Algorithms CA1 Java Application

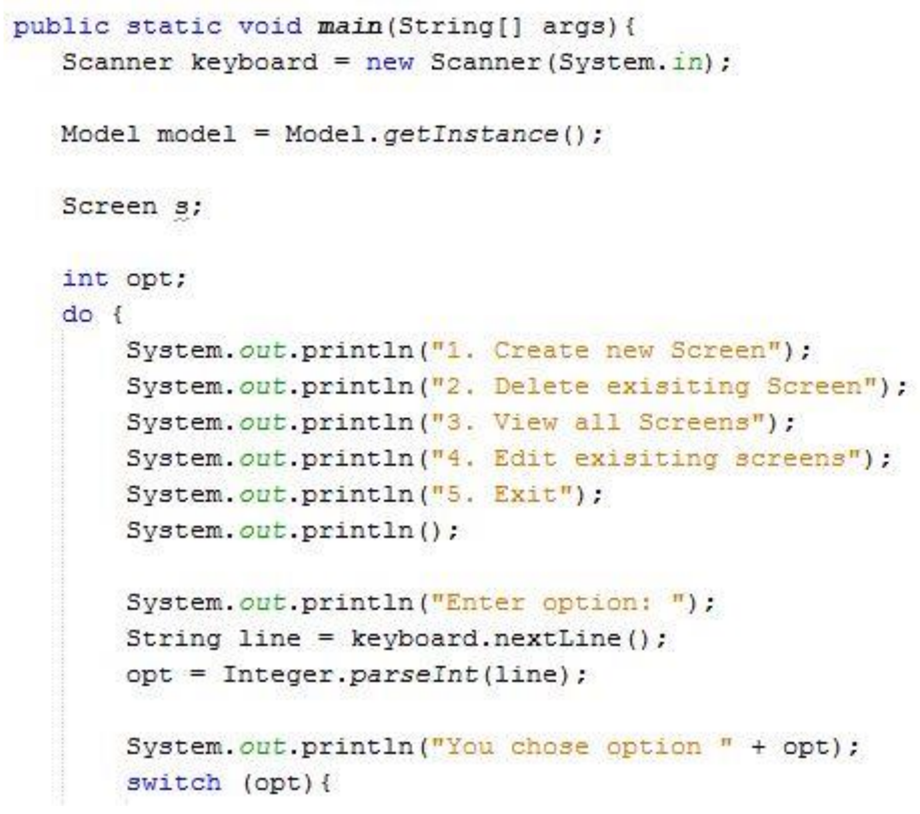
N00130962

Melissa Doyle

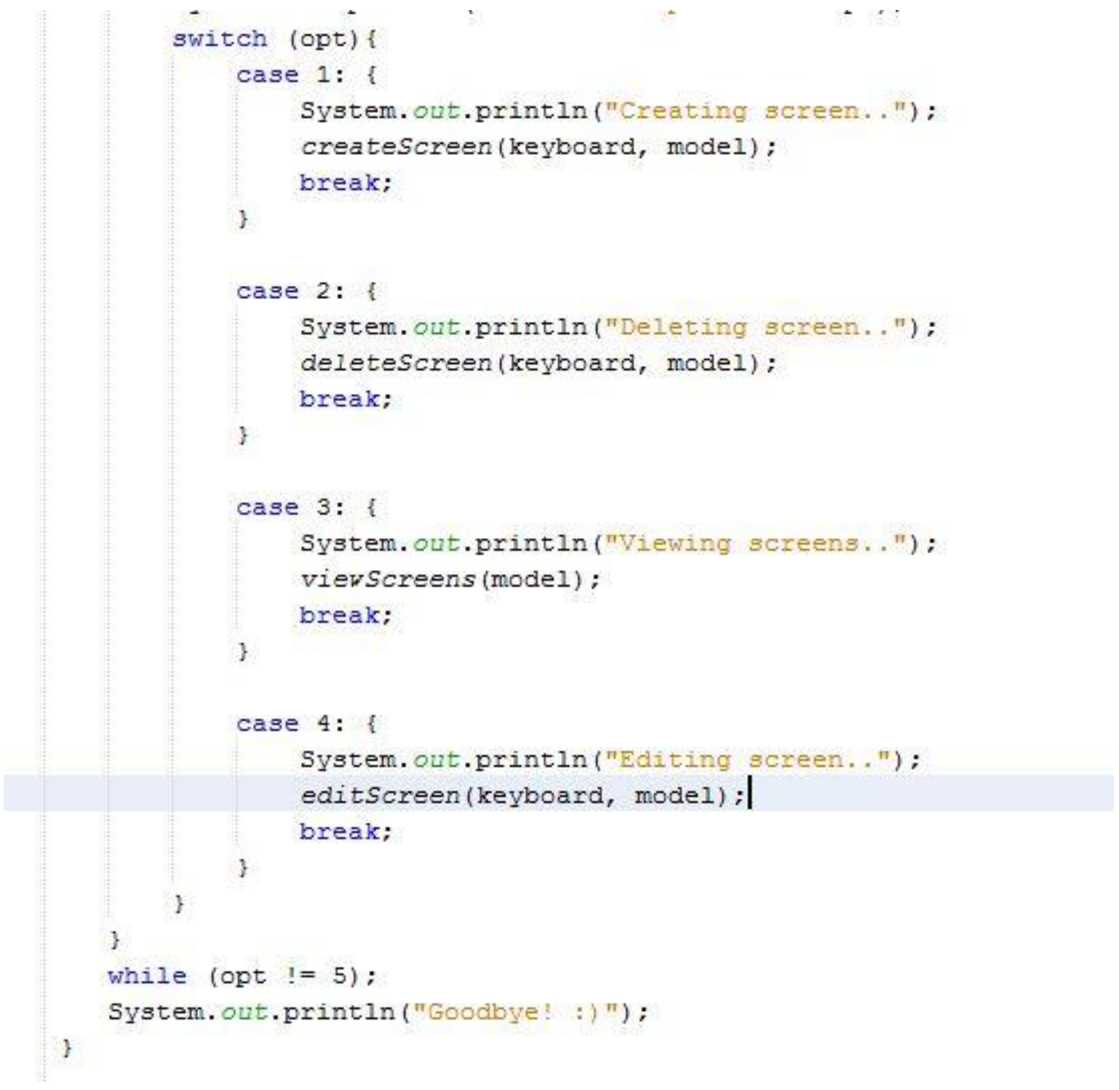
My case study was the Cinema which I have based my application upon. Firstly, I have created a table called screendb in phpMyAdmin which is linked to the database. In this table, I have an ID, name, numSeats, numExits, dateNextInspection and projectorType. These are my variables in the application itself.

