```java
import java.sql.Array;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Arrays;

public class Database {
    private Connection connect() {
        // SQLite connection string
        String url = "jdbc:sqlite:db.sqlite";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }
        return conn;
    }

    public void migrate() {
        String sql = "CREATE TABLE IF NOT EXISTS schedule_table (\n"
                + " tuid integer PRIMARY KEY,\n"
                + " course_tuid text NOT NULL,\n"
                + " classroom_tuid text NOT NULL,\n"
                + " professor_tuid text NOT NULL,\n"
                + " section integer NOT NULL,\n"
                + " start_time text NOT NULL,\n"
                + " end_time text NOT NULL,\n"
                + " days text NOT NULL);";

        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            // create a new table
            // System.out.println("Creating schedule table...");
            stmt.execute(sql);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

        sql = "CREATE TABLE IF NOT EXISTS professors_table (\n"
                + " tuid integer PRIMARY KEY,\n"
                + " professor_name text NOT NULL);";

        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            // create a new table
```

```
            // create a new table
            // System.out.println("Creating professors table...");
            stmt.execute(sql);
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }

        sql = "CREATE TABLE IF NOT EXISTS classroom_table (\n"
                + " tuid integer PRIMARY KEY,\n"
                + " classroom_name text NOT NULL,\n"
                + " capacity integer NOT NULL);";

        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            // create a new table
            // System.out.println("Creating classroom table...");
            stmt.execute(sql);
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }

        sql = "CREATE TABLE IF NOT EXISTS courses_table (\n"
                + " tuid integer PRIMARY KEY,\n"
                + " course_id text NOT NULL,\n"
                + " course_title text NOT NULL,\n"
                + " credit_hours integer NOT NULL);";

        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            // create a new table
            // System.out.println("Creating courses table...");
            stmt.execute(sql);
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }
    }

    public void drop() {
        String sql = "DROP TABLE IF EXISTS schedule_table;";

        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            // create a new table
            // System.out.println("Dropping schedule table...");
            stmt.execute(sql);
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }

        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,
    capacity)\n"
136                 + " VALUES('A', 30),\n"
137                 + " ('B', 25),\n"
138                 + " ('C', 20);";
139
140         try (Connection conn = this.connect();
141                 Statement stmt = conn.createStatement()) {
142             // create a new table
143             // System.out.println("Seeding classroom table...");
144             stmt.execute(sql);
145         } catch (SQLException e) {
146             // System.out.println(e.getMessage());
147         }
148
```

```
        // Seed courses
        sql = "INSERT INTO courses_table(course_id, course_title, credit_hours)\n"
                + " VALUES('CS 101', 'Intro to Computer Science', 3),\n"
                + " ('CS 105', 'Computers and Programming', 4),\n"
                + " ('CSC 105', 'Computers and Programming', 4), \n"
                + " ('CSC 107', 'Introduction to Code Preparation', 1), \n"
                + " ('CSC 116', 'Programming I', 4), \n"
                + " ('CSC 216', 'Programming II', 4), \n"
                + " ('CSC 227', 'Commenting and Naming Conventions', 2), \n"
                + " ('CSC 316', 'Data Structures \u0026 Algorithms', 4), \n"
                + " ('CSC 416', 'Advanced Algorithm Analysis', 3), \n"
                + " ('CSC 211', 'Introductory .NET Development', 3), \n"
                + " ('CSC 311', 'Advanced .NET Development', 4), \n"
                + " ('CSC 313', 'Real World Application Development', 3), \n"
                + " ('CSC 411', 'Data Driven Systems', 3), \n"
                + " ('CSC 412', 'Sensor Systems', 3), \n"
                + " ('CSC 413', 'Artificial Intelligence Systems', 3), \n"
                + " ('CSC 496', 'Software Engineering I', 4), \n"
                + " ('CSC 497', 'Software Engineering II', 4), \n"
                + " ('CSC 498', 'Software Engineering III', 4);";

        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            // create a new table
            // System.out.println("Seeding courses table...");
            stmt.execute(sql);
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }

        // Seed professors
        sql = "INSERT INTO professors_table(professor_name)\n"
                + " VALUES('James'),\n"
                + " ('Smith'),\n"
                + " ('Jones'),\n"
                + " ('Vasquez'),\n"
                + " ('Abdul'),\n"
                + " ('Thomas');";

        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            // create a new table
            // System.out.println("Seeding professors table...");
            stmt.execute(sql);
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }
    }
```

```
    }

    public void close() {
        // End connection to database
        Connection conn = this.connect();
        try {
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException ex) {
            // System.out.println(ex.getMessage());
        }
    }

    public void insertScheduleFile(ScheduleFile sf) {
        // Print the schedule file
        // System.out.println(sf.toString());
        // System.out.println(sf.getCourseName());

        // Initialize variables for inserting into database
        Course course = findCourse(sf.getCourseName());
        Professor professor = findProfessor(sf.getProfessorName());

        // Print the scheduleFile
        // System.out.println(sf.toString());

        // Print the course
        // System.out.println(course.toString());

        // Print the course
        // System.out.println(professor.toString());

        // Okay... So now we have the professor, course, and
        // schedule file information
        // Now we query the schedule_table to see if any courses
        // are already scheduled for the same time.

