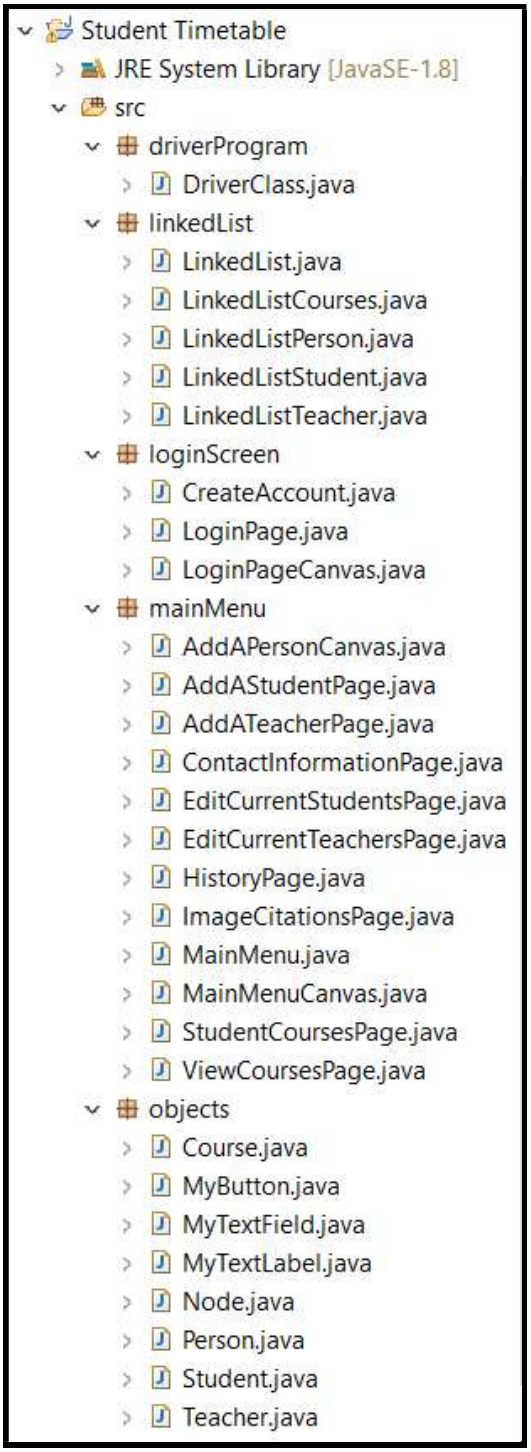
Appendix B: Source Code

For reference, here is the main structure of my program:



```java
package driverProgram;

/**
 * @author Austin Xu
 * High school student timetable program
 * IDE - Eclipse
 * Platform - PC
 */
import linkedList.*;
import loginScreen.*;
import mainMenu.*;
import objects.*;

public class DriverClass {
    private static LoginPage loginPage;
    private static LoginPageCanvas loginCanvas, createAccountCanvas;
    private static CreateAccount createAccountPage;

    private static MainMenu mainMenuPage;
    private static MainMenuCanvas mainMenuCanvas;

    private static AddAStudentPage addAStudent;
    private static AddAPersonCanvas addAStudentCanvas;

    private static EditCurrentStudentsPage editCurrentStudents;

    private static ViewCoursesPage viewCourses;

    private static AddATeacherPage addATeacher;
    private static AddAPersonCanvas addATeacherCanvas;

    private static EditCurrentTeachersPage editCurrentTeachers;

    private static ContactInformationPage contactInformationPage;

    private static HistoryPage historyPage;

    private static ImageCitationsPage imageCitationsPage;

    /**
     * The main method used in running this program
     * @param args Unused
     */
    public static void main(String[] args) {
        loginCanvas = new LoginPageCanvas();
        loginPage = new LoginPage(loginCanvas);

```

```java
        createAccountCanvas = new LoginPageCanvas();
        createAccountPage = new CreateAccount(createAccountCanvas);

        historyPage = new HistoryPage();

        mainMenuCanvas = new MainMenuCanvas();
        mainMenuPage = new MainMenu(mainMenuCanvas);

        addAStudentCanvas = new AddAPersonCanvas();
        addAStudent = new AddAStudentPage(addAStudentCanvas);

        editCurrentStudents = new EditCurrentStudentsPage();

        viewCourses = new ViewCoursesPage();

        addATeacherCanvas = new AddAPersonCanvas();
        addATeacher = new AddATeacherPage(addATeacherCanvas);

        editCurrentTeachers = new EditCurrentTeachersPage();

        contactInformationPage = new ContactInformationPage();

        imageCitationsPage = new ImageCitationsPage();
    }

    /**
     * Adds an event to the text area of the history page
     * @param str The string of the event being displayed in history page
     * @param addToRandomAccessFile "True" if the this event should be added to the "HistoryInfo" text file, "false" otherwise
     */
    public static void addEventToHistory(String str, boolean addToRandomAccessFile) {
        historyPage.addText(str, addToRandomAccessFile);
    }

    /**
     * Adds a student to the main menu page
     * @param stu A student
     */
    public static void addStudent(Student stu) {
        mainMenuPage.addStudent(stu);
    }

    /**
     * Add a teacher to the main menu page
     * @param tch A teacher
     */
    public static void addTeacher(Teacher tch) {
```

```
        mainMenuPage.addTeacher(tch);
    }

    /**
     * Clear the history from the history page
     * @param addToRandomAccessFile "True" if the history should be cleared from the "HistoryInfo" text file, "false" otherwise
     */
    public static void clearHistory(boolean addToRandomAccessFile) {
        historyPage.clearHistory(addToRandomAccessFile);
    }

    /**
     * Set the username and password for the timetable program
     * @param username The user's chosen username upon creating their account
     * @param password The user's chosen password upon creating their account
     */
    public static void createAccount(String username, String password) {
        loginPage.setUsernameAndPassword(username, password);
    }

    /**
     * Display the "Add A Student" page
     */
    public static void displayAddAStudentPage() {
        addAStudent.setVisible(true);
    }

    /**
     * Display the "Add A Teacher" page
     */
    public static void displayAddATeacherPage() {
        addATeacher.setVisible(true);
    }

    /**
     * Display the "Contact Information" page
     */
    public static void displayContactInformationPage() {
        contactInformationPage.setLists(mainMenuPage.getStudentsList(), mainMenuPage.getTeachersList());
        contactInformationPage.setVisible(true);
    }

    /**
     * Display the "Create an account" page
     */
    public static void displayCreateAccountPage() {
        createAccountPage.setVisible(true);
```

```
    }

    /**
     * Display the "Edit current students" page and set the list of students in the "Edit current students" page
     */
    public static void displayEditCurrentStudentsPage() {
        editCurrentStudents.setStudentsList(mainMenuPage.getStudentsList());
        editCurrentStudents.setVisible(true);
    }

    /**
     * Display the "Edit current teachers" page and set the list of teachers in the "Edit current teachers" page
     */
    public static void displayEditCurrentTeachersPage() {
        editCurrentTeachers.setTeachersList(mainMenuPage.getTeachersList());
        editCurrentTeachers.setVisible(true);
    }

    /**
     * Display the "History" page
     */
    public static void displayHistoryPage() {
        historyPage.setVisible(true);
    }

    /**
     * Display the "Image Citations" page
     */
    public static void displayImageCitationsPage() {
        imageCitationsPage.setVisible(true);
    }

    /**
     * Display the "Login Screen" page
     */
    public static void displayLoginPage() {
        loginPage.setVisible(true);
    }

    /**
     * Display the "Main Menu" page
     */
    public static void displayMainMenu() {
        mainMenuPage.setVisible(true);
        mainMenuPage.updateCnt();
    }

```

```java
    /**
     * Display the "View All Possible Courses Offered" page