In the Java application itself, I created a Java class called Screen. Here, I declared the variables as private, meaning they can only be called from inside the class they are declared in. COME BACK

I then created a class called DemoApp. Firstly I've written my main method, followed by the menu shown on the screen which is used for the user interface. This is so the user can see the option and then chose whatever option they wish. This is put inside a loop as we can see in the code so the menu is shown continuously to the user until they wish to exit the program.



```
public static void main(String[] args){
    Scanner keyboard = new Scanner(System.in);

    Model model = Model.getInstance();

    Screen s;

    int opt;
    do {
        System.out.println("1. Create new Screen");
        System.out.println("2. Delete exisiting Screen");
        System.out.println("3. View all Screens");
        System.out.println("4. Edit exisiting screens");
        System.out.println("5. Exit");
        System.out.println();

        System.out.println("Enter option: ");
        String line = keyboard.nextLine();
        opt = Integer.parseInt(line);

        System.out.println("You chose option " + opt);
        switch (opt){
```



```
            switch (opt){
                case 1: {
                    System.out.println("Creating screen..");
                    createScreen(keyboard, model);
                    break;
                }

                case 2: {
                    System.out.println("Deleting screen..");
                    deleteScreen(keyboard, model);
                    break;
                }

                case 3: {
                    System.out.println("Viewing screens..");
                    viewScreens(model);
                    break;
                }

                case 4: {
                    System.out.println("Editing screen..");
                    editScreen(keyboard, model);
                    break;
                }
            }
        }
        while (opt != 5);
        System.out.println("Goodbye! :)");
    }
```

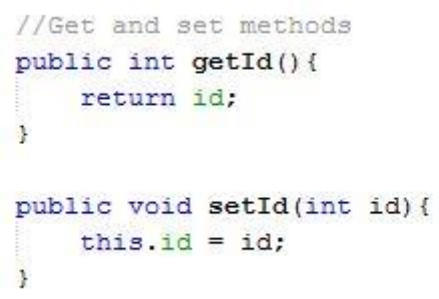
I then created a new Java class called Screen. In here I have created a Constructor called Screen.

public Screen(int id, String n, int ne, int ns, String dni, String pt) {

This refers to the current objects in the Constructor.

```
this.id = id;
        this.name = n;
        this.numSeats = ne;
        this.numExits = ns;
        this.dateNextInspection = dni;
        this.projectorType = pt;
```

I then created get and set methods for each of the attributes in my table. The get method accesses the variables and the set method changes the values in them if needed.



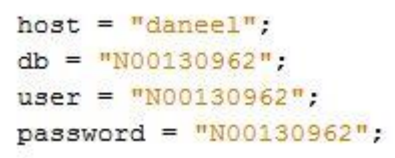
```
//Get and set methods
public int getId(){
    return id;
}

public void setId(int id){
    this.id = id;
}
```

To connect to the database, I needed a class called DBConnection. This code uses the Singleton pattern meaning it doesn't need a constructor, it uses a static method instead and returns a connection object. This is needed to connect to SQL. A SQLException is used in case something goes wrong with the connection.

Class.forName("com.mysql.jdbc.Driver"); loads the driver for the MySQL database.

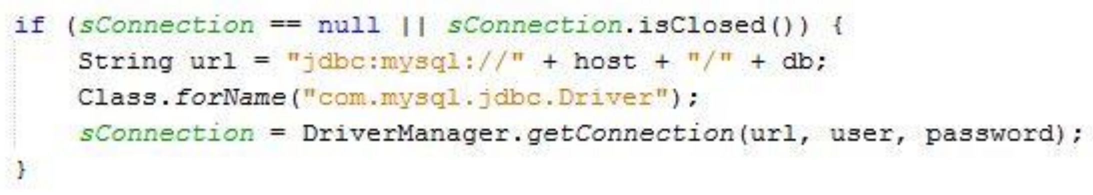
import java.sql.DriverManager; this code is also needed to correspond with the above code.

I then created variables with my details to log into the database.



```
host = "daneel";
db = "N00130962";
user = "N00130962";
password = "N00130962";
```

The static method sConnection is needed.



```
if (sConnection == null || sConnection.isClosed()) {
    String url = "jdbc:mysql://" + host + "/" + db;
    Class.forName("com.mysql.jdbc.Driver");
    sConnection = DriverManager.getConnection(url, user, password);
}
```

If the connection is null or closed, the String url is needed and the connection will go to the URL "jdbc:mysql://" with the user name and password, while calling the sConnection.

Next, I created a Model class which also uses the Singleton pattern. There is also a connection in here to the database.

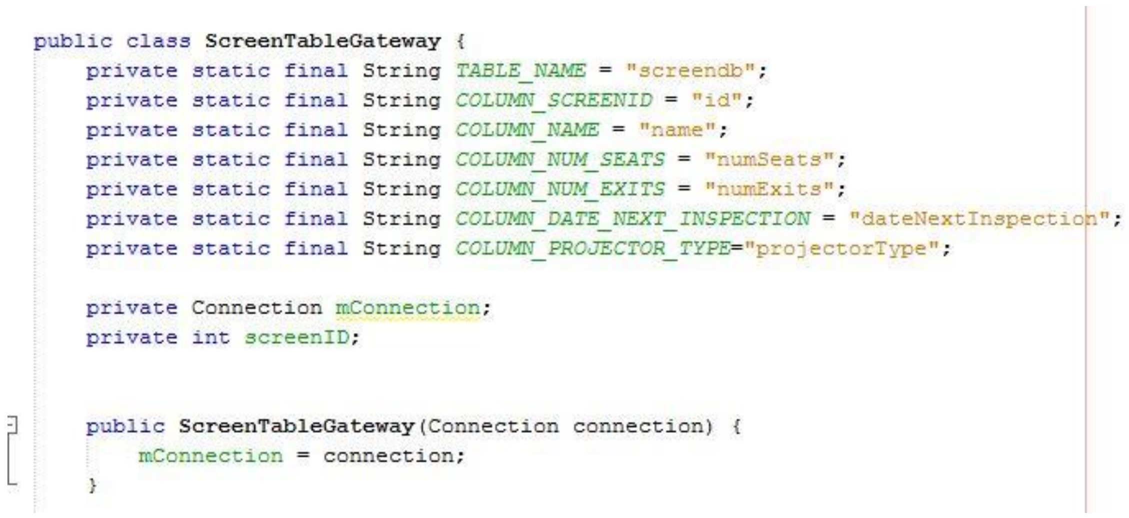
private List<Screen> screendb;

ScreenTableGateway gateway;