        // Create a new schedule object using the course, professor,
        // and schedule file information
        Schedule schedule = new Schedule(course, professor, sf);

        // Create an arraylist of schedules
        ArrayList<Schedule> schedules = new ArrayList<Schedule>();

        schedules = findSchedule(sf.getStartTime(), sf.getEndTime(),
sf.getDays());

        // Find other instances of the course to determine the
        // section number
        int section = findSection(course.getId());
```

```
        schedule.setSection(section);

        // Check the size of the schedules arraylist
        // If it's 0, then we can insert the schedule
        // If it's not 0, then we need to find an alternate time
        if (schedules.size() == 0) {
            System.out.println("No conflicts found. Inserting schedule...");

            // Then we should be good to schedule what ever
            // classroom is available.
            // So we'll query the classroom_table to find
            // A and assign the tuid to the schedule
            schedule.setClassroomId(findClassroomTuid("A"));

            // Insert the schedule
            System.out.println("Inserting schedule... NO CONFLICTS");
            insertSchedule(schedule);
        } else {
            System.out.println("There is a conflict with the schedule");
            // And here's where the real fun begins
            // Everything has been all fun and games till now.
            String[] alternateTime = findAlternateTime(schedule, course);
            if (alternateTime[0] == null) {
                System.out.println("No alternate time found");
            } else {
                // Update the schedule object with
                // the newly found data
                schedule.setStartTime(alternateTime[0]);
                schedule.setEndTime(alternateTime[1]);
                schedule.setDays(alternateTime[2]);
                schedule.setClassroomId(Integer.parseInt(alternateTime[3]));

                System.out.println("Class rescheduled to " +
schedule.getDays() + " from " + schedule.getStartTime()
                        + " to " + schedule.getEndTime() + " in classroom "
+ schedule.getClassroomId());

                insertSchedule(schedule);
            }
        }
    }

    private Course findCourse(String name) {
        // Find the course from the courses_table
        // by using the course_title and return the entire row

        String sql = "SELECT * FROM courses_table WHERE course_id = ?;";

        try (Connection conn = this.connect();
```

```
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            // set the value
            stmt.setString(1, name);

            ResultSet rs = stmt.executeQuery();

            // Check if the course was found
            if (rs.next()) {
                // Create a new course object
                Course course = new Course(rs.getInt(1), rs.getString(2),
rs.getString(3), rs.getInt(4));

                return course;
            } else {
                System.out.println("Course not found");
                return null;
            }
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }
        return null;
    }

    private Professor findProfessor(String name) {
        // Find the professor from the professors_table
        // by using the professor_name and return the entire row

        String sql = "SELECT * FROM professors_table WHERE professor_name =
?;";

        try (Connection conn = this.connect();
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            // set the value
            stmt.setString(1, name);

            ResultSet rs = stmt.executeQuery();

            // Check if the professor was found
            if (rs.next()) {
                // Create a new professor object
                Professor professor = new Professor(rs.getInt(1),
rs.getString(2));

                return professor;
            } else {
                System.out.println("Professor not found");
                return null;
            }
```

```
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }
        return null;
    }

    private ArrayList<Schedule> findSchedule(String startTime, String endTime, String days) {

        // Query Summed Up:
        // Select everything from the schedule where
        // the start time < the GIVEN end time
        // and the end time > the GIVEN start time
        // and the days are the same
        // AND THEN join the classroom table to actually
        // get the classroom name
        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 (?);";
        ArrayList<Schedule> schedules = new ArrayList<Schedule>();

        try (Connection conn = this.connect();
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            // set the value
            stmt.setString(1, endTime);
            stmt.setString(2, startTime);
            stmt.setString(3, days);

            ResultSet rs = stmt.executeQuery();

            System.out.println("Searching for schedule conflicts...");

            // Check if the professor was found
            while (rs.next()) {
                // Add the results to the schedules arraylist
                schedules.add(new Schedule(rs.getInt("tuid"), rs.getInt("course_tuid"), rs.getInt("classroom_tuid"),
                        rs.getInt("professor_tuid"), rs.getInt("section"), rs.getString("start_time"),
                        rs.getString("end_time"), rs.getString("days")));
            }

            return schedules;
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }

        return null;
    }
```

```
    }

    private void insertSchedule(Schedule schedule) {
        // Insert the schedule into the schedule_table

        String sql = "INSERT INTO schedule_table(course_tuid,
classroom_tuid, professor_tuid, section, start_time, end_time, days)
VALUES(?, ?, ?, ?, ?, ?, ?);";

        try (Connection conn = this.connect();
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            // Set the values
            stmt.setInt(1, schedule.getCourseId());
            stmt.setInt(2, schedule.getClassroomId());
            stmt.setInt(3, schedule.getProfessorId());
            stmt.setInt(4, schedule.getSection());
            stmt.setString(5, schedule.getStartTime());
            stmt.setString(6, schedule.getEndTime());
            stmt.setString(7, schedule.getDays());

            // Execute the query
            stmt.executeUpdate();

        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }

    }

    private int findSection(int courseTuid) {
        // Find the section number for the course
        // by using the course_tuid and return the section number

        String sql = "SELECT * FROM schedule_table WHERE course_tuid = ?;";

        try (Connection conn = this.connect();
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            // set the value
            stmt.setInt(1, courseTuid);

            ResultSet rs = stmt.executeQuery();

            // Get the total count of the results
            int count = 1;

            // Loop through all the results and
            // increment the count
            while (rs.next()) {
                count++;
```

```
                count++;
            }

            // System.out.println("TOTAL CURRENT ENTRIES: " + count);

            return count;
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }
        return -1;
    }

    // Return the classroom tuid from the classroom name
    private int findClassroomTuid(String classroomName) {
        // Find the classroom from the classroom_table
        // by using the classroom_name and return the tuid

        String sql = "SELECT * FROM classroom_table WHERE classroom_name =
?;";

        try (Connection conn = this.connect();
                PreparedStatement stmt = conn.prepareStatement(sql)) {

            // set the value
            stmt.setString(1, classroomName);

            ResultSet rs = stmt.executeQuery();

            // Check if the professor was found
            if (rs.next()) {
                // Create a new professor object
                return rs.getInt(1);
            } else {
                System.out.println("Classroom not found");
                return -1;
            }
        } catch (SQLException e) {
            // System.out.println(e.getMessage());
        }
        return -1;
    }

    private String[] findAlternateTime(Schedule schedule, Course course) {
        // Okay so first we're going to check the credit hours
        // to determine what the available days are.

        // If credit hours are 3 or 4, then available days are
        // MW, TR, or F
        // If credit hours are 1 or 2, then available days are
        // M,T,W,R,F
```

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

```
                    } else {
                        usedClassrooms += s.getClassroomId() + ",";
                    }
                    // System.out.println("Unavailable classroom: " +
s.getClassroomId());
                }

                // Query the classroom_table to get all the classrooms
                // except for the ones that are already being used
                ArrayList<Integer> availableClassrooms =
findAvailableClassrooms(usedClassrooms);

                // If classrooms available, then set the classroom id
                // and return the new schedule
                if (availableClassrooms.size() > 0) {
                    // System.out.println("Available classrooms: " +
availableClassrooms);
                    date[0] = schedule.getStartTime();
                    date[1] = schedule.getEndTime();
                    date[2] = schedule.getDays();
                    date[3] = availableClassrooms.get(0).toString();
                    continueExec = false;
                } else {

                    // Check if date if MW. If so, then change it to TR.
                    // If TR, then change it to MW.
                    if (schedule.getDays().equals("M,W")) {
                        // System.out.println("Changing date to TR");
                        schedule.setDays("T,R");
                    } else if (schedule.getDays().equals("T,R")) {
                        // System.out.println("Changing date to MW");
                        schedule.setDays("M,W");
                    }

                    if (schedule.getDays() == "F") {
                        // If index is at the end
                        // Then stop the loop because
                        // no date is available
                        if (index == startTimes.length - 1) {
                            continueExec = false;
                        } else {
                            // Increment the index
                            index++;
                        }
                        schedule.setStartTime(fridayStartTimes[index]);
                        schedule.setEndTime(fridayEndTimes[index]);
                    } else {
                        if (originalDate.equals(schedule.getDays())) {
                            // No classrooms available.
