

```
1 import java.sql.Array;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.sql.Statement;
8 import java.util.ArrayList;
9 import java.util.Arrays;
10
11 public class Database {
12     private Connection connect() {
13         // SQLite connection string
14         String url = "jdbc:sqlite:db.sqlite";
15         Connection conn = null;
16         try {
17             conn = DriverManager.getConnection(url);
18         } catch (SQLException e) {
19             // System.out.println(e.getMessage());
20         }
21         return conn;
22     }
23
24     public void migrate() {
25         String sql = "CREATE TABLE IF NOT EXISTS schedule_table (\n"
26             + " tuid integer PRIMARY KEY,\n"
27             + " course_tuid text NOT NULL,\n"
28             + " classroom_tuid text NOT NULL,\n"
29             + " professor_tuid text NOT NULL,\n"
30             + " section integer NOT NULL,\n"
31             + " start_time text NOT NULL,\n"
32             + " end_time text NOT NULL,\n"
33             + " days text NOT NULL);\n";
34
35         try (Connection conn = this.connect();
36             Statement stmt = conn.createStatement()) {
37             // create a new table
38             // System.out.println("Creating schedule table...");
39             stmt.execute(sql);
40         } catch (SQLException e) {
41             System.out.println(e.getMessage());
42         }
43
44         sql = "CREATE TABLE IF NOT EXISTS professors_table (\n"
45             + " tuid integer PRIMARY KEY,\n"
46             + " professor_name text NOT NULL);\n";
47
48         try (Connection conn = this.connect();
49             Statement stmt = conn.createStatement()) {
50             // create a new table
```

```
50         // create a new table
51         // System.out.println("Creating professors table...");
52         stmt.execute(sql);
53     } catch (SQLException e) {
54         // System.out.println(e.getMessage());
55     }
56
57     sql = "CREATE TABLE IF NOT EXISTS classroom_table (\n"
58         + " tuid integer PRIMARY KEY,\n"
59         + " classroom_name text NOT NULL,\n"
60         + " capacity integer NOT NULL);";
61
62     try (Connection conn = this.connect();
63         Statement stmt = conn.createStatement()) {
64         // create a new table
65         // System.out.println("Creating classroom table...");
66         stmt.execute(sql);
67     } catch (SQLException e) {
68         // System.out.println(e.getMessage());
69     }
70
71     sql = "CREATE TABLE IF NOT EXISTS courses_table (\n"
72         + " tuid integer PRIMARY KEY,\n"
73         + " course_id text NOT NULL,\n"
74         + " course_title text NOT NULL,\n"
75         + " credit_hours integer NOT NULL);";
76
77     try (Connection conn = this.connect();
78         Statement stmt = conn.createStatement()) {
79         // create a new table
80         // System.out.println("Creating courses table...");
81         stmt.execute(sql);
82     } catch (SQLException e) {
83         // System.out.println(e.getMessage());
84     }
85 }
86
87 public void drop() {
88     String sql = "DROP TABLE IF EXISTS schedule_table;";
89
90     try (Connection conn = this.connect();
91         Statement stmt = conn.createStatement()) {
92         // create a new table
93         // System.out.println("Dropping schedule table...");
94         stmt.execute(sql);
95     } catch (SQLException e) {
96         // System.out.println(e.getMessage());
97     }
98
99     sql = "DROP TABLE IF EXISTS professors_table;";
```

```
100
101     try (Connection conn = this.connect();
102           Statement stmt = conn.createStatement()) {
103         // create a new table
104         // System.out.println("Dropping professors table...");
105         stmt.execute(sql);
106     } catch (SQLException e) {
107         // System.out.println(e.getMessage());
108     }
109
110     sql = "DROP TABLE IF EXISTS courses_table;";
111
112     try (Connection conn = this.connect();
113           Statement stmt = conn.createStatement()) {
114         // create a new table
115         // System.out.println("Dropping courses table...");
116         stmt.execute(sql);
117     } catch (SQLException e) {
118         // System.out.println(e.getMessage());
119     }
120
121     sql = "DROP TABLE IF EXISTS classroom_table;";
122
123     try (Connection conn = this.connect();
124           Statement stmt = conn.createStatement()) {
125         // create a new table
126         // System.out.println("Dropping classroom table...");
127         stmt.execute(sql);
128     } catch (SQLException e) {
129         // System.out.println(e.getMessage());
130     }
131 }
132
133 public void seed() {
134     // Seed classrooms
135     String sql = "INSERT INTO classroom_table(classroom_name,
136 capacity)\n"
137         + " VALUES('A', 30),\n"
138         + " ('B', 25),\n"
139         + " ('C', 20);";
140
141     try (Connection conn = this.connect();
142           Statement stmt = conn.createStatement()) {
143         // create a new table
144         // System.out.println("Seeding classroom table...");
145         stmt.execute(sql);
146     } catch (SQLException e) {
147         // System.out.println(e.getMessage());
148     }
```

```
149 // Seed courses
150 sql = "INSERT INTO courses_table(course_id, course_title,
credit_hours)\n"
151 + " VALUES('CS 101', 'Intro to Computer Science', 3),\n"
152 + " ('CS 105', 'Computers and Programming', 4),\n"
153 + " ('CSC 105', 'Computers and Programming', 4), \n"
154 + " ('CSC 107', 'Introduction to Code Preparation', 1), \n"
155 + " ('CSC 116', 'Programming I', 4), \n"
156 + " ('CSC 216', 'Programming II', 4), \n"
157 + " ('CSC 227', 'Commenting and Naming Conventions', 2), \n"
158 + " ('CSC 316', 'Data Structures \u0026 Algorithms', 4), \n"
159 + " ('CSC 416', 'Advanced Algorithm Analysis', 3), \n"
160 + " ('CSC 211', 'Introductory .NET Development', 3), \n"
161 + " ('CSC 311', 'Advanced .NET Development', 4), \n"
162 + " ('CSC 313', 'Real World Application Development', 3),