private Model() {

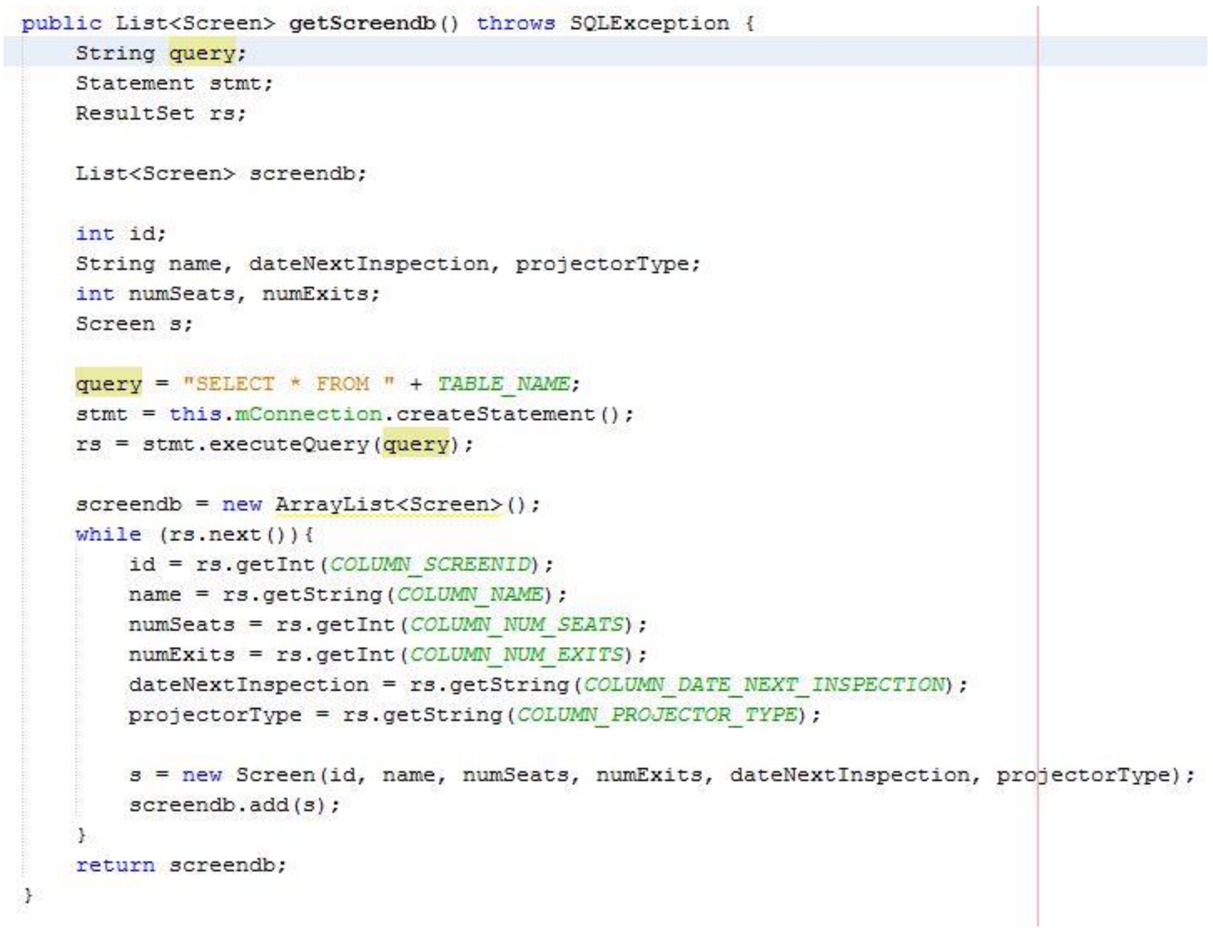
```
try {
    Connection conn = DBConnection.getInstance();
    gateway = new ScreenTableGateway(conn);
    this.screendb = this.gateway.getScreendb();
}
catch (ClassNotFoundException ex) {
    Logger.getLogger(Model.class.getName()).log(Level.SEVERE, null, ex);
}
catch (SQLException ex) {
    Logger.getLogger(Model.class.getName()).log(Level.SEVERE, null, ex);
```

A TableGateway class is needed to access gateway into a table. I called mine ScreenTableGateway. This class is the access into the table. This then has a connection to the database. A constructor is used and references to a table gateway.



```
public class ScreenTableGateway {
    private static final String TABLE_NAME = "screendb";
    private static final String COLUMN_SCREENID = "id";
    private static final String COLUMN_NAME = "name";
    private static final String COLUMN_NUM_SEATS = "numSeats";
    private static final String COLUMN_NUM_EXITS = "numExits";
    private static final String COLUMN_DATE_NEXT_INSPECTION = "dateNextInspection";
    private static final String COLUMN_PROJECTOR_TYPE="projectorType";

    private Connection mConnection;
    private int screenID;


    public ScreenTableGateway(Connection connection) {
        mConnection = connection;
    }
```

This code is constant variables which makes it easier to change if needed any time.

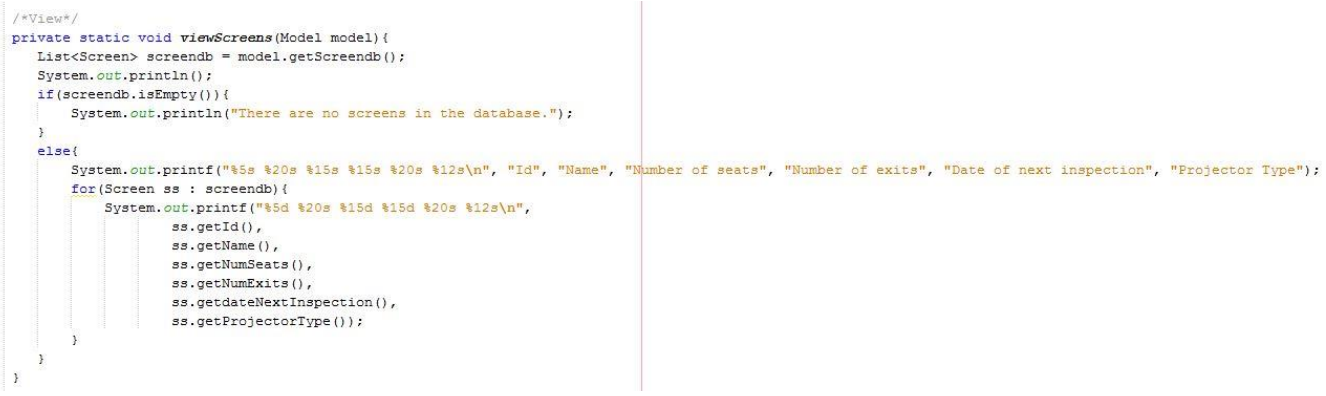


```java
public List<Screen> getScreendb() throws SQLException {
    String query;
    Statement stmt;
    ResultSet rs;

    List<Screen> screendb;

    int id;
    String name, dateNextInspection, projectorType;
    int numSeats, numExits;
    Screen s;

    query = "SELECT * FROM " + TABLE_NAME;
    stmt = this.mConnection.createStatement();
    rs = stmt.executeQuery(query);

    screendb = new ArrayList<Screen>();
    while (rs.next()){
        id = rs.getInt(COLUMN_SCREENID);
        name = rs.getString(COLUMN_NAME);
        numSeats = rs.getInt(COLUMN_NUM_SEATS);
        numExits = rs.getInt(COLUMN_NUM_EXITS);
        dateNextInspection = rs.getString(COLUMN_DATE_NEXT_INSPECTION);
        projectorType = rs.getString(COLUMN_PROJECTOR_TYPE);

        s = new Screen(id, name, numSeats, numExits, dateNextInspection, projectorType);
        screendb.add(s);
    }
    return screendb;
}
```

The String query is needed for execution, the Statement object is needed to execute the SQL query, the ResultsSet shows the results of the SQL query and then the List<Screen>screendb contains the Screen objects. Screen s reads in the details.

Next the statement object exectute the query in a results variable. The while loops through each line in the table. An empty Screen array is made. The results of each row are stored in a Screen object and the object is added to the list of screens. The list of screens in then returned.

I then started to create the code for viewing the screens in the database. Here is the code for View needed in the DemoApp class.