                            // Check if the index is at the end of the array
```

```
                            // check if the index is at the end of the array
                            // if so, then reset the index to 0
                            // or if the index is the same as the original
index
                            // then there are no available times
                            // So we'll check Friday.
                            if (index - 1 == startTimes.length - 1) {
                                // We hit the end of the times array
                                // Go to start of time slots
                                System.out.println("Hit end of time slots.
Going to beginning");
                                index = 0;
                            } else if (index != originalIndex) {
                                // If current time !== starting time
                                // Means we haven't gone through all times
                                System.out.println("WE'RE CHANGING THE
TIME");
                                schedule.setStartTime(startTimes[index]);
                                schedule.setEndTime(endTimes[index]);
                                index++;
                            } else {
                                // Nothing special, just increment the index
                                System.out.println("Hit original index.
Checking Friday");
                                // Check Friday
                                System.out.println("Changing date to
Friday");
                                schedule.setDays("F");
                                index = 0;

 schedule.setStartTime(fridayStartTimes[index]);
                                schedule.setEndTime(fridayEndTimes[index]);
                            }
                        }
                    }
                }
            }
            return date;
        } else if (course.getCreditHours() == 3) {
            // available times are:
            // 9:00-10:30
            // 10:00-11:30
            // 11:00-12:30
            // 12:00-1:30
            // 1:00-2:30
            // 2:00-3:30
            // 3:00-4:30
            // An array of the available start times
            String[] startTimes = { "09:00:00", "10:00:00", "11:00:00",
"12:00:00", "13:00:00", "14:00:00",
                    "15:00:00" };
```

```
                    "15:00:00" };
            // An array of the available end times
            String[] endTimes = { "10:30:00", "11:30:00", "12:30:00",
"13:30:00", "14:30:00", "15:30:00", "16:30:00" };

            // An array of the available Friday start times
            // 09:00:00 - 13:00:00
            String[] fridayStartTimes = { "09:00:00", "10:00:00",
"11:00:00", "12:00:00", "13:00:00" };
            // An array of the available Friday end times
            String[] fridayEndTimes = { "10:30:00", "11:30:00", "12:30:00",
"13:30:00", "14:30:00" };

            String currentTime = schedule.getStartTime();
            // Get the index of the currentTime within the startTimes array
            int originalIndex =
Arrays.asList(startTimes).indexOf(currentTime);
            int index = originalIndex + 1;
            boolean continueExec = true;
            String[] date = new String[4];
            ArrayList<Schedule> schedules;
            // Making a copy of the currently selected days
            // So I can reference later what day I'm switching the
            // times at
            String originalDate = schedule.getDays();
            while (continueExec) {

                // Check the same date for a different room
                schedules = findSchedule(schedule.getStartTime(),
schedule.getEndTime(), schedule.getDays());

                // We shouldn't have to worry about schedules being empty
                // as we've already checked for conflicts

                System.out.println("Searching day " + schedule.getDays() + "
at time " + schedule.getStartTime() + " - "
                        + schedule.getEndTime());

                // Loop through and create a csv
                // of the used classrooms
                String usedClassrooms = "";
                for (Schedule s : schedules) {
                    // If last index, don't add comma
                    if (schedules.indexOf(s) == schedules.size() - 1) {
                        usedClassrooms += s.getClassroomId();
                    } else {
                        usedClassrooms += s.getClassroomId() + ",";
                    }
                    // System.out.println("Unavailable classroom: " +
s.getClassroomId());
```

```
                }

                // Query the classroom_table to get all the classrooms
                // except for the ones that are already being used
                ArrayList<Integer> availableClassrooms =
findAvailableClassrooms(usedClassrooms);

                // If classrooms available, then set the classroom id
                // and return the new schedule
                if (availableClassrooms.size() > 0) {
                    // System.out.println("Available classrooms: " +
availableClassrooms);
                    date[0] = schedule.getStartTime();
                    date[1] = schedule.getEndTime();
                    date[2] = schedule.getDays();
                    date[3] = availableClassrooms.get(0).toString();
                    continueExec = false;
                } else {

                    // Check if date if MW. If so, then change it to TR.
                    // If TR, then change it to MW.
                    if (schedule.getDays().equals("M,W")) {
                        // System.out.println("Changing date to TR");
                        schedule.setDays("T,R");
                    } else if (schedule.getDays().equals("T,R")) {
                        // System.out.println("Changing date to MW");
                        schedule.setDays("M,W");
                    }

                    if (schedule.getDays() == "F") {
                        // If index is at the end
                        // Then stop the loop because
                        // no date is available
                        if (index == startTimes.length - 1) {
                            continueExec = false;
                        } else {
                            // Increment the index
                            index++;
                        }
                        schedule.setStartTime(fridayStartTimes[index]);
                        schedule.setEndTime(fridayEndTimes[index]);
                    } else {
                        if (originalDate.equals(schedule.getDays())) {
                            // No classrooms available.