     */
    public static void displayViewCoursesPage() {
        viewCourses.setVisible(true);
    }

    /**
     * Get the list of students from the main menu page
     * @return A list of students
     */
    public static LinkedListStudent getStudents() {
        return mainMenuPage.getStudentsList();
    }

    /**
     * Get the list of teachers from the main menu page
     * @return A list of teachers
     */
    public static LinkedListTeacher getTeachers() {
        return mainMenuPage.getTeachersList();
    }
}
```

```java
package objects;

import javax.swing.*;

// Custom class for allowing easier creation of buttons
public class MyButton extends JButton {
    private static final long serialVersionUID = 1L;

    /**
     * Initializes the values for a "MyButton" object
     * @param text The title of this MyButton
     */
    public MyButton (String text) {
        super (text, null);
    }

    /**
     * Set the width, height, and position of this MyButton
     * @param width The width of this MyButton
     * @param height The height of this MyButton
     * @param x The horizontal location of this MyButton
     * @param y The vertical location of this MyButton
     */
    public void setButton(int width, int height, int x, int y) {
        setSize(width, height);
        setLocation(x, y);
        setOpaque(true);
    }
}
```

```java
package objects;

import javax.swing.*;

// Custom class for allowing easier creation of text fields
public class MyTextField extends JTextField {
    private static final long serialVersionUID = 1L;

    /**
     * Set the width, height, and position of this MyTextField
     * @param width The width of this MyTextField
     * @param height The height of this MyTextField
     * @param x The horizontal location of this MyTextField
     * @param y The vertical location of this MyTextField
     */
    public void setTextField(int width, int height, int x, int y) {
        setSize(width, height);
        setLocation(x, y);
    }
}
```

```java
package objects;

import java.awt.*;
import javax.swing.*;

// Custom class for allowing easier creation of labels
public class MyTextLabel extends JLabel {
    private static final long serialVersionUID = 1L;

    /**
     * Initializes the values for a "MyTextLabel" object
     * @param text The text of this MyTextLabel
     * @param center The positioning of the text for this MyTextLabel
     */
    public MyTextLabel(String text, int center) {
        super (text, null, center);
    }

    /**
     * Set the width, height, position, foreground colour, background colour, and font for this MyTextLabel
     * @param width The width of this MyTextLabel
     * @param height The height of this MyTextLabel
     * @param x The horizontal location of this MyTextLabel
     * @param y The vertical location of this MyTextLabel
     * @param foregroundColour The foreground colour of this MyTextLabel
     * @param backgroundColour The background colour of this MyTextLabel
     * @param f The font of this MyTextLabel
     */
    public void setTextLabel(int width, int height, int x, int y, Color foregroundColour, Color backgroundColour, Font f) {
        setSize(width, height);
        setLocation(x, y);
        setForeground(foregroundColour);
        setBackground(backgroundColour);
        setFont(f);
        setOpaque(true);
    }
}
```

```java
package objects;

public class Node {
    private Node next, prev;

    /**
     * Initializes the values for a "Node" object
     */
    public Node() {
        next = prev = null;
    }

    /**
     * Set the next Node of this current Node
     * @param n The next Node of this current Node
     */
    public void setNext(Node n) {
        next = n;
    }

    /**
     * Get the next Node of this current Node
     * @return The next node of this current Node
     */
    public Node getNext() {
        return next;
    }

    /**
     * Set the previous Node of this current Node
     * @param n The previous Node of this current Node
     */
    public void setPrev(Node n) {
        prev = n;
    }

    /**
     * Get the previous node of this current Node
     * @return The previous Node of this current Node
     */
    public Node getPrev() {
        return prev;
    }
}
```

```java
package objects;

// This class provides a template for creating "Course" objects
public class Course extends Node {
    private String courseCode;

    /**
     * Initializes the values for a "Course" object
     * @param courseCode The course code for this Course
     */
    public Course(String courseCode) {
        super();
        this.courseCode = courseCode;
    }

    /**
     * Get the course code for this Course
     * @return The course code for this Course
     */
    public String getCourseCode() {
        return courseCode;
    }
}
```

```java
package objects;

// This class provides a template for creating "Person" objects
public class Person extends Node {
    private String name;   // Name of person
    private int id;   // ID of person

    /**
     * Initializes the values for a "Person" object
     */
    public Person() {
        super();
        name = "";
        id = -1;
    }

    /**
     * Initializes the values for a "Person" object
     * @param name The name of this Person
     * @param id The ID of this Person
     */
    public Person (String name, int id) {
        super();
        this.name = name;
        this.id = id;
    }

    /**
     * Set up the name for this Person
     * @param name The name of this Person
     */
    public void setName(String name) {
        this.name = name.trim();
    }

    /**
     * Set up the ID for this Person
     * @param id The ID of this Person
     */
    public void setID (int id) {