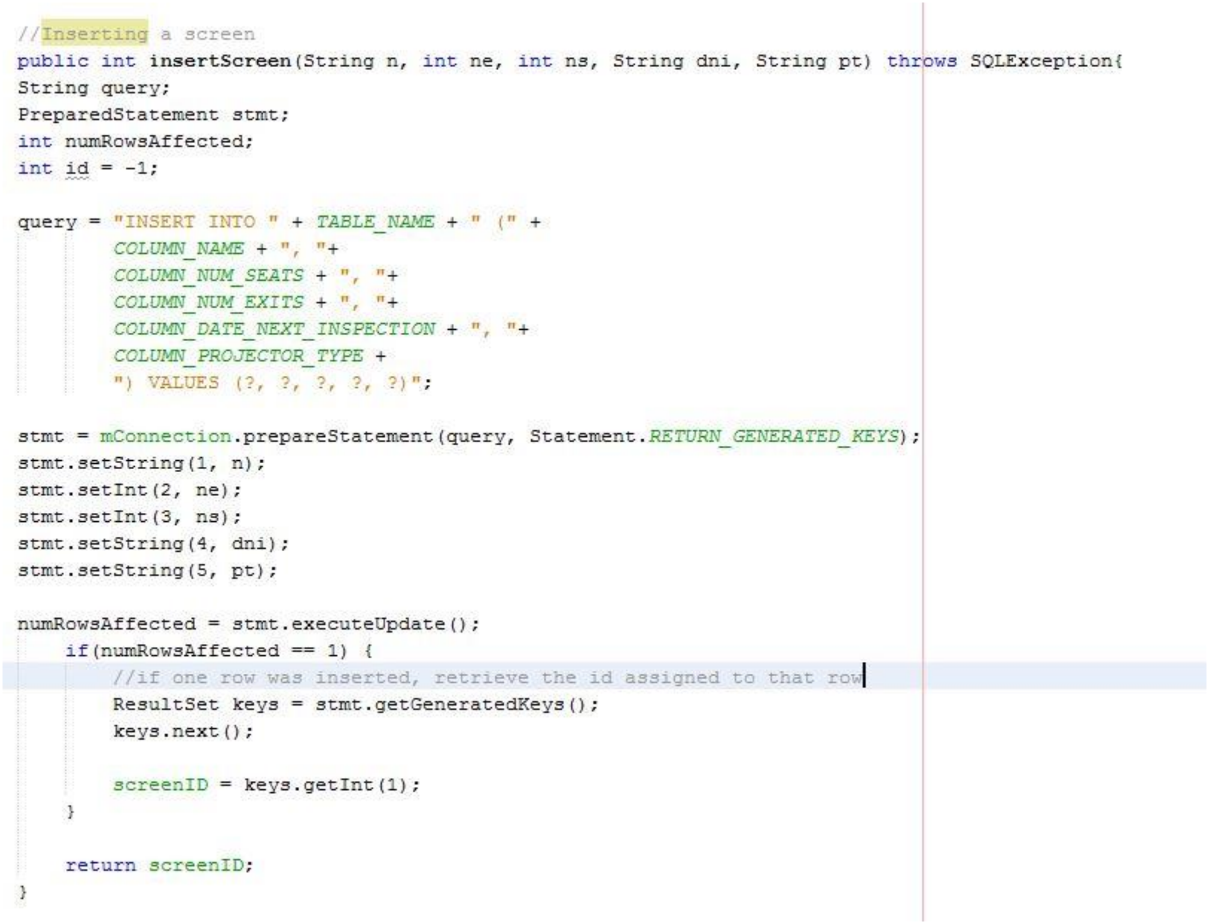
```java
/*View*/
private static void viewScreens(Model model){
    List<Screen> screendb = model.getScreendb();
    System.out.println();
    if(screendb.isEmpty()){
        System.out.println("There are no screens in the database.");
    }
    else{
        System.out.printf("%5s %20s %15s %15s %20s %12s\n", "Id", "Name", "Number of seats", "Number of exits", "Date of next inspection", "Projector Type");
        for(Screen ss : screendb){
            System.out.printf("%5d %20s %15d %15d %20s %12s\n",
                    ss.getId(),
                    ss.getName(),
                    ss.getNumSeats(),
                    ss.getNumExits(),
                    ss.getdateNextInspection(),
                    ss.getProjectorType());
        }
    }
}
```

The if statement prints out a message “no screens” if the table in empty. Else, it retrieves the data from the table and prints it out. The “%5s %20s,” etc, is used for spacing out the data from the table, making it more easy to read. There is an error when we run this code which is why we need to import a library called MySQL JDBC Driver.

```
case 3: {
            System.out.println("Viewing screens..");
            viewScreens(model);
            break;
```

At the start of the code then, I put this piece of code in which is a reference to the model object. This creates a method and I inserted the above snippet of code in here.

Next, I created the code the Insert a new screen. In the ScreenTableGateway, we have similar code to the view code.



```
//Inserting a screen
public int insertScreen(String n, int ne, int ns, String dni, String pt) throws SQLException{
String query;
PreparedStatement stmt;
int numRowsAffected;
int id = -1;

query = "INSERT INTO " + TABLE_NAME + " (" +
        COLUMN_NAME + ", "+
        COLUMN_NUM_SEATS + ", "+
        COLUMN_NUM_EXITS + ", "+
        COLUMN_DATE_NEXT_INSPECTION + ", "+
        COLUMN_PROJECTOR_TYPE +
        ") VALUES (?, ?, ?, ?, ?)";

stmt = mConnection.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
stmt.setString(1, n);
stmt.setInt(2, ne);
stmt.setInt(3, ns);
stmt.setString(4, dni);
stmt.setString(5, pt);

numRowsAffected = stmt.executeUpdate();
    if(numRowsAffected == 1) {
        //if one row was inserted, retrieve the id assigned to that row
        ResultSet keys = stmt.getGeneratedKeys();
        keys.next();

        screenID = keys.getInt(1);
    }

    return screenID;
}
```

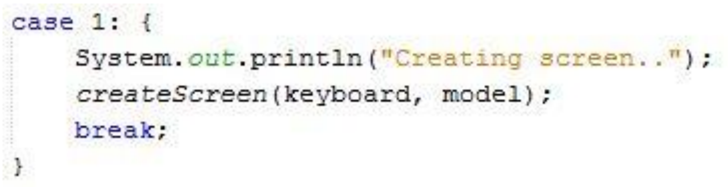
In this case, the id is given a default value of -1, as it is not a valid id, when I run the code and get this value back, I will know that there is an error in the code.

VALUES(?, ?, ?, ?, ?)”; is used for the id. We don’t give this a specific value as it is auto incremented in the database itself. The five values are inserted in the statement object. This is then executed. The NumRowsAffected basically tells us how many rows were affected. If the result is equal to 1, it will give us the generated keys in the results set.

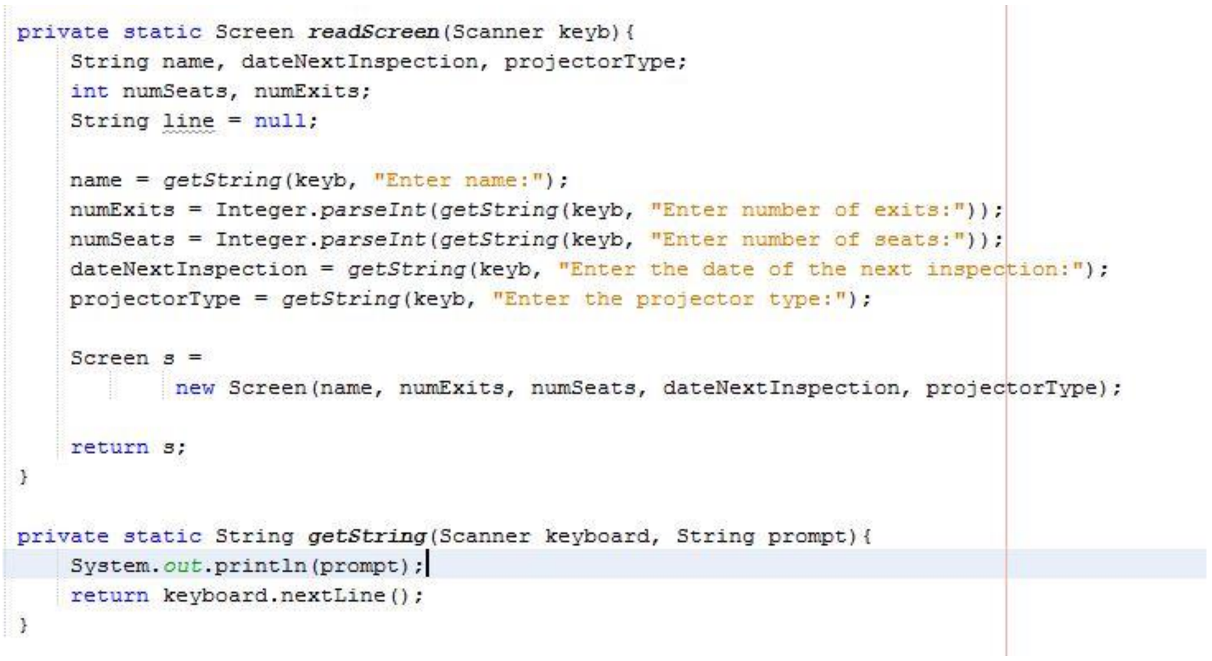
Keys.next(); moves to the first row.

screenID = keys.getInt(1); will give the value in the first row.