                            // Check if the index is at the end of the array
                            // if so, then reset the index to 0
                            // or if the index is the same as the original
index
                            // then there are no available times
                            // So we'll check Fridav.
```

```
                            if (index - 1 == startTimes.length - 1) {
                                // We hit the end of the times array
                                // Go to start of time slots
                                System.out.println("Hit end of time slots.
Going to beginning");
                                index = 0;
                            } else if (index != originalIndex) {
                                // If current time !== starting time
                                // Means we haven't gone through all times
                                System.out.println("WE'RE CHANGING THE
TIME");
                                schedule.setStartTime(startTimes[index]);
                                schedule.setEndTime(endTimes[index]);
                                index++;
                            } else {
                                // Nothing special, just increment the index
                                System.out.println("Hit original index.
Checking Friday");
                                // Check Friday
                                System.out.println("Changing date to
Friday");
                                schedule.setDays("F");
                                index = 0;

 schedule.setStartTime(fridayStartTimes[index]);
                                schedule.setEndTime(fridayEndTimes[index]);
                            }
                        }
                    }
                }
            }
            return date;
        } else if (course.getCreditHours() == 2) {
            // available times are any mark between:
            // 9:00 and 2:00
            // An array of the available start times
            String[] startTimes = { "09:00:00", "10:00:00", "11:00:00",
"12:00:00", "13:00:00", "14:00:00" };
            // An array of the available end times
            String[] endTimes = { "11:00:00", "12:00:00", "13:00:00",
"14:00:00", "15:00:00", "16:00:00" };

            // An array of the available Friday start times
            // 09:00:00 - 13:00:00
            String[] fridayStartTimes = { "09:00:00", "10:00:00",
"11:00:00", "12:00:00", "13:00:00" };
            // An array of the available Friday end times
            String[] fridayEndTimes = { "10:30:00", "11:30:00", "12:30:00",
"13:30:00", "14:30:00" };
```

```
            String currentTime = schedule.getStartTime();
            // Get the index of the currentTime within the startTimes array
            int originalIndex =
Arrays.asList(startTimes).indexOf(currentTime);
            int index = originalIndex + 1;
            boolean continueExec = true;
            String[] date = new String[4];
            ArrayList<Schedule> schedules;
            // Making a copy of the currently selected days
            // So I can reference later what day I'm switching the
            // times at
            String originalDate = schedule.getDays();
            while (continueExec) {

                // Check the same date for a different room
                schedules = findSchedule(schedule.getStartTime(),
schedule.getEndTime(), schedule.getDays());

                // We shouldn't have to worry about schedules being empty
                // as we've already checked for conflicts

                System.out.println("Searching day " + schedule.getDays() + "
at time " + schedule.getStartTime() + " - "
                        + schedule.getEndTime());

                // Loop through and create a csv
                // of the used classrooms
                String usedClassrooms = "";
                for (Schedule s : schedules) {
                    // If last index, don't add comma
                    if (schedules.indexOf(s) == schedules.size() - 1) {
                        usedClassrooms += s.getClassroomId();
                    } else {
                        usedClassrooms += s.getClassroomId() + ",";
                    }
                    // System.out.println("Unavailable classroom: " +
s.getClassroomId());
                }

                // Query the classroom_table to get all the classrooms
                // except for the ones that are already being used
                ArrayList<Integer> availableClassrooms =
findAvailableClassrooms(usedClassrooms);

                // If classrooms available, then set the classroom id
                // and return the new schedule
                if (availableClassrooms.size() > 0) {
                    // System.out.println("Available classrooms: " +
availableClassrooms);
```

```
                    date[0] = schedule.getStartTime();
                    date[1] = schedule.getEndTime();
                    date[2] = schedule.getDays();
                    date[3] = availableClassrooms.get(0).toString();
                    continueExec = false;
                } else {

                    // Check if date if MW. If so, then change it to TR.