\n"
163 + " ('CSC 411', 'Data Driven Systems', 3), \n"
164 + " ('CSC 412', 'Sensor Systems', 3), \n"
165 + " ('CSC 413', 'Artificial Intelligence Systems', 3), \n"
166 + " ('CSC 496', 'Software Engineering I', 4), \n"
167 + " ('CSC 497', 'Software Engineering II', 4), \n"
168 + " ('CSC 498', 'Software Engineering III', 4);";
169
170 try (Connection conn = this.connect();
171      Statement stmt = conn.createStatement()) {
172     // create a new table
173     // System.out.println("Seeding courses table...");
174     stmt.execute(sql);
175 } catch (SQLException e) {
176     // System.out.println(e.getMessage());
177 }
178
179 // Seed professors
180 sql = "INSERT INTO professors_table(professor_name)\n"
181 + " VALUES('James'),\n"
182 + " ('Smith'),\n"
183 + " ('Jones'),\n"
184 + " ('Vasquez'),\n"
185 + " ('Abdul'),\n"
186 + " ('Thomas');";
187
188 try (Connection conn = this.connect();
189      Statement stmt = conn.createStatement()) {
190     // create a new table
191     // System.out.println("Seeding professors table...");
192     stmt.execute(sql);
193 } catch (SQLException e) {
194     // System.out.println(e.getMessage());
195 }
196 }
```

```
197  
198 public void close() {  
199     // End connection to database  
200     Connection conn = this.connect();  
201     try {  
202         if (conn != null) {  
203             conn.close();  
204         }  
205     } catch (SQLException ex) {  
206         // System.out.println(ex.getMessage());  
207     }  
208 }  
209  
210 public void insertScheduleFile(ScheduleFile sf) {  
211     // Print the schedule file  
212     // System.out.println(sf.toString());  
213     // System.out.println(sf.getCourseName());  
214  
215     // Initialize variables for inserting into database  
216     Course course = findCourse(sf.getCourseName());  
217     Professor professor = findProfessor(sf.getProfessorName());  
218  
219     // Print the scheduleFile  
220     // System.out.println(sf.toString());  
221  
222     // Print the course  
223     // System.out.println(course.toString());  
224  
225     // Print the professor  
226     // System.out.println(professor.toString());  
227  
228     // Okay... So now we have the professor, course, and  
229     // schedule file information  
230     // Now we query the schedule_table to see if any courses  
231     // are already scheduled for the same time.  
232  
233     // Create a new schedule object using the course, professor,  
234     // and schedule file information  
235     Schedule schedule = new Schedule(course, professor, sf);  
236  
237     // Create an arraylist of schedules  
238     ArrayList<Schedule> schedules = new ArrayList<Schedule>();  
239  
240     schedules = findSchedule(sf.getStartTime(), sf.getEndTime(),  
241 sf.getDays());  
242  
243     // Find other instances of the course to determine the  
244     // section number  
245     int section = findSection(course.getId());
```

```
245
246     schedule.setSection(section);
247
248     // Check the size of the schedules arraylist
249     // If it's 0, then we can insert the schedule
250     // If it's not 0, then we need to find an alternate time
251     if (schedules.size() == 0) {
252         System.out.println("No conflicts found. Inserting schedule...");
253
254         // Then we should be good to schedule what ever
255         // classroom is available.
256         // So we'll query the classroom_table to find
257         // A and assign the tuid to the schedule
258         schedule.setClassroomId(findClassroomTuid("A"));
259
260         // Insert the schedule
261         System.out.println("Inserting schedule... NO CONFLICTS");
262         insertSchedule(schedule);
263     } else {
264         System.out.println("There is a conflict with the schedule");
265         // And here's where the real fun begins
266         // Everything has been all fun and games till now.
267         String[] alternateTime = findAlternateTime(schedule, course);
268         if (alternateTime[0] == null) {
269             System.out.println("No alternate time found");
270         } else {
271             // Update the schedule object with
272             // the newly found data
273             schedule.setStartTime(alternateTime[0]);
274             schedule.setEndTime(alternateTime[1]);
275             schedule.setDays(alternateTime[2]);
276             schedule.setClassroomId(Integer.parseInt(alternateTime[3]));
277
278             System.out.println("Class rescheduled to " +
279 schedule.getDays() + " from " + schedule.getStartTime()
280             + " to " + schedule.getEndTime() + " in classroom "
281 + schedule.getClassroomId());
282
283             insertSchedule(schedule);
284         }
285     }
286 }
287
288 private Course findCourse(String name) {
289     // Find the course from the courses_table
290     // by using the course_title and return the entire row
291
292     String sql = "SELECT * FROM courses_table WHERE course_id = ?";
293
294     try (Connection conn = this.connect();
```

```
293         PreparedStatement stmt = conn.prepareStatement(sql)) {
294
295         // set the value
296         stmt.setString(1, name);
297
298         ResultSet rs = stmt.executeQuery();
299
300         // Check if the course was found
301         if (rs.next()) {
302             // Create a new course object
303             Course course = new Course(rs.getInt(1), rs.getString(2),
rs.getString(3), rs.getInt(4));
304
305             return course;
306         } else {
307             System.out.println("Course not found");
308             return null;
309         }
310     } catch (SQLException e) {
311         // System.out.println(e.getMessage());
312     }
313     return null;
314 }
315
316 private Professor findProfessor(String name) {
317     // Find the professor from the professors_table
318     // by using the professor_name and return the entire row
319
320     String sql = "SELECT * FROM professors_table WHERE professor_name =