        this.id = id;
    }

    /**
     * Get the name of this Person
     * @return The name of this Person
     */
```

```java
    public String getName() {
        return name;
    }

    /**
     * Get the ID of this Person
     * @return The ID of this Person
     */
    public int getID() {
        return id;
    }

    /**
     * Randomly generate an ID number
     * @return A randomly generated ID number
     */
    public static int generateID() {
        String str = "";

        // ID's are 9 comprised of 9 numbers
        for (int i = 0; i < 9; i ++) {
            str += String.valueOf((int) (Math.random() * 10));
        }

        return(Integer.parseInt(str));
    }
}
```

```java
package objects;

import driverProgram.*;
import linkedList.*;
import mainMenu.*;

// This class provides a template for creating "Student" objects
public class Student extends Person {
    private int grade;   // Grade of student
    private boolean ib;   // Is "true" if student is in pre-IB or IB program, "false" otherwise
    private String email;   // The student's school email

    private StudentCoursesPage coursesPage;   // This object contains the completed & current courses of the student

    private String[] timetable;   // Timetable of the student

    /**
     * Initializes the values for a "Student" object
     */
    public Student() {
        super();
        grade = 0;
        ib = false;

        /* Timetable of student has length 8 because student must have 8 courses in a school year.
         * In this program, a spare period is counted as a course as well, for simplicity. */
        timetable = new String[8];

        coursesPage = new StudentCoursesPage("Edit/View Student's Courses", this);
    }

    /**
     * Display this Student's completed and current courses
     * @param showCompletedCourses "True" if the user wants to show the page of this Student's completed courses, "false" if the user wants to show the page of this Student's current courses
     */
    public void displayCoursesPage(boolean showCompletedCourses) {
        coursesPage.setBool(showCompletedCourses);
        coursesPage.setPage();
        coursesPage.setVisible(true);
    }

    /**
     * Set the timetable for this Student
     * @param timeTbl The timetable of this Student
     */
    public void setTimetable(String[] timeTbl) {
```

```java
        timetable = timeTbl;
    }

    /**
     * Get the timetable of this Student
     * @return The timetable of this Student
     */
    public String[] getTimetable() {
        return timetable;
    }

    /**
     * Set the grade for this Student
     * @param grade The grade of this Student
     */
    public void setGrade(int grade) {
        this.grade = grade;
    }

    /**
     * Get the grade of this Student
     * @return The grade of this Student
     */
    public int getGrade() {
        return grade;
    }

    /**
     * Set whether or not this Student is in the pre-IB or IB program
     * @param b "True" if this Student is in the pre-IB or IB program, "false" otherwise
     */
    public void setIB(boolean b) {
        ib = b;
    }

    /**
     * Check if this Student is in the pre-IB or IB program
     * @return "True" if this Student is in the pre-IB or IB program, "false" otherwise
     */
    public boolean isInIB() {
        return ib;
    }

    /**
     * Set the linked lists for this Student's completed and current courses
     */
    public void setLists() {
```

```
        coursesPage.setLists();
    }

    /**
     * Get the 2D array representing this Student's completed courses
     * @return The 2D array representing this Student's completed courses
     */
    public boolean[][] getCompletedCourses() {
        return coursesPage.getCompletedCourses();
    }

    /**
     * Set this Student's completed courses 2D array
     * @param c A linked list of courses which is used to set this Student's completed courses 2D array
     */
    public void setCompletedCourses(LinkedListCourses c) {
        coursesPage.setCompletedCourses(c);
    }

    /**
     * Get the 2D array representing this Student's current courses
     * @return The 2D array representing this Student's current courses
     */
    public boolean[][] getCurrentCourses() {
        return coursesPage.getCurrentCourses();
    }

    /**
     * Set this Student's current courses 2D array
     * @param c A linked list of courses which is used to set this Student's current courses 2D array
     */
    public void setCurrentCourses(LinkedListCourses c) {
        coursesPage.setCurrentCourses(c);
    }

    /**
     * Get the list of this Student's completed courses
     * @return The list of this Student's completed courses
     */
    public LinkedListCourses getCompletedCoursesList() {
        return coursesPage.getCompletedCoursesList();
    }

    /**
     * Get the list of this Student's current courses
     * @return The list of this Student's current courses
     */
```

```java
    public LinkedListCourses getCurrentCoursesList() {
        return coursesPage.getCurrentCoursesList();
    }

    /**
     * Generate an email for this Student.
     * Emails for Students with the same full name as other Students are distinguished from each other by numbers.
     */
    public void generateEmail() {
        LinkedListStudent list = DriverClass.getStudents();

        int cnt = 1;   // "cnt" represents the number of unique current students having the name
        for (int i = 0; i < list.getSize(); i ++) {
            if (list.get(0, list.getHead(), i).getName().equalsIgnoreCase(super.getName())) {
                cnt ++;
            }
        }

        /* fullName[0] represents student's first name, fullName[fullName.length - 1] represents student's last name.