I then move back to the DemoApp class and we repeat this similar to view screens.

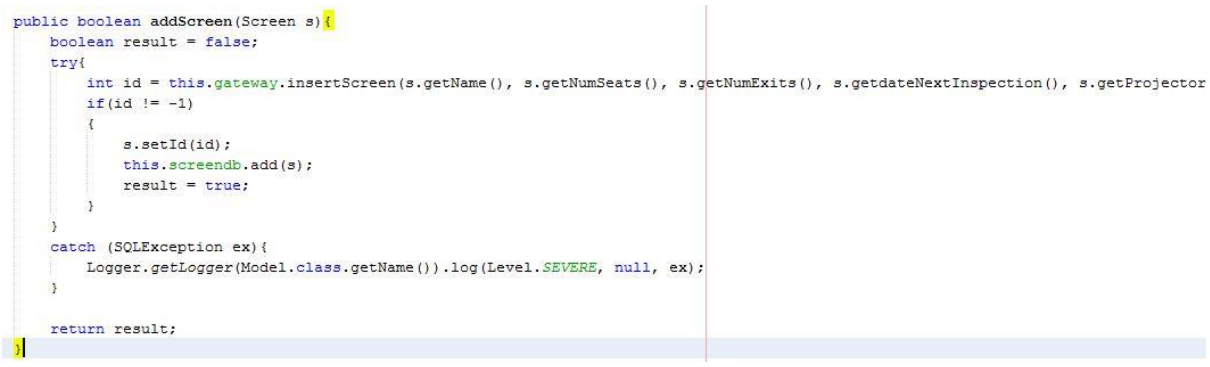


```
case 1: {
    System.out.println("Creating screen..");
    createScreen(keyboard, model);
    break;
}
```



```
private static Screen readScreen(Scanner keyb){
    String name, dateNextInspection, projectorType;
    int numSeats, numExits;
    String line = null;

    name = getString(keyb, "Enter name:");
    numExits = Integer.parseInt(getString(keyb, "Enter number of exits:"));
    numSeats = Integer.parseInt(getString(keyb, "Enter number of seats:"));
    dateNextInspection = getString(keyb, "Enter the date of the next inspection:");
    projectorType = getString(keyb, "Enter the projector type:");

    Screen s =
            new Screen(name, numExits, numSeats, dateNextInspection, projectorType);

    return s;
}

private static String getString(Scanner keyboard, String prompt){
    System.out.println(prompt);
    return keyboard.nextLine();
}
```

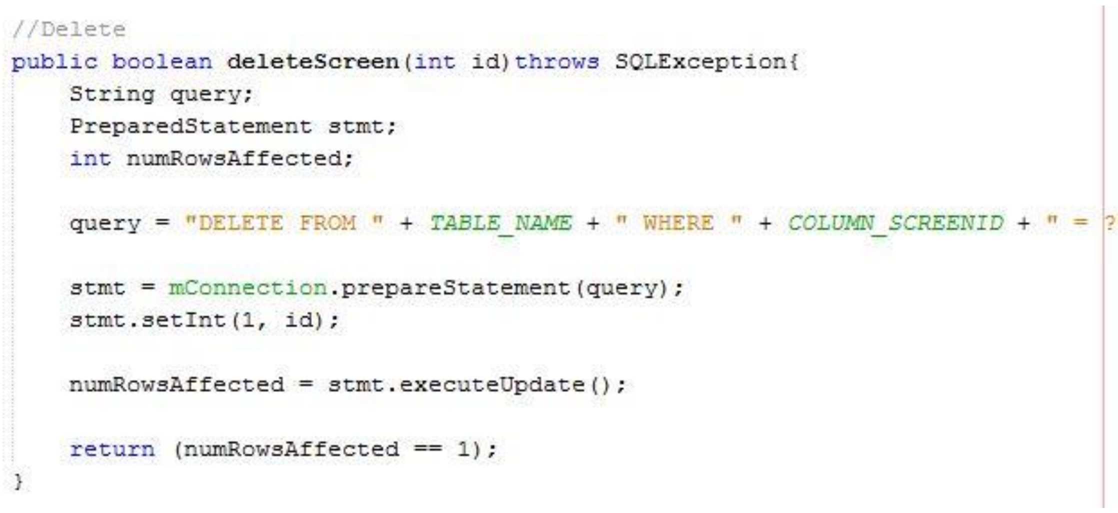
This will code will print out “Enter name:”, etc for the user so they can enter in their values to create a new screen. Everything gets read in as a String so it will not causes errors if numbers are inserted. For numbers that need to be inserted like numExits and numSeats, we use a parseInt. This gets reads in as a string but is outputted as an int which is what we want. Screen s reads this in and returns it in s. It then reads in what is written in the keyboard object and print it out in a prompt. This will then display a new Screen in the database.



```java
public boolean addScreen(Screen s){
    boolean result = false;
    try{
        int id = this.gateway.insertScreen(s.getName(), s.getNumSeats(), s.getNumExits(), s.getdateNextInspection(), s.getProjector
        if(id != -1)
        {
            s.setId(id);
            this.screendb.add(s);
            result = true;
        }
    }
    catch (SQLException ex){
        Logger.getLogger(Model.class.getName()).log(Level.SEVERE, null, ex);
    }

    return result;
}
```

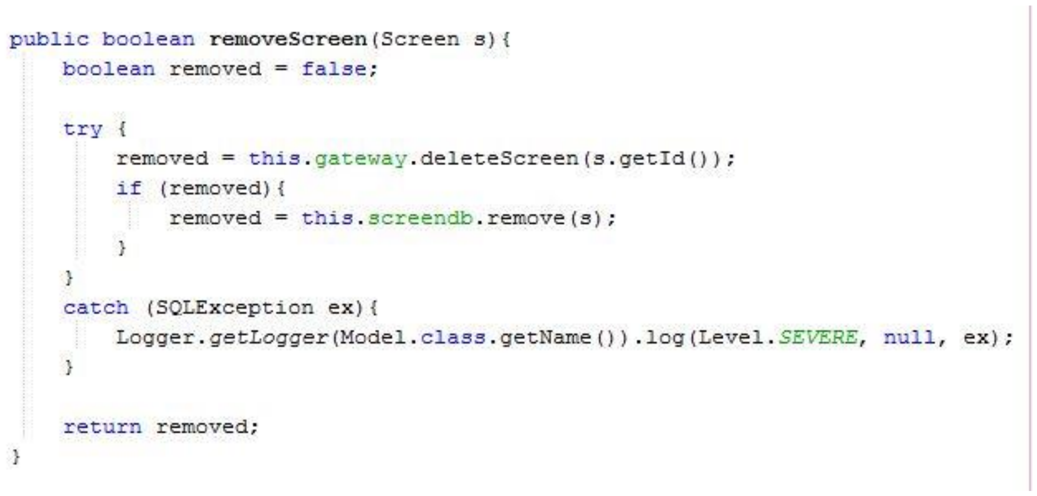
If the id is not -1, this means the screen is added successfully. This.screendb.add(s); will add this screen to the database. The Boolean type will return a false result if the screen is not added to the database.