                    // If TR, then change it to MW.
                    if (schedule.getDays().equals("M")) {
                        // System.out.println("Changing date to TR");
                        schedule.setDays("T");
                    } else if (schedule.getDays().equals("T")) {
                        // System.out.println("Changing date to MW");
                        schedule.setDays("W");
                    } else if (schedule.getDays().equals("W")) {
                        // System.out.println("Changing date to TR");
                        schedule.setDays("R");
                    } else if (schedule.getDays().equals("R")) {
                        // System.out.println("Changing date to MW");
                        schedule.setDays("M");
                    }

                    if (schedule.getDays() == "F") {
                        // If index is at the end
                        // Then stop the loop because
                        // no date is available
                        if (index == startTimes.length - 1) {
                            continueExec = false;
                        } else {
                            // Increment the index
                            index++;
                        }
                        schedule.setStartTime(fridayStartTimes[index]);
                        schedule.setEndTime(fridayEndTimes[index]);
                    } else {
                        if (originalDate.equals(schedule.getDays())) {
                            // No classrooms available.
                            // Check if the index is at the end of the array
                            // if so, then reset the index to 0
                            // or if the index is the same as the original
index
                            // then there are no available times
                            // So we'll check Friday.
                            if (index - 1 == startTimes.length - 1) {
                                // We hit the end of the times array
                                // Go to start of time slots
                                System.out.println("Hit end of time slots.
Going to beginning");
                                index = 0;
```

```
                            } else if (index != originalIndex) {
                                // If current time !== starting time
                                // Means we haven't gone through all times
                                System.out.println("WE'RE CHANGING THE
TIME");
                                schedule.setStartTime(startTimes[index]);
                                schedule.setEndTime(endTimes[index]);
                                index++;
                            } else {
                                // Nothing special, just increment the index
                                System.out.println("Hit original index.
Checking Friday");
                                // Check Friday
                                System.out.println("Changing date to
Friday");
                                schedule.setDays("F");
                                index = 0;

 schedule.setStartTime(fridayStartTimes[index]);
                                schedule.setEndTime(fridayEndTimes[index]);
                            }
                        }
                    }
                }
            }
            return date;
        } else {
            // available times are any mark between:
            // 9:00 and 3:00
            // An array of the available start times
            String[] startTimes = { "09:00:00", "10:00:00", "11:00:00",
"12:00:00", "13:00:00", "14:00:00",
                    "15:00:00" };
            // An array of the available end times
            String[] endTimes = { "10:00:00", "11:00:00", "12:00:00",
"13:00:00", "14:00:00", "15:00:00", "16:00:00" };

            // An array of the available Friday start times
            // 09:00:00 - 16:00:00
            String[] fridayStartTimes = { "09:00:00", "10:00:00",
"11:00:00", "12:00:00", "13:00:00", "14:00:00",
                    "15:00:00", "16:00:00" };
            // An array of the available Friday end times
            String[] fridayEndTimes = { "10:00:00", "11:00:00", "12:00:00",
"13:00:00", "14:00:00", "15:00:00",
                    "16:00:00", "17:00:00" };

            String currentTime = schedule.getStartTime();
            // Get the index of the currentTime within the startTimes array
            int originalIndex =
```

```
Arrays.asList(startTimes).indexOf(currentTime);
            int index = originalIndex + 1;
            boolean continueExec = true;
            String[] date = new String[4];
            ArrayList<Schedule> schedules;
            // Making a copy of the currently selected days
            // So I can reference later what day I'm switching the
            // times at
            String originalDate = schedule.getDays();
            while (continueExec) {

                // Check the same date for a different room
                schedules = findSchedule(schedule.getStartTime(),
schedule.getEndTime(), schedule.getDays());

                // We shouldn't have to worry about schedules being empty
                // as we've already checked for conflicts

                System.out.println("Searching day " + schedule.getDays() + "
at time " + schedule.getStartTime() + " - "
                        + schedule.getEndTime());

                // Loop through and create a csv
                // of the used classrooms
                String usedClassrooms = "";
                for (Schedule s : schedules) {
                    // If last index, don't add comma
                    if (schedules.indexOf(s) == schedules.size() - 1) {
                        usedClassrooms += s.getClassroomId();
                    } else {
                        usedClassrooms += s.getClassroomId() + ",";
                    }
                    // System.out.println("Unavailable classroom: " +
s.getClassroomId());
                }

                // Query the classroom_table to get all the classrooms
                // except for the ones that are already being used
                ArrayList<Integer> availableClassrooms =
findAvailableClassrooms(usedClassrooms);

                // If classrooms available, then set the classroom id
                // and return the new schedule
                if (availableClassrooms.size() > 0) {
                    // System.out.println("Available classrooms: " +
availableClassrooms);
                    date[0] = schedule.getStartTime();
                    date[1] = schedule.getEndTime();
                    date[2] = schedule.getDays();
                    date[3] = availableClassrooms.get(0).toString();
```

```
                    date[3] = availableClassrooms.get(0).toString();
                    continueExec = false;
                } else {

                    // Check if date if MW. If so, then change it to TR.