         * Anything between fullName[0] and fullName[fullName.length - 1] is the middle name of the student. */
        String[] fullName = super.getName().split(" ");

        if (cnt == 1) email = fullName[0].toLowerCase() + "." + fullName[fullName.length - 1].toLowerCase() + "@student.tdsb.on.ca";
        else email = fullName[0].toLowerCase() + "." + fullName[fullName.length - 1].toLowerCase() + cnt + "@student.tdsb.on.ca";
    }

    /**
     * Set this Student's email
     * @param email The email of this Student
     */
    public void setEmail(String email) {
        this.email = email;
    }

    /**
     * Get this Student's email
     * @return This Student's email
     */
    public String getEmail() {
        return email;
    }
}
```

```java
package objects;

import driverProgram.*;
import linkedList.*;

// This class provides a template for creating "Teacher" objects
public class Teacher extends Person {
    String email;   // Teacher's school email

    /**
     * Initializes the values for a "Teacher" object
     * @param name The name of this Teacher
     * @param id The ID of this Teacher
     */
    public Teacher(String name, int id) {
        super (name, id);
    }

    /**
     * Generate an email for this Teacher.
     * Emails for Teacher with the same full name as other Teacher are distinguished from each other by numbers.
     */
    public void generateEmail() {
        LinkedListTeacher list = DriverClass.getTeachers();

        int cnt = 1;   // "cnt" represents the number of unique current teachers having the name
        for (int i = 0; i < list.getSize(); i ++) {
            if (list.get(0, list.getHead(), i).getName().equalsIgnoreCase(super.getName())) {
                cnt ++;
            }
        }

        /* fullName[0] represents teacher's first name, fullName[fullName.length - 1] represents teacher's last name.
         * Anything between fullName[0] and fullName[fullName.length - 1] is the middle name of the teacher. */
        String[] fullName = super.getName().split(" ");

        if (cnt == 1) email = fullName[0].toLowerCase() + "." + fullName[fullName.length - 1].toLowerCase() + "@tdsb.on.ca";
        else email = fullName[0].toLowerCase() + "." + fullName[fullName.length - 1].toLowerCase() + cnt + "@tdsb.on.ca";
    }

    /**
     * Set this Teacher's email
     * @param email This Teacher's email
     */
    public void setEmail(String email) {
        this.email = email;
    }
```

```java

    /**
     * Get this Teacher's email
     * @return This Teacher's email
     */
    public String getEmail() {
        return email;
    }
}
```

```java
package linkedList;

import objects.*;

// My doubly linked list class
public class LinkedList {
    private Node head, tail;
    private int size;   // Size of linked list

    /**
     * Initializes the "head", "tail", and "size" values of the linked list
     */
    public LinkedList() {
        head = tail = null;
        size = 0;
    }

    /**
     * Get the head of the linked list
     * @return The head of the linked list
     */
    public Node getHead() {
        return head;
    }

    /**
     * Set the head of the linked list
     * @param n A Node which the head is assigned to
     */
    public void setHead(Node n) {
        head = n;
    }

    /**
     * Get the tail of the linked list
     * @return The tail of the linked list
     */
    public Node getTail() {
        return tail;
    }

    /**
     * Set the tail of the linked list
     * @param n A Node which the tail is assigned to
     */
    public void setTail(Node n) {
        tail = n;
```

```java
    }

    /**
     * Add a Node to the front of the linked list
     * @param n A Node to be added to the front of the linked list
     */
    public void addFirst(Node n) {
        if (head == null) {
            head = tail = n;
            n.setNext(null);
            n.setPrev(null);
        } else {
            n.setNext(head);
            n.setPrev(null);
            head.setPrev(n);
            head = n;
        }
        size ++;
    }

    /**
     * Add a Node to the end of the linked list
     * @param n A Node to be added to the end of the linked list
     */
    public void addLast(Node n) {
        if (tail == null) {
            head = tail = n;
            n.setNext(null);
            n.setPrev(null);
        } else {
            tail.setNext(n);
            n.setPrev(tail);
            n.setNext(null);
            tail = n;
        }
        size ++;
    }

    /**
     * Remove and return the first Node of the linked list
     * @return The first Node of the linked list
     */
    public Node removeFirst() {
        Node n = head;
        head = head.getNext();
        n.setNext(null);
        head.setPrev(null);
```

```java
        size --;
        return n;
    }

    /**
     * Remove and return the last Node of the linked list
     * @return The last Node of the linked list
     */
    public Node removeLast() {
        Node n = tail;
        tail = tail.getPrev();
        n.setPrev(null);
        tail.setNext(null);
        size --;
        return n;
    }

    /**
     * Remove a particular Node from the linked list
     * @param n The node who is to be removed from the linked list
     * @return A particular Node from the linked list
     */
    public Node remove(Node n) {
        if (n.getPrev() == null && n.getNext() == null) {
            head = tail = null;
            size --;
            return n;
        }
        else if (n.getPrev() == null) return(removeFirst());
        else if (n.getNext() == null) return(removeLast());
        else {
            Node prev = n.getPrev(), next = n.getNext();
            n.setPrev(null);
            n.setNext(null);
            prev.setNext(next);
            next.setPrev(prev);
            size --;
            return n;
        }
    }

    /**
     * Recursive algorithm for retrieving the Node associated with a current index in the linked list
     * @param curIdx The current index of this recursive algorithm
     * @param curNode The current node of this recursive algorithm
     * @param target The index of the element in this linked which which we want to retrieve
     * @return The element in this linked list which we want to retrieve
```

```java
     */
    public Node get(int curIdx, Node curNode, int target) {
        if (curIdx == target) return curNode;