Next, I created the code to delete a Screen.



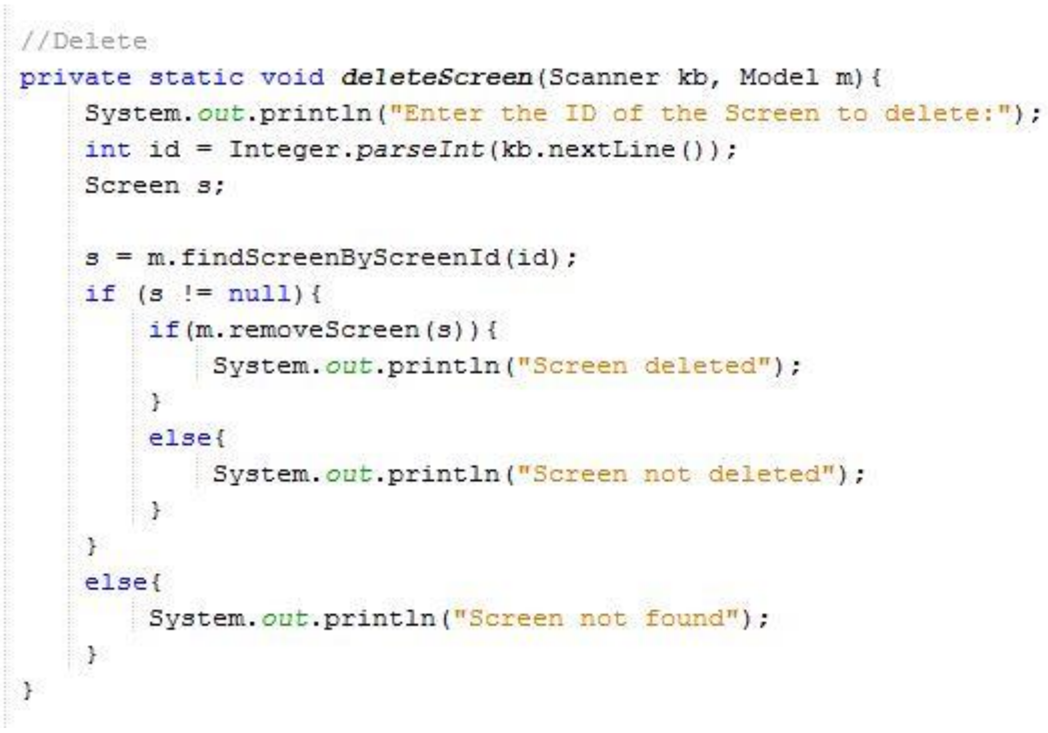
```java
//Delete
public boolean deleteScreen(int id)throws SQLException{
    String query;
    PreparedStatement stmt;
    int numRowsAffected;

    query = "DELETE FROM " + TABLE_NAME + " WHERE " + COLUMN_SCREENID + " = ?";

    stmt = mConnection.prepareStatement(query);
    stmt.setInt(1, id);

    numRowsAffected = stmt.executeUpdate();

    return (numRowsAffected == 1);
}
```

This is code is also similar to the code for the other option. In the ScreenTableGateway class, the Boolean type will return a true or false statement to say if the Screen was removed or not. The return (numRowsAffected == 1); will return a true value if only one row has been deleted.



```java
public boolean removeScreen(Screen s){
    boolean removed = false;

    try {
        removed = this.gateway.deleteScreen(s.getId());
        if (removed){
            removed = this.screendb.remove(s);
        }
    }
    catch (SQLException ex){
        Logger.getLogger(Model.class.getName()).log(Level.SEVERE, null, ex);
    }

    return removed;
}
```

In the Model class, the above code is needed. The Boolean type will set as a default to false as it presumes a row has not been deleted yet. The code will then take a parameter of the ID that needs to be deleted in (s.getId()); and will put it in the removed variable. The if statement will use the remove variable and the row will also be deleted from the s array. This will then be removed in the return removed line of code.

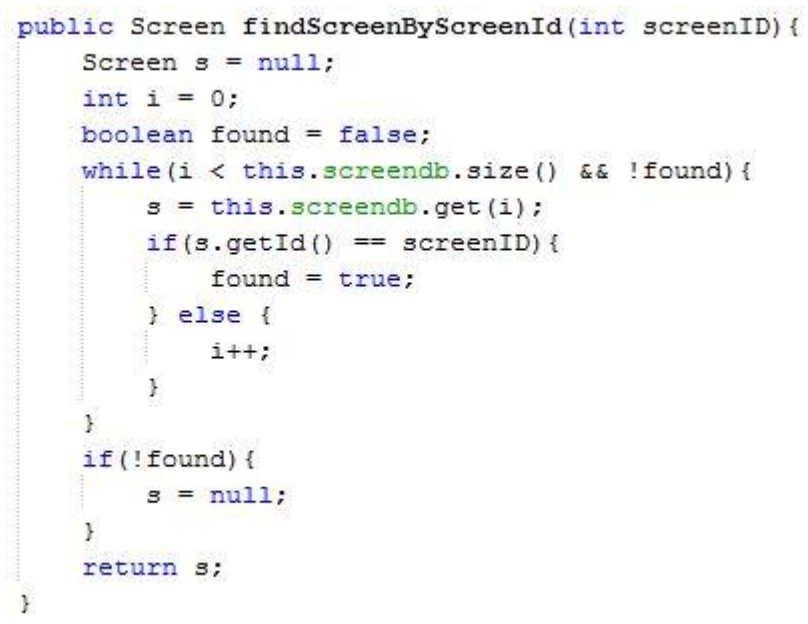


```
//Delete
private static void deleteScreen(Scanner kb, Model m){
    System.out.println("Enter the ID of the Screen to delete:");
    int id = Integer.parseInt(kb.nextLine());
    Screen s;

    s = m.findScreenByScreenId(id);
    if (s != null){
        if(m.removeScreen(s)){
            System.out.println("Screen deleted");
        }
        else{
            System.out.println("Screen not deleted");
        }
    }
    else{
        System.out.println("Screen not found");
    }
}
```

In the DemoApp class, the user is asked to enter the ID of the screen they wish to delete. The ID is passed in through the keyboard object as an int and return into the s variable.