?;";
321
322     try (Connection conn = this.connect();
323         PreparedStatement stmt = conn.prepareStatement(sql)) {
324
325         // set the value
326         stmt.setString(1, name);
327
328         ResultSet rs = stmt.executeQuery();
329
330         // Check if the professor was found
331         if (rs.next()) {
332             // Create a new professor object
333             Professor professor = new Professor(rs.getInt(1),
rs.getString(2));
334
335             return professor;
336         } else {
337             System.out.println("Professor not found");
338             return null;
339         }
340     }
```

```
340     } catch (SQLException e) {
341         // System.out.println(e.getMessage());
342     }
343     return null;
344 }
345
346 private ArrayList<Schedule> findSchedule(String startTime, String
endTime, String days) {
347
348     // Query Summed Up:
349     // Select everything from the schedule where
350     // the start time < the GIVEN end time
351     // and the end time > the GIVEN start time
352     // and the days are the same
353     // AND THEN join the classroom table to actually
354     // get the classroom name
355     String sql = "SELECT * FROM schedule_table LEFT JOIN classroom_table
ON schedule_table.classroom_tuid = classroom_table.tuid WHERE
time(start_time) < time(?) AND time(end_time)> time(?) AND days IN (?);";
356     ArrayList<Schedule> schedules = new ArrayList<Schedule>();
357
358     try (Connection conn = this.connect();
359         PreparedStatement stmt = conn.prepareStatement(sql)) {
360
361         // set the value
362         stmt.setString(1, endTime);
363         stmt.setString(2, startTime);
364         stmt.setString(3, days);
365
366         ResultSet rs = stmt.executeQuery();
367
368         System.out.println("Searching for schedule conflicts...");
369
370         // Check if the professor was found
371         while (rs.next()) {
372             // Add the results to the schedules arraylist
373             schedules.add(new Schedule(rs.getInt("tuid"),
rs.getInt("course_tuid"), rs.getInt("classroom_tuid"),
374             rs.getInt("professor_tuid"), rs.getInt("section"),
rs.getString("start_time"),
375             rs.getString("end_time"), rs.getString("days")));
376         }
377
378         return schedules;
379     } catch (SQLException e) {
380         // System.out.println(e.getMessage());
381     }
382
383     return null;
384 }
```



```
385  
386 private void insertSchedule(Schedule schedule) {  
387     // Insert the schedule into the schedule_table  
388  
389     String sql = "INSERT INTO schedule_table(course_tuid,  
classroom_tuid, professor_tuid, section, start_time, end_time, days)  
VALUES(?, ?, ?, ?, ?, ?, ?);";  
390  
391     try (Connection conn = this.connect();  
392         PreparedStatement stmt = conn.prepareStatement(sql)) {  
393  
394         // Set the values  
395         stmt.setInt(1, schedule.getCourseId());  
396         stmt.setInt(2, schedule.getClassroomId());  
397         stmt.setInt(3, schedule.getProfessorId());  
398         stmt.setInt(4, schedule.getSection());  
399         stmt.setString(5, schedule.getStartTime());  
400         stmt.setString(6, schedule.getEndTime());  
401         stmt.setString(7, schedule.getDays());  
402  
403         // Execute the query  
404         stmt.executeUpdate();  
405  
406     } catch (SQLException e) {  
407         // System.out.println(e.getMessage());  
408     }  
409  
410 }  
411  
412 private int findSection(int courseTuid) {  
413     // Find the section number for the course  
414     // by using the course_tuid and return the section number  
415  
416     String sql = "SELECT * FROM schedule_table WHERE course_tuid = ?;";  
417  
418     try (Connection conn = this.connect();  
419         PreparedStatement stmt = conn.prepareStatement(sql)) {  
420  
421         // set the value  
422         stmt.setInt(1, courseTuid);  
423  
424         ResultSet rs = stmt.executeQuery();  
425  
426         // Get the total count of the results  
427         int count = 1;  
428  
429         // Loop through all the results and  
430         // increment the count  
431         while (rs.next()) {  
432             count++;  
433         }  
434     }  
435     return count;  
436 }
```

```
432         count++;
433     }
434
435     // System.out.println("TOTAL CURRENT ENTRIES: " + count);
436
437     return count;
438 } catch (SQLException e) {
439     // System.out.println(e.getMessage());
440 }
441 return -1;
442 }
443
444 // Return the classroom tuid from the classroom name
445 private int findClassroomTuid(String classroomName) {
446     // Find the classroom from the classroom_table
447     // by using the classroom_name and return the tuid
448
449     String sql = "SELECT * FROM classroom_table WHERE classroom_name =
450     ?;";
451
452     try (Connection conn = this.connect();
453         PreparedStatement stmt = conn.prepareStatement(sql)) {
454
455         // set the value
456         stmt.setString(1, classroomName);
457
458         ResultSet rs = stmt.executeQuery();
459
460         // Check if the professor was found
461         if (rs.next()) {
462             // Create a new professor object
463             return rs.getInt(1);
464         } else {
465             System.out.println("Classroom not found");
466             return -1;
467         }
468     } catch (SQLException e) {
469         // System.out.println(e.getMessage());
470     }
471     return -1;
472 }
473
474 private String[] findAlternateTime(Schedule schedule, Course course) {
475     // Okay so first we're going to check the credit hours
476     // to determine what the available days are.
477
478     // If credit hours are 3 or 4, then available days are
479     // MW, TR, or F
480     // If credit hours are 1 or 2, then available days are
481     // M,T,W,R,F
```

```
481         if (course.getCreditHours() == 4) {
482             // Cracks knuckles... Here we go...