        return get(curIdx + 1, curNode.getNext(), target);
    }

    /**
     * Get the size of the linked list
     * @return The size of the linked list
     */
    public int getSize() {
        return size;
    }
}
```

```java
package linkedList;

import objects.*;

public class LinkedListCourses extends LinkedList {

    /**
     * Initializes the values for a "LinkedListCourses" object
     */
    public LinkedListCourses() {
        super();
    }

    /**
     * Get the head of the linked list
     * @return The head of the linked list
     */
    public Course getHead() {
        return (Course) super.getHead();
    }

    /**
     * Get the tail of the linked list
     * @return The tail of the linked list
     */
    public Course getTail() {
        return (Course) super.getTail();
    }

    /**
     * Add a Course to the front of the linked list
     * @param c A Course to be added to the front of the linked list
     */
    public void addFirst(Course c) {
        super.addFirst(c);
    }

    /**
     * Add a Course to the end of the linked list
     * @param c A Course to be added to the end of the linked list
     */
    public void addLast(Course c) {
        super.addLast(c);
    }

    /**
     * Remove and return the first Course of the linked list
```

```java
     * @return The first Course of the linked list
     */
    public Course removeFirst() {
        return (Course) super.removeFirst();
    }

    /**
     * Remove and return the last Course of the linked list
     * @return The last Course of the linked list
     */
    public Course removeLast() {
        return (Course) super.removeLast();
    }

    /**
     * Remove a particular Course from the linked list
     * @param c A Course to be removed from the linked list
     * @return The removed Course from the linked list
     */
    public Course remove(Course c) {
        return (Course) super.remove(c);
    }

    /**
     * Retrieve the Course associated with a particular index in the linked list
     * @return A particular Course from the linked list
     */
    public Course get(int curIdx, Node curNode, int target) {
        return (Course) super.get(curIdx, curNode, target);
    }

    /**
     * Check if the linked list contains a particular Course
     * @param s A course code which is to be checked if it is contained in the list
     * @return True if 's' is in the linked list, false otherwise
     */
    public boolean contains (String s) {  // Check if the list contains a certain course
        Course cur = (Course) super.getHead();

        for (int i = 0; i < super.getSize(); i ++) {
            if (cur.getCourseCode().equals(s)) return true;
            cur = (Course) cur.getNext();
        }

        return false;
    }
}
```

```java
package linkedList;

import objects.*;

public class LinkedListPerson extends LinkedList {

    /* The "originalUnsortedList" array contains the list of people in the order that they were added in.
     * The "alphabeticallySortedList" array contains the list of people sorted in alphabetical order. */
    private Person[] originalUnsortedList, alphabeticallySortedList;

    private boolean sorted;   // Variable that keeps track if the current list is sorted alphabetically or not

    /**
     * Initializes the values for a "LinkedListPerson" object
     */
    public LinkedListPerson() {
        super();
        sorted = false;
        originalUnsortedList = new Person[super.getSize()];
        alphabeticallySortedList = new Person[super.getSize()];
    }

    /**
     * Get the head of the linked list
     * @return The head of the linked list
     */
    public Person getHead() {
        return (Person) super.getHead();
    }

    /**
     * Get the tail of the linked list
     * @return The tail of the linked list
     */
    public Person getTail() {
        return (Person) super.getTail();
    }

    /**
     * Add a Person to the front of the linked list
     * @param p A Person to be added to the front of the linked list
     */
    public void addFirst(Person p) {
        super.addFirst(p);
    }

    /**
```

```java
     * Add a Person to the end of the linked list
     * @param p A Person to be added to the end of the linked list
     */
    public void addLast(Person p) {
        super.addLast(p);
    }

    /**
     * Remove and return the first Person of the linked list
     * @return The first Person of the linked list
     */
    public Person removeFirst() {
        return (Person) super.removeFirst();
    }

    /**
     * Remove and return the last Person of the linked list
     * @return The last Person of the linked list
     */
    public Person removeLast() {
        return (Person) super.removeLast();
    }

    /**
     * Remove a particular Person from the linked list
     * @param p A person to be removed from the linked list
     * @return The removed Person from the linked list
     */
    public Person remove(Person p) {
        return (Person) super.remove(p);
    }

    /**
     * Retrieve the Person associated with a particular index in the linked list
     * @return A particular Person from the linked list
     */
    public Person get(int curIdx, Node curNode, int target) {
        return (Person) super.get(curIdx, curNode, target);
    }

    /**
     * Checks if the linked list of people is currently sorted alphabetically or not
     * @return True if the linked list of people is sorted alphabetically, false otherwise
     */
    public boolean isSorted() {
        return sorted;
    }
```

```java

    /**
     * Sort the linked list of people alphabetically
     * @param flag "True" if the user has chosen to sort the linked list of people alphabetically, "false" if the user has chosen to unsort the list of people back into its original order (i.e. sort the list of people by date added)
     */
    public void sortAlphabetically(boolean flag) {
        int size = super.getSize();  // Size of linked list

        // If the user has chosen to sort the list alphabetically
        if (flag) {
            alphabeticallySortedList = new Person[super.getSize()];   // Array used to store the alphabetically sorted list of people

            boolean[] vis = new boolean[size];  // Stores which indices of "originalUnsortedList" have been assigned to the "alphabeticallySortedList"

            // My sorting algorithm for sorting the list of people alphabetically.