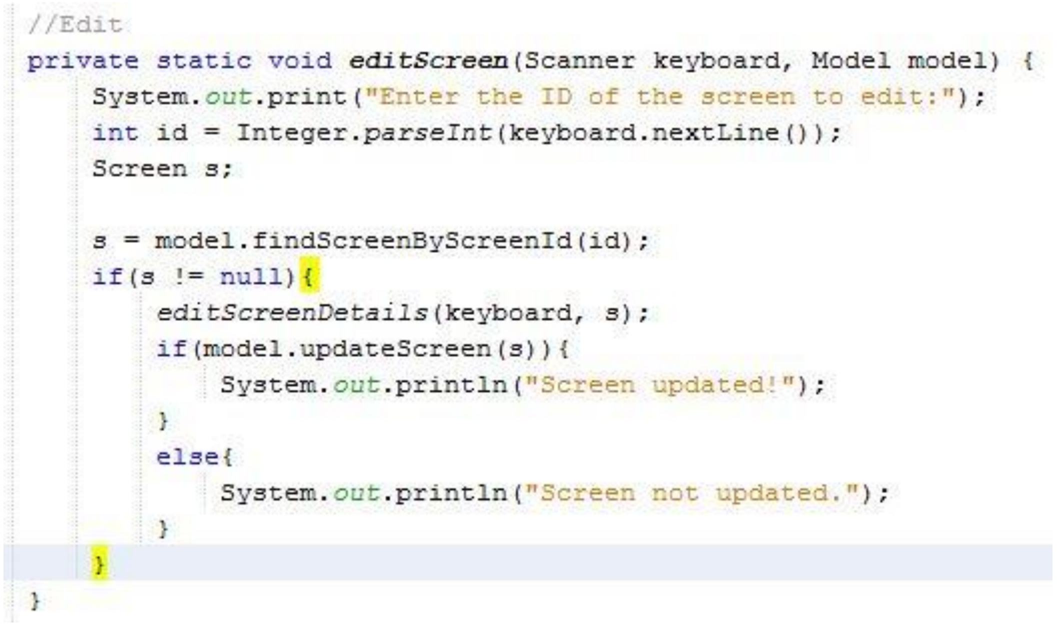
The s = m.findScreenByScreenId(id) will search the database for the ID number entered the user.

If the value that is returned is not nulled, we can remove the screen and it will print out "Screen deleted", otherwise it will read out "Screen not deleted." If the ID number is not found, it will read out that the screen is not found.



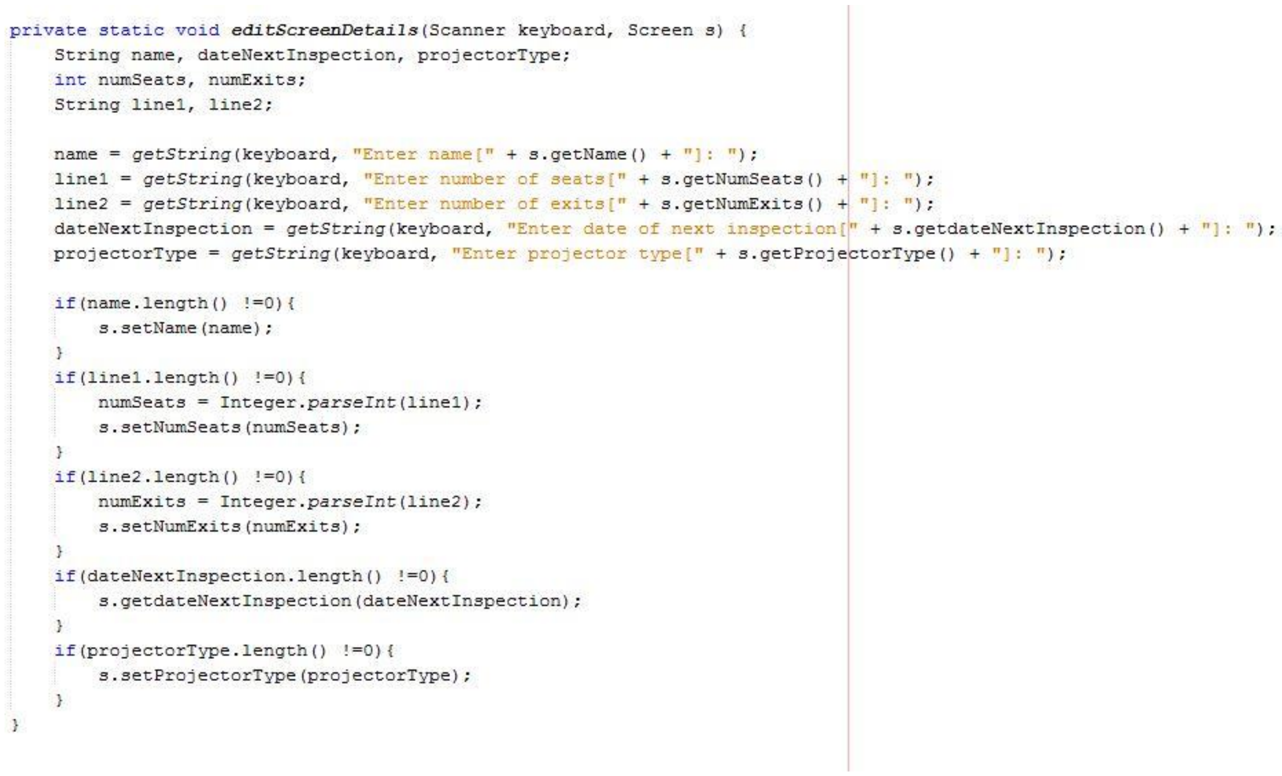
```
public Screen findScreenByScreenId(int screenID){
    Screen s = null;
    int i = 0;
    boolean found = false;
    while(i < this.screendb.size() && !found){
        s = this.screendb.get(i);
        if(s.getId() == screenID){
            found = true;
        } else {
            i++;
        }
    }
    if(!found){
        s = null;
    }
    return s;
}
```

This code is found in the Model class and it will search the array findScreenByScreenId to find the screenID to be removed. The loop will run through until the end of the array and will keep running through the list of screens until it finds the ID. It will return true if the ID is found, if not found, it will return false. It will then return the result in S.



```
//Edit
private static void editScreen(Scanner keyboard, Model model) {
    System.out.print("Enter the ID of the screen to edit:");
    int id = Integer.parseInt(keyboard.nextLine());
    Screen s;

    s = model.findScreenByScreenId(id);
    if(s != null){
        editScreenDetails(keyboard, s);
        if(model.updateScreen(s)){
            System.out.println("Screen updated!");
        }
        else{
            System.out.println("Screen not updated.");
        }
    }
}
```

Lastly, I created the code to edit a Screen. In the DemoApp class, I inserted this code. A message will appear asking the user to enter the ID of the screen they wish to edit. The keyboard object will then read in an int and return it to the s variable. If the id is found and changed, it will read “Screen updated!” otherwise it will read “Screen not updated.”