483
484             // An array of the available start times
485             String[] startTimes = { "08:30:00", "10:30:00", "12:30:00",
486 "14:30:00" };
487             // An array of the available end times
488             String[] endTimes = { "10:30:00", "12:30:00", "14:30:00",
489 "16:30:00" };
490
491             // An array of the available Friday start times
492             String[] fridayStartTimes = { "08:30:00", "09:30:00",
493 "10:30:00", "11:30:00", "12:30:00" };
494             // An array of the available Friday end times
495             String[] fridayEndTimes = { "10:30:00", "11:30:00", "12:30:00",
496 "13:30:00", "14:30:00" };
497
498             String currentTime = schedule.getStartTime();
499             // Get the index of the currentTime within the startTimes array
500             int originalIndex =
501 Arrays.asList(startTimes).indexOf(currentTime);
502             int index = originalIndex + 1;
503             boolean continueExec = true;
504             String[] date = new String[4];
505             ArrayList<Schedule> schedules;
506             // Making a copy of the currently selected days
507             // So I can reference later what day I'm switching the
508             // times at
509             String originalDate = schedule.getDays();
510             while (continueExec) {
511                 // Check the same date for a different room
512                 schedules = findSchedule(schedule.getStartTime(),
513 schedule.getEndTime(), schedule.getDays());
514
515                 // We shouldn't have to worry about schedules being empty
516                 // as we've already checked for conflicts
517
518                 System.out.println("Searching day " + schedule.getDays() + "
519 at time " + schedule.getStartTime() + " - "
520 + schedule.getEndTime());
521
522                 // Loop through and create a csv
523                 // of the used classrooms
524                 String usedClassrooms = "";
525                 for (Schedule s : schedules) {
526                     // If last index, don't add comma
527                     if (schedules.indexOf(s) == schedules.size() - 1) {
528                         usedClassrooms += s.getClassroomId();
529                     }
530                 }
531             }
532         }
```

```
524         } else {
525             usedClassrooms += s.getClassroomId() + ",";
526         }
527         // System.out.println("Unavailable classroom: " +
s.getClassroomId());
528     }
529
530     // Query the classroom_table to get all the classrooms
531     // except for the ones that are already being used
532     ArrayList<Integer> availableClassrooms =
findAvailableClassrooms(usedClassrooms);
533
534     // If classrooms available, then set the classroom id
535     // and return the new schedule
536     if (availableClassrooms.size() > 0) {
537         // System.out.println("Available classrooms: " +
availableClassrooms);
538         date[0] = schedule.getStartTime();
539         date[1] = schedule.getEndTime();
540         date[2] = schedule.getDays();
541         date[3] = availableClassrooms.get(0).toString();
542         continueExec = false;
543     } else {
544
545         // Check if date is MW. If so, then change it to TR.
546         // If TR, then change it to MW.
547         if (schedule.getDays().equals("M,W")) {
548             // System.out.println("Changing date to TR");
549             schedule.setDays("T,R");
550         } else if (schedule.getDays().equals("T,R")) {
551             // System.out.println("Changing date to MW");
552             schedule.setDays("M,W");
553         }
554
555         if (schedule.getDays() == "F") {
556             // If index is at the end
557             // Then stop the loop because
558             // no date is available
559             if (index == startTimes.length - 1) {
560                 continueExec = false;
561             } else {
562                 // Increment the index
563                 index++;
564             }
565             schedule.setStartTime(fridayStartTimes[index]);
566             schedule.setEndTime(fridayEndTimes[index]);
567         } else {
568             if (originalDate.equals(schedule.getDays())) {
569                 // No classrooms available.
570                 // Check if the index is at the end of the array
```

```

570 // Check if the index is at the end of the array
571 // if so, then reset the index to 0
572 // or if the index is the same as the original
index
573 // then there are no available times
574 // So we'll check Friday.
575 if (index - 1 == startTimes.length - 1) {
576     // We hit the end of the times array
577     // Go to start of time slots
578     System.out.println("Hit end of time slots.
Going to beginning");
579     index = 0;
580 } else if (index != originalIndex) {
581     // If current time != starting time
582     // Means we haven't gone through all times
583     System.out.println("WE'RE CHANGING THE
TIME");
584     schedule.setStartTime(startTimes[index]);
585     schedule.setEndTime(endTimes[index]);
586     index++;
587 } else {
588     // Nothing special, just increment the index
589     System.out.println("Hit original index.
Checking Friday");
590     // Check Friday
591     System.out.println("Changing date to
Friday");
592     schedule.setDays("F");
593     index = 0;
594     schedule.setStartTime(fridayStartTimes[index]);
595     schedule.setEndTime(fridayEndTimes[index]);
596 }
597 }
598 }
599 }
600 }
601 return date;
602 } else if (course.getCreditHours() == 3) {
603     // available times are:
604     // 9:00-10:30
605     // 10:00-11:30
606     // 11:00-12:30
607     // 12:00-1:30
608     // 1:00-2:30
609     // 2:00-3:30
610     // 3:00-4:30
611     // An array of the available start times
612     String[] startTimes = { "09:00:00", "10:00:00", "11:00:00",
"12:00:00", "13:00:00", "14:00:00",
"15:00:00" };

```

```

613         "15:00:00" };
614         // An array of the available end times
615         String[] endTimes = { "10:30:00", "11:30:00", "12:30:00",
"13:30:00", "14:30:00", "15:30:00", "16:30:00" };
616
617         // An array of the available Friday start times
618         // 09:00:00 - 13:00:00
619         String[] fridayStartTimes = { "09:00:00", "10:00:00",
"11:00:00", "12:00:00", "13:00:00" };
620         // An array of the available Friday end times
621         String[] fridayEndTimes = { "10:30:00", "11:30:00", "12:30:00",
"13:30:00", "14:30:00" };
622
623         String currentTime = schedule.getStartTime();
624         // Get the index of the currentTime within the startTimes array
625         int originalIndex =
Arrays.asList(startTimes).indexOf(currentTime);
626         int index = originalIndex + 1;
627         boolean continueExec = true;
628         String[] date = new String[4];
629         ArrayList<Schedule> schedules;
630         // Making a copy of the currently selected days
631         // So I can reference later what day I'm switching the
632         // times at
633         String originalDate = schedule.getDays();
634         while (continueExec) {
635
636             // Check the same date for a different room
637             schedules = findSchedule(schedule.getStartTime(),
schedule.getEndTime(), schedule.getDays());
638
639             // We shouldn't have to worry about schedules being empty
640             // as we've already checked for conflicts
641
642             System.out.println("Searching day " + schedule.getDays() + "
at time " + schedule.getStartTime() + " - "
+ schedule.getEndTime());
643
644
645             // Loop through and create a csv
646             // of the used classrooms
647             String usedClassrooms = "";
648             for (Schedule s : schedules) {
649                 // If last index, don't add comma
650                 if (schedules.indexOf(s) == schedules.size() - 1) {
651                     usedClassrooms += s.getClassroomId();
652                 } else {
653                     usedClassrooms += s.getClassroomId() + ",";
654                 }
655                 // System.out.println("Unavailable classroom: " +
s.getClassroomId());

```

```

656     }
657
658     // Query the classroom_table to get all the classrooms
659     // except for the ones that are already being used
660     ArrayList<Integer> availableClassrooms =
findAvailableClassrooms(usedClassrooms);
661
662     // If classrooms available, then set the classroom id
663     // and return the new schedule
664     if (availableClassrooms.size() > 0) {
665         // System.out.println("Available classrooms: " +
availableClassrooms);
666         date[0] = schedule.getStartTime();
667         date[1] = schedule.getEndTime();
668         date[2] = schedule.getDays();
669         date[3] = availableClassrooms.get(0).toString();
670         continueExec = false;
671     } else {
672
673         // Check if date is MW. If so, then change it to TR.