            for (int i = 0; i < size; i ++) {
                int minIdx = -1;  // "minIdx" stores the index of an element in "originalUnsortedList" which has "!vis[index]" and is the smallest lexicographically

                for (int j = 0; j < size; j ++) {  // Looping from the indices of "originalUnsortedList"
                    if (!vis[j]) {  // "vis[j]" is "false" if "originalUnsortedList[j]" is not in the "alphabeticallySortedList" yet
                        if (minIdx == -1) {  // If the default value for "minIdx" is used, update it to 'j'
                            minIdx = j;
                        } else {  // Compare the alphabetic values of "originalUnsortedList[minIdx]" with "originalUnsortedList[j]"
                            String minName = originalUnsortedList[minIdx].getName();
                            String currentName = originalUnsortedList[j].getName();

                            // If "originalUnsortedList[j]" is lexicographically smaller than "originalUnsortedList[minIdx]", update the value of "minIdx"
                            if (currentName.compareTo(minName) < 0) minIdx = j;
                        }
                    }
                }

                alphabeticallySortedList[i] = originalUnsortedList[minIdx];  // Set the 'i-th' smallest element in "alphabeticallySortedList"
                vis[minIdx] = true;  // Mark "vis[minIdx]" as true, so "minIdx" can no longer be analyzed from the "originalUnsortedList" in future iterations
            }

            // Since this sorting algorithm deals with objects, it is necessary to reset the pointers of the "originalUnsortedList"
            for (int i = 0; i < size; i ++) {
                originalUnsortedList[i].setPrev(null);
                originalUnsortedList[i].setNext(null);
            }

```

```java
            // Setting up the list based on the sorted "alphabeticallySortedList" array
            for (int i = 0; i < size; i ++) {
                if (size == 1) {
                    super.setHead(alphabeticallySortedList[i]);
                    super.setTail(alphabeticallySortedList[i]);
                } else if (i == 0) {
                    super.setHead(alphabeticallySortedList[i]);
                } else if (i == size - 1) {
                    super.setTail(alphabeticallySortedList[i]);
                    alphabeticallySortedList[i - 1].setNext(alphabeticallySortedList[i]);
                    alphabeticallySortedList[i].setPrev(alphabeticallySortedList[i - 1]);
                } else {
                    alphabeticallySortedList[i].setPrev(alphabeticallySortedList[i - 1]);
                    alphabeticallySortedList[i - 1].setNext(alphabeticallySortedList[i]);
                }
            }

            sorted = true;   // Indicates that the list is currently sorted alphabetically
        }

        // If the user has chosen to unsort the list back into its original order.
        else {
            if (alphabeticallySortedList.length == 0) return;

            for (int i = 0; i < size; i ++) {
                alphabeticallySortedList[i].setPrev(null);
                alphabeticallySortedList[i].setNext(null);
            }

            // Setting up the list based on the original ordering of the list (i.e. sorting the list in terms of date added)
            for (int i = 0; i < size; i ++) {
                if (size == 1) {
                    super.setHead(originalUnsortedList[i]);
                    super.setTail(originalUnsortedList[i]);
                } else if (i == 0) {
                    super.setHead(originalUnsortedList[i]);
                } else if (i == size - 1) {
                    super.setTail(originalUnsortedList[i]);
                    originalUnsortedList[i - 1].setNext(originalUnsortedList[i]);
                    originalUnsortedList[i].setPrev(originalUnsortedList[i - 1]);
                } else {
                    originalUnsortedList[i].setPrev(originalUnsortedList[i - 1]);
                    originalUnsortedList[i - 1].setNext(originalUnsortedList[i]);
                }
            }

            sorted = false;   // Indicates that the list is currently not sorted alphabetically
```

```java
        }
    }

    /**
     * Setting up the original unsorted list of people, or, in other words, setting up the list of people by date added
     */
    public void setOriginalList() {
        originalUnsortedList = new Person[super.getSize()];

        // Copy elements of list into the "originalUnsortedList" array.
        int idx = 0;
        Person cur = (Person) super.getHead();
        while (cur != null) {
            originalUnsortedList[idx] = cur;
            cur = (Person) cur.getNext();
            idx ++;
        }
    }

    /**
     * Update the original unsorted list to accommodate for removing a person from the list
     * @param removed A person which was removed from the list
     */
    public void updateOriginalList(Person removed) {
        Person[] temp = new Person[super.getSize()];

        // Copying all elements from "originalUnsortedList" array to "temp" array excluding the person that has been removed
        for (int i = 0, j = 0; i < super.getSize(); i ++, j ++) {
            if (originalUnsortedList[j] != removed) {
                temp[i] = originalUnsortedList[j];
            } else {
                j ++;
                temp[i] = originalUnsortedList[j];
            }
        }

        originalUnsortedList = temp;   // The original list of people have been updated
    }
}
```

```java
package linkedList;

import objects.*;

public class LinkedListStudent extends LinkedListPerson {

    /**
     * Initializes the values for a "LinkedListStudent" object
     */
    public LinkedListStudent() {
        super();
    }

    /**
     * Get the head of the linked list
     * @return The head of the linked list
     */
    public Student getHead() {
        return (Student) super.getHead();
    }

    /**
     * Get the tail of the linked list
     * @return The tail of the linked list
     */
    public Student getTail() {
        return (Student) super.getTail();
    }

    /**
     * Add a Student to the front of the linked list
     * @param st A Student to be added to the front of the linked list
     */
    public void addFirst(Student st) {
        super.addFirst(st);
    }

    /**
     * Add a Student to the end of the linked list
     * @param st A Student to be added to the end of the linked list
     */
    public void addLast(Student st) {
        super.addLast(st);
    }

    /**
     * Remove and return the first Student of the linked list
```

```java
     * @return The first Student of the linked list
     */
    public Student removeFirst() {
        return (Student) super.removeFirst();
    }

    /**
     * Remove and return the last Student of the linked list
     * @return The last Student of the linked list
     */
    public Student removeLast() {
        return (Student) super.removeLast();
    }

    /**
     * Remove a particular Student from the linked list
     * @param st A Student to be removed from the linked list
     * @return The removed Student from the linked list
     */
    public Student remove(Student st) {
        return (Student) super.remove(st);
    }

    /**
     * Retrieve the Student associated with a particular index in the linked list
     * @return A particular Student from the linked list
     */
    public Student get(int curIdx, Node curNode, int target) {
        return (Student) super.get(curIdx, curNode, target);
    }

    /**
     * Sort the linked list of students alphabetically
     */
    public void sortAlphabetically(boolean flag) {
        super.sortAlphabetically(flag);
    }
}
```

```java
package linkedList;

import objects.*;

public class LinkedListTeacher extends LinkedListPerson {

    /**
     * Initializes the values for a "LinkedListStudent" object
     */
