}

// If the system is disarmed, set the status to no alarm
@Test
void IfTheSystemIsDisarmedSetTheStatusToNoAlarm() {
    securityService.setArmingStatus(ArmingStatus.DISARMED);
    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
}

// If the system is armed while the camera shows a cat, set the alarm status to alarm.
@Test
void IfTheSystemIsArmedWhileTheCameraShowsACatSetTheAlarmStatusToAlarm() {
    BufferedImage catImage = new BufferedImage(100, 100, TYPE_INT_RGB);

    when(imageService.imageContainsCat(any(BufferedImage.class), anyFloat())).thenReturn(true);
    when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.DISARMED);
    securityService.processImage(catImage);
    securityService.setArmingStatus(ArmingStatus.ARMED_HOME);

    verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
}

```

```
test > java > com > udacity > service > SecurityServiceTest
```

```
SecurityService.java x SecurityServiceTest.java x Sensor.java x
```

```
File Edi View Navigate Code Refactor Build Run Tools VCS Window Help Parent -
```

```
Project
```

```
140
```

```
141     verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
```

```
142 }
```

```
143 // 9. If the system is disarmed, set the status to no alarm
```

```
144 @Test
```

```
145 void ifTheSystemIsDisarmedSetTheStatusToNoAlarm(){
```

```
146     securityService.setArmingStatus(ArmingStatus.DISARMED);
```

```
147     verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.NO_ALARM);
```

```
148 }
```

```
149 //10. If the system is armed, reset all sensors to inactive
```

```
150
```

```
151 @ParameterizedTest
```

```
152 @EnumSource(value = ArmingStatus.class, names = {"ARMED_HOME", "ARMED_AWAY"})
```

```
153 void ifTheSystemIsArmedResetAllSensorsToInactive(ArmingStatus status){
```

```
154     securityService.setArmingStatus(status);
```

```
155     securityService.getSensor().forEach(sensor ->
```

```
156         assertEquals(Boolean.FALSE.equals(sensor.getActive()));
```

```
157     );
```

```
158 }
```

```
159 //11. If the system is armed home while the camera shows a cat, set the alarm status to alarm.
```

```
160 @Test
```

```
161 void ifTheSystemIsArmedHomeWhileTheCameraShowsACatSetTheAlarmStatusToAlarm(){
```

```
162     BufferedImage catImage = new BufferedImage( width: 1, height: 1, TYPE_INT_RGB);
```

```
163
```

```
164     when(imageService.imageContainsCat(any(BufferedImage.class),anyFloat())).thenReturn(true);
```

```
165
```

```
166     when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.DISARMED);
```

```
167     securityService.setImage(catImage);
```

```
168     securityService.setArmingStatus(ArmingStatus.ARMED_HOME);
```

```
169
```

```
170     verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
```

```
171 }
```

```
172
```

```
173 //Extra tests to cover the whole SecurityService class
```

```
174 //Status Listener test
```

```
175 @Test
```

```
176 void addAndRemoveStatusListener() {
```

```
177     securityService.addStatusListener(statusListener);
```

```
178     securityService.removeStatusListener(statusListener);
```

```
179 }
```

```
180
```

```
181 //Sensor Listener test
```

```
182 @Test
```

```
183 void addAndRemoveSensor() {
```

```
184     securityService.addSensor(sensor);
```

```
185     securityService.removeSensor(sensor);
```

```
186 }
```

```
187 //System disarmed Test
```

```
188 @ParameterizedTest
```

```
189 @EnumSource(value = AlarmStatus.class, names = {"NO_ALARM", "PENDING_ALARM"})
```

```
190 void ifTheSystemIsDisarmedSetTheStatusToNoAlarm(AlarmStatus status){
```

```
191     when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.DISARMED);
```

```
192     securityService.setAlarmStatus(status);
```

```
193     verify(securityRepository, times(1)).setAlarmStatus(status);
```

```
194 }
```

```
195
```

```
196 //System armed Test
```

```
197 @ParameterizedTest
```

```
198 @EnumSource(value = AlarmStatus.class, names = {"NO_ALARM", "PENDING_ALARM"})
```

```
199 void ifTheSystemIsArmedSetTheStatusToPendingAlarm(AlarmStatus status){
```