674         // If TR, then change it to MW.
675         if (schedule.getDays().equals("M,W")) {
676             // System.out.println("Changing date to TR");
677             schedule.setDays("T,R");
678         } else if (schedule.getDays().equals("T,R")) {
679             // System.out.println("Changing date to MW");
680             schedule.setDays("M,W");
681         }
682
683         if (schedule.getDays() == "F") {
684             // If index is at the end
685             // Then stop the loop because
686             // no date is available
687             if (index == startTimes.length - 1) {
688                 continueExec = false;
689             } else {
690                 // Increment the index
691                 index++;
692             }
693             schedule.setStartTime(fridayStartTimes[index]);
694             schedule.setEndTime(fridayEndTimes[index]);
695         } else {
696             if (originalDate.equals(schedule.getDays())) {
697                 // No classrooms available.
698                 // Check if the index is at the end of the array
699                 // if so, then reset the index to 0
700                 // or if the index is the same as the original
index
701                 // then there are no available times
702                 // So we'll check Friday.

```

```

703         if (index - 1 == startTimes.length - 1) {
704             // We hit the end of the times array
705             // Go to start of time slots
706             System.out.println("Hit end of time slots.
Going to beginning");
707             index = 0;
708         } else if (index != originalIndex) {
709             // If current time != starting time
710             // Means we haven't gone through all times
711             System.out.println("WE'RE CHANGING THE
TIME");
712             schedule.setStartTime(startTimes[index]);
713             schedule.setEndTime(endTimes[index]);
714             index++;
715         } else {
716             // Nothing special, just increment the index
717             System.out.println("Hit original index.
Checking Friday");
718             // Check Friday
719             System.out.println("Changing date to
Friday");
720             schedule.setDays("F");
721             index = 0;
722             schedule.setStartTime(fridayStartTimes[index]);
723             schedule.setEndTime(fridayEndTimes[index]);
724         }
725     }
726 }
727 }
728 }
729 return date;
730 } else if (course.getCreditHours() == 2) {
731     // available times are any mark between:
732     // 9:00 and 2:00
733     // An array of the available start times
734     String[] startTimes = { "09:00:00", "10:00:00", "11:00:00",
"12:00:00", "13:00:00", "14:00:00" };
735     // An array of the available end times
736     String[] endTimes = { "11:00:00", "12:00:00", "13:00:00",
"14:00:00", "15:00:00", "16:00:00" };
737
738     // An array of the available Friday start times
739     // 09:00:00 - 13:00:00
740     String[] fridayStartTimes = { "09:00:00", "10:00:00",
"11:00:00", "12:00:00", "13:00:00" };
741     // An array of the available Friday end times
742     String[] fridayEndTimes = { "10:30:00", "11:30:00", "12:30:00",
"13:30:00", "14:30:00" };
743

```



```

744 String currentTime = schedule.getStartTime();
745 // Get the index of the currentTime within the startTimes array
746 int originalIndex =
Arrays.asList(startTimes).indexOf(currentTime);
747 int index = originalIndex + 1;
748 boolean continueExec = true;
749 String[] date = new String[4];
750 ArrayList<Schedule> schedules;
751 // Making a copy of the currently selected days
752 // So I can reference later what day I'm switching the
753 // times at
754 String originalDate = schedule.getDays();
755 while (continueExec) {
756
757     // Check the same date for a different room
758     schedules = findSchedule(schedule.getStartTime(),
schedule.getEndTime(), schedule.getDays());
759
760     // We shouldn't have to worry about schedules being empty
761     // as we've already checked for conflicts
762
763     System.out.println("Searching day " + schedule.getDays() + "
at time " + schedule.getStartTime() + " - "
+ schedule.getEndTime());
764
765     // Loop through and create a csv
766     // of the used classrooms
767     String usedClassrooms = "";
768     for (Schedule s : schedules) {
769         // If last index, don't add comma
770         if (schedules.indexOf(s) == schedules.size() - 1) {
771             usedClassrooms += s.getClassroomId();
772         } else {
773             usedClassrooms += s.getClassroomId() + ",";
774         }
775     }
776     // System.out.println("Unavailable classroom: " +
s.getClassroomId());
777 }
778
779 // Query the classroom_table to get all the classrooms
780 // except for the ones that are already being used
781 ArrayList<Integer> availableClassrooms =
findAvailableClassrooms(usedClassrooms);
782
783 // If classrooms available, then set the classroom id
784 // and return the new schedule
785 if (availableClassrooms.size() > 0) {
786     // System.out.println("Available classrooms: " +
availableClassrooms);

```

```

787      date[0] = schedule.getStartTime();
788      date[1] = schedule.getEndTime();
789      date[2] = schedule.getDays();
790      date[3] = availableClassrooms.get(0).toString();
791      continueExec = false;
792  } else {
793
794      // Check if date is MW. If so, then change it to TR.
795      // If TR, then change it to MW.