    public LinkedListTeacher() {
        super();
    }

    /**
     * Get the head of the linked list
     * @return The head of the linked list
     */
    public Teacher getHead() {
        return (Teacher) super.getHead();
    }

    /**
     * Get the tail of the linked list
     * @return The tail of the linked list
     */
    public Teacher getTail() {
        return (Teacher) super.getTail();
    }

    /**
     * Add a Teacher to the front of the linked list
     * @param tch A Teacher to be added to the front of the linked list
     */
    public void addFirst(Teacher tch) {
        super.addFirst(tch);
    }

    /**
     * Add a Teacher to the end of the linked list
     * @param tch A Teacher to be added to the end of the linked list
     */
    public void addLast(Teacher tch) {
        super.addLast(tch);
    }

    /**
     * Remove and return the first Teacher of the linked list
```

```java
     * @return The first Teacher of the linked list
     */
    public Teacher removeFirst() {
        return (Teacher) super.removeFirst();
    }

    /**
     * Remove and return the last Teacher of the linked list
     * @return The last Teacher of the linked list
     */
    public Teacher removeLast() {
        return (Teacher) super.removeLast();
    }

    /**
     * Remove a particular Teacher from the linked list
     * @param tch A Teacher to be removed from the linked list
     * @return The removed Teacher from the linked list
     */
    public Teacher remove(Teacher tch) {
        return (Teacher) super.remove(tch);
    }

    /**
     * Retrieve the Teacher associated with a particular index in the linked list
     * @return A particular Teacher from the linked list
     */
    public Teacher get(int curIdx, Node curNode, int target) {
        return (Teacher) super.get(curIdx, curNode, target);
    }
}
```

```java
package loginScreen;

import javax.imageio.*;
import javax.swing.*;
import java.awt.*;
import java.awt.image.*;
import java.io.*;

// Canvas for the "Login" page
public class LoginPageCanvas extends JPanel {
    private static final long serialVersionUID = 1L;

    private BufferedImage background;

    /**
     * Initializes the values for a "LoginPageCanvas" object
     */
    public LoginPageCanvas() {
        try {
            background = ImageIO.read(this.getClass().getResource("/loginScreenBackground.jpg"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        setLayout(null);
    }

    /**
     * Setting up the background image for this canvas
     */
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(background, 0, 0, getWidth(), getHeight(), null);
    }
}
```

```java
package loginScreen;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import driverProgram.*;
import objects.*;

public class LoginPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private MyTextLabel welcomeMessageLabel, usernameLabel, passwordLabel;
    private MyTextField usernameTextField, passwordTextField;
    private MyButton createAccountButton, loginButton, quitButton;

    private long usernameHashcode, passwordHashcode;  // Hashcode values for the username and password that user enters

    private RandomAccessFile loginInfo;

    /**
     * Initializes the values for a "LoginPage" object
     * @param canvas The canvas for the "Login" page
     */
    public LoginPage(JPanel canvas) {
        super ("Login Screen");
        add(canvas);

        /* Recalling the username and password set by the user once they created an account.
         * This information is recalled from the "LoginInfo.txt" RandomAccessFile. */
        try {
            loginInfo = new RandomAccessFile("LoginInfo.txt", "rw");
            try {
                String username = loginInfo.readLine();
                String password = loginInfo.readLine();

                if (username == null) {
                    usernameHashcode = 0;
                    passwordHashcode = 0;
                } else {
                    usernameHashcode = Long.parseLong(username);
                    passwordHashcode = Long.parseLong(password);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
```

```
            e.printStackTrace();
        }

        setSize(1200, 820);
        setVisible(true);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        welcomeMessageLabel = new MyTextLabel("Welcome to the High School Student Timetable", SwingConstants.CENTER);
        welcomeMessageLabel.setTextLabel(800, 150, 200, 50, Color.BLACK, Color.GREEN, new Font("Arial", Font.BOLD, 35));

        usernameLabel = new MyTextLabel("Please enter your username:", SwingConstants.CENTER);
        usernameLabel.setTextLabel(260, 20, 450, 330, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 18));

        usernameTextField = new MyTextField();
        usernameTextField.setTextField(500, 40, 450, 350);

        passwordLabel = new MyTextLabel("Please enter your password:", SwingConstants.CENTER);
        passwordLabel.setTextLabel(260, 20, 450, 430, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 18));

        passwordTextField = new MyTextField();
        passwordTextField.setTextField(500, 40, 450, 450);

        createAccountButton = new MyButton("Create an account");
        createAccountButton.setButton(150, 50, 450, 600);
        createAccountButton.addActionListener(this);

        loginButton = new MyButton("Login");
        loginButton.setButton(150, 50, 450, 500);
        loginButton.addActionListener(this);

        quitButton = new MyButton("Quit");
        quitButton.setButton(150, 70, 0, 715);
        quitButton.addActionListener(this);

        canvas.add(loginButton);
        canvas.add(welcomeMessageLabel);

        canvas.add(usernameLabel);
        canvas.add(usernameTextField);

        canvas.add(passwordLabel);
        canvas.add(passwordTextField);

        canvas.add(createAccountButton);
        canvas.add(quitButton);
```

```
    }

    public void paint(Graphics g) {
        super.paintComponents(g);
        repaint();
    }

    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();

        if (button.equals(quitButton)) {  // User has chosen to exit the program
            System.exit(0);
        } else if (button.equals(createAccountButton)) {  // User has chosen to create an account
            if (usernameHashcode != 0 && passwordHashcode != 0) {
                JOptionPane.showMessageDialog(this, "You have already created an account. Another account cannot be created.", "Warning", JOptionPane.WARNING_MESSAGE);
            } else {
                setVisible(false);
                DriverClass.displayCreateAccountPage();
            }
        } else if (button.equals(loginButton)) {  // User has chosen to try logging in
            String tempUsername = usernameTextField.getText().toLowerCase().trim();
            String tempPassword = passwordTextField.getText().toLowerCase().trim();

            long tempUsernameHashcode = generateHashcode(tempUsername);
            long tempPasswordHashcode = generateHashcode(tempPassword);

            if (tempUsernameHashcode != 0 && tempPasswordHashcode != 0 && tempUsernameHashcode == usernameHashcode && tempPasswordHashcode == passwordHashcode) {
                setVisible(false);
                DriverClass.displayMainMenu();
            } else {
                JOptionPane.showMessageDialog(this, "The username or password that you entered is incorrect.", "Warning", JOptionPane.WARNING_MESSAGE);
            }
        }
    }

    /**
     * Method for generating the hashcode for a string
     * @param str The string whose hashcode is going to be generated
     * @return The hashcode of the given string "str"
     */
    public long generateHashcode(String str) {
        int n = str.length();
        long hshCode = 0;

```

```java
        /*
         * A prime number is chosen as the seed because a prime number is coprime to all other numbers.