                    // If TR, then change it to MW.
                    if (schedule.getDays().equals("M")) {
                        // System.out.println("Changing date to TR");
                        schedule.setDays("T");
                    } else if (schedule.getDays().equals("T")) {
                        // System.out.println("Changing date to MW");
                        schedule.setDays("W");
                    } else if (schedule.getDays().equals("W")) {
                        // System.out.println("Changing date to TR");
                        schedule.setDays("R");
                    } else if (schedule.getDays().equals("R")) {
                        // System.out.println("Changing date to MW");
                        schedule.setDays("M");
                    }

                    if (schedule.getDays() == "F") {
                        // If index is at the end
                        // Then stop the loop because
                        // no date is available
                        if (index == startTimes.length - 1) {
                            continueExec = false;
                        } else {
                            // Increment the index
                            index++;
                        }
                        schedule.setStartTime(fridayStartTimes[index]);
                        schedule.setEndTime(fridayEndTimes[index]);
                    } else {
                        if (originalDate.equals(schedule.getDays())) {
                            // No classrooms available.
                            // Check if the index is at the end of the array
                            // if so, then reset the index to 0
                            // or if the index is the same as the original
index
                            // then there are no available times
                            // So we'll check Friday.
                            if (index - 1 == startTimes.length - 1) {
                                // We hit the end of the times array
                                // Go to start of time slots
                                System.out.println("Hit end of time slots.
Going to beginning");
                                index = 0;
                            } else if (index != originalIndex) {
                                // If current time !== starting time
                                // Means we haven't gone through all times
```

```
                                System.out.println("WE'RE CHANGING THE
TIME");
                                schedule.setStartTime(startTimes[index]);
                                schedule.setEndTime(endTimes[index]);
                                index++;
                            } else {
                                // Nothing special, just increment the index
                                System.out.println("Hit original index.
Checking Friday");
                                // Check Friday
                                System.out.println("Changing date to
Friday");
                                schedule.setDays("F");
                                index = 0;

 schedule.setStartTime(fridayStartTimes[index]);
                                schedule.setEndTime(fridayEndTimes[index]);
                            }
                        }
                    }
                }
            }
            return date;
        }
    }

    private ArrayList<Integer> findAvailableClassrooms(String
usedClassrooms) {
        // I was going to make this DB driven but decided not to
        // as the assignment description
        // only mentioned the 3 rooms.

        // Available classrooms are A, B, and C with their indexes
        // being 1,2,3 respectively

        // Create an array of the available classrooms
        // and remove the ones that are already being used
        ArrayList<Integer> availableClassrooms = new ArrayList<Integer>();
        availableClassrooms.add(1);
        availableClassrooms.add(2);
        availableClassrooms.add(3);

        System.out.println("Unavailable classrooms: " + usedClassrooms);

        // If there are no used classrooms, then return the available
        // classrooms
        if (usedClassrooms.equals("")) {
            return availableClassrooms;
        }

```

```
        // The usedClassrooms is a csv of the used classrooms
        // we need to remove these from the available classrooms
        String[] usedClassroomsArray = usedClassrooms.split(",");
        if (usedClassroomsArray.length == 3) {
            // All are gone
            availableClassrooms.clear();
        } else {
            for (String s : usedClassroomsArray) {
                // Remove the used classroom from the available classrooms
                // System.out.println("Removing classroom: " + s);
                try {

 availableClassrooms.remove(availableClassrooms.indexOf(Integer.parseInt(s))
);
                } catch (Exception e) {
                    // System.out.println("Error removing classroom: " + s);
                    // This means that we are completely out of room
                }
            }
        }

        System.out.println("Available classrooms: " + availableClassrooms);
        return availableClassrooms;
    }

    public ArrayList<Schedule> getSchedule() {
        ArrayList<Schedule> schedule = new ArrayList<Schedule>();

        // Get the entire schedule from the database while joining the
classroom_table
        // and professor_table and courses_table
        String sql = "SELECT * FROM schedule_table JOIN classroom_table ON
schedule_table.classroom_tuid = classroom_table.tuid JOIN professors_table
ON schedule_table.professor_tuid = professors_table.tuid JOIN courses_table
ON schedule_table.course_tuid = courses_table.tuid;";

        // Execute the query and loop through the results and
        // output the console the results
        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            stmt.execute(sql);