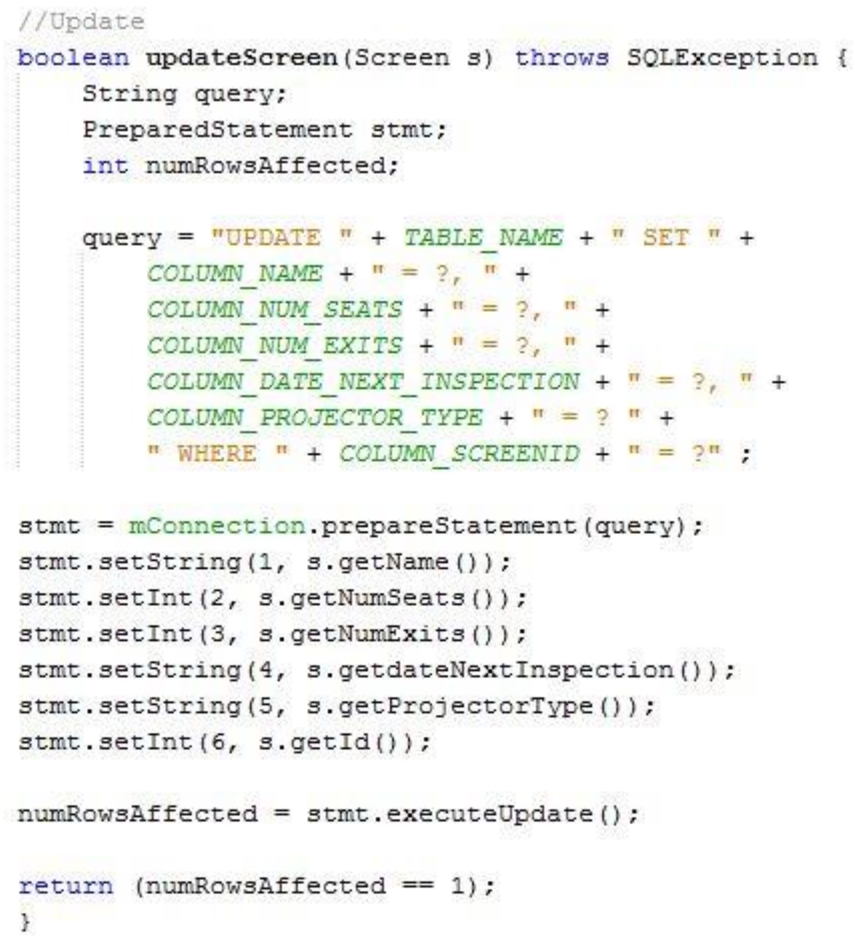
```
private static void editScreenDetails(Scanner keyboard, Screen s) {
    String name, dateNextInspection, projectorType;
    int numSeats, numExits;
    String line1, line2;

    name = getString(keyboard, "Enter name[" + s.getName() + "]: ");
    line1 = getString(keyboard, "Enter number of seats[" + s.getNumSeats() + "]: ");
    line2 = getString(keyboard, "Enter number of exits[" + s.getNumExits() + "]: ");
    dateNextInspection = getString(keyboard, "Enter date of next inspection[" + s.getdateNextInspection() + "]: ");
    projectorType = getString(keyboard, "Enter projector type[" + s.getProjectorType() + "]: ");

    if(name.length() !=0){
        s.setName(name);
    }
    if(line1.length() !=0){
        numSeats = Integer.parseInt(line1);
        s.setNumSeats(numSeats);
    }
    if(line2.length() !=0){
        numExits = Integer.parseInt(line2);
        s.setNumExits(numExits);
    }
    if(dateNextInspection.length() !=0){
        s.getdateNextInspection(dateNextInspection);
    }
    if(projectorType.length() !=0){
        s.setProjectorType(projectorType);
    }
}
```

This code is further down in the DemoApp class. Any ints are also created as Strings as line1 and line2 as they can be read in as Strings later on but parsed in as an integer.

Name = getString(keyboard, “Enter name[“ + s.getName() + “]: “); this piece of code will read in whatever the user types. The square brackets mean that if the user does not want to change this row of information and they hit return, the information will stay the same.

```
if(name.length() !=0){
        s.setName(name);
    }
```

This code will read in if the name is not null, the name entered will become the new name. So if the name is null, it will stay the same.



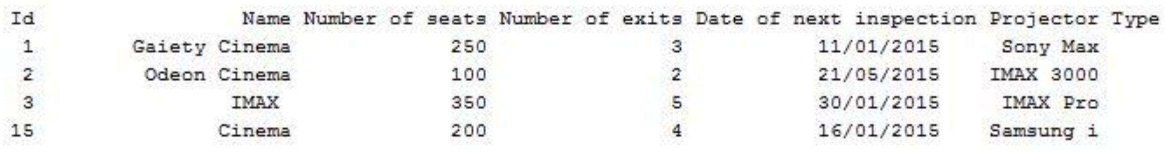
```
//Update
boolean updateScreen(Screen s) throws SQLException {
    String query;
    PreparedStatement stmt;
    int numRowsAffected;

    query = "UPDATE " + TABLE_NAME + " SET " +
        COLUMN_NAME + " = ?, " +
        COLUMN_NUM_SEATS + " = ?, " +
        COLUMN_NUM_EXITS + " = ?, " +
        COLUMN_DATE_NEXT_INSPECTION + " = ?, " +
        COLUMN_PROJECTOR_TYPE + " = ? " +
        " WHERE " + COLUMN_SCREENID + " = ?" ;

stmt = mConnection.prepareStatement(query);
stmt.setString(1, s.getName());
stmt.setInt(2, s.getNumSeats());
stmt.setInt(3, s.getNumExits());
stmt.setString(4, s.getdateNextInspection());
stmt.setString(5, s.getProjectorType());
stmt.setInt(6, s.getId());

numRowsAffected = stmt.executeUpdate();

return (numRowsAffected == 1);
}
```

This code is also very similar to the other code like this. The query = "UPDATE " + TABLE_NAME + " SET " +

COLUMN_NAME + " = ?, " + will update the table and set the new values to the values entered by the user.

The last value is not followed by a comma as the id is followed by a "WHERE " + COLUMN_SCREENID + " = ?" ; and this is what is entered to be updated.

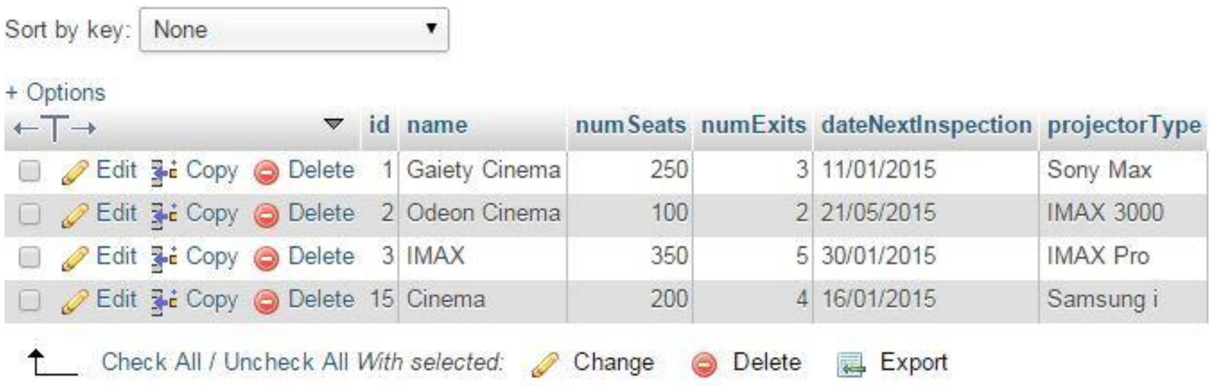
Unfortunately, I have one bug when it comes to creating a new screen. The screens change but the ID number does not automatically change and will follow on from the last created screen. I ran the debugger but I could not find the error.



```
Id              Name Number of seats Number of exits Date of next inspection Projector Type
 1     Gaiety Cinema             250               3              11/01/2015       Sony Max
 2      Odeon Cinema             100               2              21/05/2015      IMAX 3000
 3              IMAX             350               5              30/01/2015       IMAX Pro
15            Cinema             200               4              16/01/2015      Samsung i
```

It also updates like this in the database table.

This is what my database table looks like in phpMyAdmin.


Sort by key: None

+ Options

	id	name	numSeats	numExits	dateNextInspection	projectorType
Edit Copy Delete	1	Gaiety Cinema	250	3	11/01/2015	Sony Max
Edit Copy Delete	2	Odeon Cinema	100	2	21/05/2015	IMAX 3000
Edit Copy Delete	3	IMAX	350	5	30/01/2015	IMAX Pro
Edit Copy Delete	15	Cinema	200	4	16/01/2015	Samsung i

Check All / Uncheck All *With selected:* Change Delete Export