```
200     when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_HOME);
```

```
201     securityService.setAlarmStatus(status);
```

```
202     verify(securityRepository, times(1)).setAlarmStatus(status);
```

```
203 }
```

```
204
```

```
205 //System armed away Test
```

```
206 @ParameterizedTest
```

```
207 @EnumSource(value = AlarmStatus.class, names = {"NO_ALARM", "PENDING_ALARM"})
```

```
208 void ifTheSystemIsArmedAwaySetTheStatusToPendingAlarm(AlarmStatus status){
```

```
209     when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.ARMED_AWAY);
```

```
210     securityService.setAlarmStatus(status);
```

```
211     verify(securityRepository, times(1)).setAlarmStatus(status);
```

```
212 }
```

```
213
```

```
214 //System armed home while the image service identifies cat change status to alarm
```

```
215 @Test
```

```
216 void ifTheSystemIsArmedHomeWhileTheImageServiceIdentifiesCat_changeStatusToAlarm() {
```

```
217     BufferedImage catImage = new BufferedImage(256, 256, BufferedImage.TYPE_INT_RGB);
```

```
218     when(imageService.imageContainsCat(any(BufferedImage.class),anyFloat())).thenReturn(true);
```

```
219     when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.DISARMED);
```

```
220     securityService.setImage(catImage);
```

```
221     securityService.setArmingStatus(ArmingStatus.ARMED_HOME);
```

```
222
```

```
223     verify(securityRepository, times(1)).setAlarmStatus(AlarmStatus.ALARM);
```

```
224 }
```

```
225
```

The screenshot shows two browser tabs side-by-side, both displaying the same Java code for a test class named `SecurityServiceTest.java`. The code is annotated with JUnit 5 assertions and includes several test methods. A green rectangular box highlights a specific section of the code.

```
test > java > com > udacity > service > SecurityServiceTest > Current File
```

```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help Parent - x
```

```
SecurityService.java x SecurityServiceTest.java x Sensor.java x
```

```
148     verify(securityRepository, times(wantedNumberOfInvocations)).setAlarmStatus(AlarmStatus.NO_ALARM);
```

```
149 }
```

```
150 // 9. If the system is disarmed, set the status to no alarm
```

```
151 @Test
```

```
152 void ifTheSystemIsDisarmedSetTheStatusToNoAlarm(){
```

```
153     securityService.setArmingStatus(ArmingStatus.DISARMED);
```

```
154     verify(securityRepository, times(wantedNumberOfInvocations)).setAlarmStatus(AlarmStatus.NO_ALARM);
```

```
155 }
```

```
156 //10. If the system is armed, reset all sensors to inactive
```

```
157 @ParameterizedTest
```

```
@EnumSource(value = ArmingStatus.class, names = {"ARMED_HOME", "ARMED_AWAY"})
```

```
158 void ifTheSystemIsArmedResetAllSensorsToInactive(ArmingStatus status){
```

```
159     securityService.setArmingStatus(status);
```

```
160     securityService.getSensors().forEach(sensor ->
```

```
161         assertEquals(Boolean.FALSE.equals(sensor.getActive()));
```

```
162     );
```

```
163 }
```

```
164 }
```

```
165 }
```

```
166 //11. If the system is armed while the camera shows a cat, set the alarm status to alarm.
```

```
167 @Test
```

```
168 void ifTheSystemIsArmedWhileTheCameraShowsACatSetTheAlarmStatusToAlarm(){
```

```
169     BufferedImage catImage = new BufferedImage(100, 100, TYPE_INT_RGB);
```

```
170     when(imageService.imageContainsCat(any(BufferedImage.class), anyFloat())).thenReturn(true);
```

```
171     when(securityRepository.getArmingStatus()).thenReturn(ArmingStatus.DISARMED);
```

```
172     securityService.processImage(catImage);
```

```
173     securityService.setArmingStatus(ArmingStatus.ARMED_HOME);
```

```
174 }
```

```
175 verify(securityRepository, times(wantedNumberOfInvocations)).setAlarmStatus(AlarmStatus.ALARM);
```

```
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 }
```

So, as can be seen above, you have basically plagiarized the logic and code of the classes:

- SecurityService.java
 - SecurityServiceTest.java

From the repositories that were pointed to as sources.

Please, send an **original project**, so it can be **evaluated**

I wish you a great day

RESUBMIT

 DOWNLOAD PROJECT FILES

Learn the best practices for revising and resubmitting your project.

[RETURN TO PATH](#)