796      if (schedule.getDays().equals("M")) {
797          // System.out.println("Changing date to TR");
798          schedule.setDays("T");
799      } else if (schedule.getDays().equals("T")) {
800          // System.out.println("Changing date to MW");
801          schedule.setDays("W");
802      } else if (schedule.getDays().equals("W")) {
803          // System.out.println("Changing date to TR");
804          schedule.setDays("R");
805      } else if (schedule.getDays().equals("R")) {
806          // System.out.println("Changing date to MW");
807          schedule.setDays("M");
808      }
809
810      if (schedule.getDays() == "F") {
811          // If index is at the end
812          // Then stop the loop because
813          // no date is available
814          if (index == startTimes.length - 1) {
815              continueExec = false;
816          } else {
817              // Increment the index
818              index++;
819          }
820          schedule.setStartTime(fridayStartTimes[index]);
821          schedule.setEndTime(fridayEndTimes[index]);
822      } else {
823          if (originalDate.equals(schedule.getDays())) {
824              // No classrooms available.
825              // Check if the index is at the end of the array
826              // if so, then reset the index to 0
827              // or if the index is the same as the original
828              // then there are no available times
829              // So we'll check Friday.
830              if (index - 1 == startTimes.length - 1) {
831                  // We hit the end of the times array
832                  // Go to start of time slots
833                  System.out.println("Hit end of time slots.
834                  Going to beginning");
835                  index = 0;

```

```
835         } else if (index != originalIndex) {
836             // If current time != starting time
837             // Means we haven't gone through all times
838             System.out.println("WE'RE CHANGING THE
TIME");
839             schedule.setStartTime(startTimes[index]);
840             schedule.setEndTime(endTimes[index]);
841             index++;
842         } else {
843             // Nothing special, just increment the index
844             System.out.println("Hit original index.
Checking Friday");
845             // Check Friday
846             System.out.println("Changing date to
Friday");
847             schedule.setDays("F");
848             index = 0;
849             schedule.setStartTime(fridayStartTimes[index]);
850             schedule.setEndTime(fridayEndTimes[index]);
851         }
852     }
853 }
854 }
855 }
856 return date;
857 } else {
858     // available times are any mark between:
859     // 9:00 and 3:00
860     // An array of the available start times
861     String[] startTimes = { "09:00:00", "10:00:00", "11:00:00",
"12:00:00", "13:00:00", "14:00:00",
862         "15:00:00" };
863     // An array of the available end times
864     String[] endTimes = { "10:00:00", "11:00:00", "12:00:00",
"13:00:00", "14:00:00", "15:00:00", "16:00:00" };
865
866     // An array of the available Friday start times
867     // 09:00:00 - 16:00:00
868     String[] fridayStartTimes = { "09:00:00", "10:00:00",
"11:00:00", "12:00:00", "13:00:00", "14:00:00",
869         "15:00:00", "16:00:00" };
870     // An array of the available Friday end times
871     String[] fridayEndTimes = { "10:00:00", "11:00:00", "12:00:00",
"13:00:00", "14:00:00", "15:00:00",
872         "16:00:00", "17:00:00" };
873
874     String currentTime = schedule.getStartTime();
875     // Get the index of the currentTime within the startTimes array
876     int originalIndex =
```

```

877     Arrays.asList(startTimes).indexOf(currentTime);
878     int index = originalIndex + 1;
879     boolean continueExec = true;
880     String[] date = new String[4];
881     ArrayList<Schedule> schedules;
882     // Making a copy of the currently selected days
883     // So I can reference later what day I'm switching the
884     // times at
885     String originalDate = schedule.getDays();
886     while (continueExec) {
887         // Check the same date for a different room
888         schedules = findSchedule(schedule.getStartTime(),
889         schedule.getEndTime(), schedule.getDays());
890
891         // We shouldn't have to worry about schedules being empty
892         // as we've already checked for conflicts
893
894         System.out.println("Searching day " + schedule.getDays() + "
895         at time " + schedule.getStartTime() + " - "
896         + schedule.getEndTime());
897
898         // Loop through and create a csv
899         // of the used classrooms
900         String usedClassrooms = "";
901         for (Schedule s : schedules) {
902             // If last index, don't add comma
903             if (schedules.indexOf(s) == schedules.size() - 1) {
904                 usedClassrooms += s.getClassroomId();
905             } else {
906                 usedClassrooms += s.getClassroomId() + ",";
907             }
908             // System.out.println("Unavailable classroom: " +
909             s.getClassroomId());
910         }
911
912         // Query the classroom_table to get all the classrooms
913         // except for the ones that are already being used
914         ArrayList<Integer> availableClassrooms =
915         findAvailableClassrooms(usedClassrooms);
916
917         // If classrooms available, then set the classroom id
918         // and return the new schedule
919         if (availableClassrooms.size() > 0) {
920             // System.out.println("Available classrooms: " +
921             availableClassrooms);
922             date[0] = schedule.getStartTime();
923             date[1] = schedule.getEndTime();
924             date[2] = schedule.getDays();
925             date[3] = availableClassrooms.get(0).toString();

```

```

920         date[5] = availableClassrooms.get(0).toString(),
921         continueExec = false;
922     } else {
923
924         // Check if date if MW. If so, then change it to TR.
925         // If TR, then change it to MW.
926         if (schedule.getDays().equals("M")) {
927             // System.out.println("Changing date to TR");
928             schedule.setDays("T");
929         } else if (schedule.getDays().equals("T")) {
930             // System.out.println("Changing date to MW");
931             schedule.setDays("W");
932         } else if (schedule.getDays().equals("W")) {
933             // System.out.println("Changing date to TR");
934             schedule.setDays("R");
935         } else if (schedule.getDays().equals("R")) {
936             // System.out.println("Changing date to MW");
937             schedule.setDays("M");
938         }
939
940         if (schedule.getDays() == "F") {
941             // If index is at the end
942             // Then stop the loop because
943             // no date is available
944             if (index == startTimes.length - 1) {
945                 continueExec = false;
946             } else {
947                 // Increment the index
948                 index++;
949             }
950             schedule.setStartTime(fridayStartTimes[index]);
951             schedule.setEndTime(fridayEndTimes[index]);
952         } else {
953             if (originalDate.equals(schedule.getDays())) {
954                 // No classrooms available.