            ResultSet rs = stmt.getResultSet();

            // loop through the result set
            while (rs.next()) {
                // System.out.println(rs.getString("tuid"));
                schedule.add(
                        new Schedule(rs.getInt("course_tuid"),
rs.getInt("classroom_tuid"), rs.getInt("professor_tuid"),
```

```
                                    rs.getInt("section"),
rs.getString("start_time"), rs.getString("end_time"),
                                    rs.getString("days"),
rs.getString("course_title"), rs.getString("professor_name"),
                                    rs.getString("classroom_name")));
            }

        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

        return schedule;
    }

    public ArrayList<Professor> getProfessors() {
        // Return all professors
        ArrayList<Professor> professors = new ArrayList<Professor>();

        // Get all the professors from the DB
        String sql = "SELECT * FROM professors_table";

        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            stmt.execute(sql);

            ResultSet rs = stmt.getResultSet();

            // loop through the result set
            while (rs.next()) {
                // System.out.println(rs.getString("tuid"));
                professors.add(new Professor(rs.getInt("tuid"),
rs.getString("professor_name")));
            }

        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

        return professors;
    }

    public ArrayList<Schedule> getFacultySchedule() {
        ArrayList<Schedule> schedule = new ArrayList<Schedule>();

        // Get the entire schedule from the database while joining the
classroom_table
        // and professor_table and courses_table
        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
```

```
JOIN courses_table ON schedule_table.course_tuid = courses_table.tuid ORDER
BY professors_table.tuid;";

        // Execute the query and loop through the results and
        // output the console the results
        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            stmt.execute(sql);

            ResultSet rs = stmt.getResultSet();

            // loop through the result set
            while (rs.next()) {
                // System.out.println(rs.getString("tuid"));
                schedule.add(
                        new Schedule(rs.getInt("course_tuid"),
rs.getInt("classroom_tuid"), rs.getInt("professor_tuid"),
                                rs.getInt("section"),
rs.getString("start_time"), rs.getString("end_time"),
                                rs.getString("days"),
rs.getString("course_title"), rs.getString("professor_name"),
                                rs.getString("classroom_name")));
            }

        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

        return schedule;
    }

    public ArrayList<Schedule> getScheduleByProfessor(Integer id) {
        // Get the schedule for a specific professor
        ArrayList<Schedule> schedule = new ArrayList<Schedule>();

        // Get the entire schedule from the database while joining the
classroom_table
        // and professor_table and courses_table
        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 LEFT
JOIN classroom_table ON schedule_table.classroom_tuid = classroom_table.tuid
WHERE professors_table.tuid = "
                + id + " ORDER BY professors_table.tuid;";

        // Execute the query and loop through the results and
        // output the console the results
        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            stmt.execute(sql);
```

```
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"),
1150                        rs.getString("classroom_name"));
1151                classes.setCredits((rs.getInt("credit_hours")));
1152                schedule.add(classes);
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"),
1181                        rs.getInt("credit_hours")));
1182            }
1183
1184        } catch (SQLException e) {
1185            System.out.println(e.getMessage());
```

```
        }
        return courses;
    }

    public ArrayList<Schedule> getScheduleByCourse(Integer courseId) {
        // Get the schedule for all courses with the same course id
        // Joining the classroom_table and courses_table
        ArrayList<Schedule> schedule = new ArrayList<Schedule>();

        // Get the entire schedule from the database while joining the
classroom_table
        // and professor_table and courses_table
        String sql = "SELECT * FROM courses_table INNER JOIN schedule_table
ON courses_table.tuid = schedule_table.course_tuid LEFT JOIN
professors_table ON schedule_table.professor_tuid = professors_table.tuid
LEFT JOIN classroom_table ON schedule_table.classroom_tuid =
classroom_table.tuid WHERE courses_table.tuid = "
                + courseId + " ORDER BY courses_table.tuid;";
        // Execute the query and loop through the results and
        // output the console the results
        try (Connection conn = this.connect();
                Statement stmt = conn.createStatement()) {
            stmt.execute(sql);

            ResultSet rs = stmt.getResultSet();

            // loop through the result set
            while (rs.next()) {
                // System.out.println(rs.getString("tuid"));
                Schedule course = new Schedule(rs.getInt("course_tuid"),
rs.getInt("classroom_tuid"),
                        rs.getInt("professor_tuid"),
                        rs.getInt("section"), rs.getString("start_time"),
rs.getString("end_time"),
                        rs.getString("days"), rs.getString("course_title"),
rs.getString("professor_name"),
                        rs.getString("classroom_name"));
                course.setCapacity((rs.getInt("capacity")));
                schedule.add(course);
            }

        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

        return schedule;
    }
}
```