955                 // Check if the index is at the end of the array
956                 // if so, then reset the index to 0
957                 // or if the index is the same as the original
958                 // then there are no available times
959                 // So we'll check Friday.
960                 if (index - 1 == startTimes.length - 1) {
961                     // We hit the end of the times array
962                     // Go to start of time slots
963                     System.out.println("Hit end of time slots.
964                     Going to beginning");
965                     index = 0;
966                 } else if (index != originalIndex) {
967                     // If current time != starting time
968                     // Means we haven't gone through all times

```

```

968         System.out.println("WE'RE CHANGING THE
TIME");
969         schedule.setStartTime(startTimes[index]);
970         schedule.setEndTime(endTimes[index]);
971         index++;
972     } else {
973         // Nothing special, just increment the index
974         System.out.println("Hit original index.
Checking Friday");
975         // Check Friday
976         System.out.println("Changing date to
Friday");
977         schedule.setDays("F");
978         index = 0;
979
980         schedule.setStartTime(fridayStartTimes[index]);
981         schedule.setEndTime(fridayEndTimes[index]);
982     }
983 }
984 }
985 }
986 return date;
987 }
988 }
989
990 private ArrayList<Integer> findAvailableClassrooms(String
usedClassrooms) {
991     // I was going to make this DB driven but decided not to
992     // as the assignment description
993     // only mentioned the 3 rooms.
994
995     // Available classrooms are A, B, and C with their indexes
996     // being 1,2,3 respectively
997
998     // Create an array of the available classrooms
999     // and remove the ones that are already being used
1000     ArrayList<Integer> availableClassrooms = new ArrayList<Integer>();
1001     availableClassrooms.add(1);
1002     availableClassrooms.add(2);
1003     availableClassrooms.add(3);
1004
1005     System.out.println("Unavailable classrooms: " + usedClassrooms);
1006
1007     // If there are no used classrooms, then return the available
1008     // classrooms
1009     if (usedClassrooms.equals("")) {
1010         return availableClassrooms;
1011     }
1012

```

```
1013 // The usedClassrooms is a csv of the used classrooms
1014 // we need to remove these from the available classrooms
1015 String[] usedClassroomsArray = usedClassrooms.split(",");
1016 if (usedClassroomsArray.length == 3) {
1017     // All are gone
1018     availableClassrooms.clear();
1019 } else {
1020     for (String s : usedClassroomsArray) {
1021         // Remove the used classroom from the available classrooms
1022         // System.out.println("Removing classroom: " + s);
1023         try {
1024             availableClassrooms.remove(availableClassrooms.indexOf(Integer.parseInt(s))
1025 );
1026         } catch (Exception e) {
1027             // System.out.println("Error removing classroom: " + s);
1028             // This means that we are completely out of room
1029         }
1030     }
1031 }
1032 System.out.println("Available classrooms: " + availableClassrooms);
1033 return availableClassrooms;
1034 }
1035
1036 public ArrayList<Schedule> getSchedule() {
1037     ArrayList<Schedule> schedule = new ArrayList<Schedule>();
1038
1039     // Get the entire schedule from the database while joining the
1040     // classroom_table
1041     // and professor_table and courses_table
1042     String sql = "SELECT * FROM schedule_table JOIN classroom_table ON
1043     schedule_table.classroom_tuid = classroom_table.tuid JOIN professors_table
1044     ON schedule_table.professor_tuid = professors_table.tuid JOIN courses_table
1045     ON schedule_table.course_tuid = courses_table.tuid;";
1046
1047     // Execute the query and loop through the results and
1048     // output the console the results
1049     try (Connection conn = this.connect();
1050         Statement stmt = conn.createStatement()) {
1051         stmt.execute(sql);
1052
1053         ResultSet rs = stmt.getResultSet();
1054
1055         // loop through the result set
1056         while (rs.next()) {
1057             // System.out.println(rs.getString("tuid"));
1058             schedule.add(
1059                 new Schedule(rs.getInt("course_tuid"),
1060 rs.getInt("classroom_tuid"), rs.getInt("professor_tuid"))
1061             );
1062         }
1063     }
1064 }
```

```

1056         rs.getInt("section"),
rs.getString("start_time"), rs.getString("end_time"),
1057         rs.getString("days"),
rs.getString("course_title"), rs.getString("professor_name"),
1058         rs.getString("classroom_name"))));
1059     }
1060
1061     } catch (SQLException e) {
1062         System.out.println(e.getMessage());
1063     }
1064
1065     return schedule;
1066 }
1067
1068 public ArrayList<Professor> getProfessors() {
1069     // Return all professors
1070     ArrayList<Professor> professors = new ArrayList<Professor>();
1071
1072     // Get all the professors from the DB
1073     String sql = "SELECT * FROM professors_table";
1074
1075     try (Connection conn = this.connect();
1076         Statement stmt = conn.createStatement()) {
1077         stmt.execute(sql);
1078
1079         ResultSet rs = stmt.getResultSet();
1080
1081         // loop through the result set
1082         while (rs.next()) {
1083             // System.out.println(rs.getString("tuid"));
1084             professors.add(new Professor(rs.getInt("tuid"),
rs.getString("professor_name")));
1085         }
1086
1087     } catch (SQLException e) {
1088         System.out.println(e.getMessage());
1089     }
1090
1091     return professors;
1092 }
1093
1094 public ArrayList<Schedule> getFacultySchedule() {
1095     ArrayList<Schedule> schedule = new ArrayList<Schedule>();
1096
1097     // Get the entire schedule from the database while joining the
classroom_table
1098     // and professor_table and courses_table
1099     String sql = "SELECT * FROM professors_table INNER JOIN
schedule_table ON professors_table.tuid = schedule_table.professor_tuid LEFT
JOIN courses_table ON schedule_table.course_tuid = courses_table.tuid ORDER

```



```

1100 JOIN courses_table ON schedule_table.course_tuid = courses_table.tuid ORDER
1101 BY professors_table.tuid;";
1102
1103 // Execute the query and loop through the results and
1104 // output the console the results
1105 try (Connection conn = this.connect();
1106      Statement stmt = conn.createStatement()) {
1107     stmt.execute(sql);
1108
1109     ResultSet rs = stmt.getResultSet();
1110
1111     // loop through the result set
1112     while (rs.next()) {
1113         // System.out.println(rs.getString("tuid"));
1114         schedule.add(
1115             new Schedule(rs.getInt("course_tuid"),
1116 rs.getInt("classroom_tuid"), rs.getInt("professor_tuid"),
1117 rs.getInt("section"),
1118 rs.getString("start_time"), rs.getString("end_time"),
1119 rs.getString("days"),
1120 rs.getString("course_title"), rs.getString("professor_name"),
1121 rs.getString("classroom_name")));
1122     }
1123 } catch (SQLException e) {
1124     System.out.println(e.getMessage());
1125 }
1126
1127 return schedule;
1128 }
1129
1130 public ArrayList<Schedule> getScheduleByProfessor(Integer id) {
1131     // Get the schedule for a specific professor
1132     ArrayList<Schedule> schedule = new ArrayList<Schedule>();
1133
1134     // Get the entire schedule from the database while joining the
1135     classroom_table
1136     // and professor_table and courses_table
1137     String sql = "SELECT * FROM professors_table INNER JOIN
1138 schedule_table ON professors_table.tuid = schedule_table.professor_tuid LEFT
1139 JOIN courses_table ON schedule_table.course_tuid = courses_table.tuid LEFT
1140 JOIN classroom_table ON schedule_table.classroom_tuid = classroom_table.tuid
1141 WHERE professors_table.tuid = "
1142 + id + " ORDER BY professors_table.tuid;";
1143
1144 // Execute the query and loop through the results and
1145 // output the console the results
1146 try (Connection conn = this.connect();
1147      Statement stmt = conn.createStatement()) {
1148     stmt.execute(sql);
1149
1150

```

```
1140
1141     ResultSet rs = stmt.getResultSet();
1142
1143     // loop through the result set
1144     while (rs.next()) {
1145         // System.out.println(rs.getString("tuid"));
1146         Schedule classes = new Schedule(rs.getInt("course_tuid"),
rs.getInt("classroom_tuid"),
1147         rs.getInt("professor_tuid"),
1148         rs.getInt("section"), rs.getString("start_time"),
rs.getString("end_time"),
1149         rs.getString("days"), rs.getString("course_title"),
rs.getString("professor_name"),
rs.getString("classroom_name"));
1150         classes.setCredits((rs.getInt("credit_hours")));
1151         schedule.add(classes);
1152     }
1153
1154     }
1155
1156     } catch (SQLException e) {
1157         System.out.println(e.getMessage());
1158     }
1159
1160     return schedule;
1161
1162 }
1163
1164 public ArrayList<Course> getCourses() {
1165     // Get all the courses in the DB and return them
1166     ArrayList<Course> courses = new ArrayList<Course>();
1167
1168     // Get all the professors from the DB
1169     String sql = "SELECT * FROM courses_table";
1170
1171     try (Connection conn = this.connect();
1172         Statement stmt = conn.createStatement()) {
1173         stmt.execute(sql);
1174
1175         ResultSet rs = stmt.getResultSet();
1176
1177         // loop through the result set
1178         while (rs.next()) {
1179             // System.out.println(rs.getString("tuid"));
1180             courses.add(new Course(rs.getInt("tuid"),
rs.getString("course_id"), rs.getString("course_title"),
rs.getInt("credit_hours")));
1181         }
1182     }
1183
1184     } catch (SQLException e) {
1185         System.out.println(e.getMessage());
```

```
1186     }
1187     return courses;
1188 }
1189
1190 public ArrayList<Schedule> getScheduleByCourse(Integer courseId) {
1191     // Get the schedule for all courses with the same course id
1192     // Joining the classroom_table and courses_table
1193     ArrayList<Schedule> schedule = new ArrayList<Schedule>();
1194
1195     // Get the entire schedule from the database while joining the
1196     // classroom_table
1197     // and professor_table and courses_table
1198     String sql = "SELECT * FROM courses_table INNER JOIN schedule_table
1199 ON courses_table.tuid = schedule_table.course_tuid LEFT JOIN
1200 professors_table ON schedule_table.professor_tuid = professors_table.tuid
1201 LEFT JOIN classroom_table ON schedule_table.classroom_tuid =
1202 classroom_table.tuid WHERE courses_table.tuid = "
1203 + courseId + " ORDER BY courses_table.tuid;";
1204
1205     // Execute the query and loop through the results and
1206     // output the console the results
1207     try (Connection conn = this.connect();
1208         Statement stmt = conn.createStatement()) {
1209         stmt.execute(sql);
1210
1211         ResultSet rs = stmt.getResultSet();
1212
1213         // loop through the result set
1214         while (rs.next()) {
1215             // System.out.println(rs.getString("tuid"));
1216             Schedule course = new Schedule(rs.getInt("course_tuid"),
1217 rs.getInt("classroom_tuid"),
1218 rs.getInt("professor_tuid"),
1219 rs.getInt("section"), rs.getString("start_time"),
1220 rs.getString("end_time"),
1221 rs.getString("days"), rs.getString("course_title"),
1222 rs.getString("professor_name"),
1223 rs.getString("classroom_name"));
1224             course.setCapacity((rs.getInt("capacity")));
1225             schedule.add(course);
1226         }
1227     } catch (SQLException e) {
1228         System.out.println(e.getMessage());
1229     }
1230
1231     return schedule;
1232 }
1233 }
```