         * When overflow happens, since 131 is coprime to Long.MAX_VALUE, the generated hashcode will be secure by evenly distributing the numbers.
         */
        int seed = 131;

        for (int i = 1; i <= n; i ++) {
            hshCode = hshCode * seed + str.charAt(i - 1);
        }

        return hshCode;
    }

    /**
     * Sets the username and password of the user for this program
     * @param username The username set by the user for this program
     * @param password The password set by the user for this program
     */
    public void setUsernameAndPassword(String username, String password) {
        usernameHashcode = generateHashcode(username);
        passwordHashcode = generateHashcode(password);

        // Saving the user's login info
        try {
            loginInfo.seek(0);
            loginInfo.writeBytes(usernameHashcode + "\n" + passwordHashcode);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
package loginScreen;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import driverProgram.*;
import objects.*;

public class CreateAccount extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private MyTextLabel createAccountMessageLabel, usernameLabel, passwordLabel;
    private MyTextField usernameTextField, passwordTextField;
    private MyButton backToLoginScreenButton, createAccountButton;

    /**
     * Initializes the values for a "CreateAccount" object
     * @param canvas The canvas for the "Create an account" page
     */
    public CreateAccount(JPanel canvas) {
        super ("Create an account page");
        add(canvas);

        setSize(800, 800);
        setVisible(false);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        createAccountMessageLabel = new MyTextLabel("Create your account here", SwingConstants.CENTER);
        createAccountMessageLabel.setTextLabel(400, 70, 200, 80, Color.BLACK, Color.GREEN, new Font("Arial", Font.BOLD, 20));

        usernameLabel = new MyTextLabel("Username:", SwingConstants.CENTER);
        usernameLabel.setTextLabel(260, 20, 150, 330, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 18));

        usernameTextField = new MyTextField();
        usernameTextField.setTextField(500, 40, 150, 350);

        passwordLabel = new MyTextLabel("Password:", SwingConstants.CENTER);
        passwordLabel.setTextLabel(260, 20, 150, 430, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 18));

        passwordTextField = new MyTextField();
        passwordTextField.setTextField(500, 40, 150, 450);

        backToLoginScreenButton = new MyButton("Back");
        backToLoginScreenButton.setButton(150, 50, 0, 715);
        backToLoginScreenButton.addActionListener(this);
```

```java

        createAccountButton = new MyButton("Create account");
        createAccountButton.setButton(150, 50, 150, 550);
        createAccountButton.addActionListener(this);

        canvas.add(createAccountMessageLabel);

        canvas.add(usernameLabel);
        canvas.add(usernameTextField);

        canvas.add(passwordLabel);
        canvas.add(passwordTextField);

        canvas.add(backToLoginScreenButton);
        canvas.add(createAccountButton);
    }

    public void paint(Graphics g) {
        super.paintComponents(g);
        repaint();
    }

    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();

        if (button.equals(backToLoginScreenButton)) {  // User has chosen to go back to the login screen
            setVisible(false);
            DriverClass.displayLoginPage();
        } else if (button.equals(createAccountButton)) {  // User has decided to create an account
            String username = usernameTextField.getText().toLowerCase().trim();
            String password = passwordTextField.getText().toLowerCase().trim();

            if (username.equals("") || password.equals("")) {
                JOptionPane.showMessageDialog(this, "You must enter a username and a password.", "Warning", JOptionPane.WARNING_MESSAGE);
            }
            else {
                usernameTextField.setText("");
                passwordTextField.setText("");

                DriverClass.createAccount(username, password);

                setVisible(false);
                DriverClass.displayLoginPage();

                JOptionPane.showMessageDialog(this, "An account has successfuly been created.");
            }
        }
```

```
    }
}
```

```java
package mainMenu;

import javax.imageio.*;
import javax.swing.*;
import java.awt.*;
import java.awt.image.*;
import java.io.*;

// Canvas for the "Main Menu" page
public class MainMenuCanvas extends JPanel {
    private static final long serialVersionUID = 1L;

    private BufferedImage background;

    public MainMenuCanvas() {
        try {
            background = ImageIO.read(this.getClass().getResource("/mainMenuBackground.jpg"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        setLayout(null);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(background, 0, 0, getWidth(), getHeight(), null);
    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import driverProgram.*;
import linkedList.*;
import objects.*;

public class MainMenu extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private static MyTextLabel infoLabel, currentStudentsCntLabel, currentTeachersCntLabel, teachersNeededLabel;

    private MyButton backToLoginScreenButton, quitButton;
    private MyButton addStudentButton, editCurrentStudentsButton, addTeacherButton, editCurrentTeachersButton, coursesCodeButton, contactInformationButton, historyButton;
    private MyButton imageCitationsButton;

    private LinkedListStudent students;   // Linked list of current students in the database
    private LinkedListTeacher teachers;   // Linked list of current teachers in the database

    private JPanel canvas;

    /**
     * Initializes the values for a "MainMenu" object
     * @param c The canvas for this "Main Menu" page
     */
    public MainMenu(JPanel c) {
        super("Main Menu");

        canvas = c;
        add(canvas);

        setSize(1200, 820);
        setVisible(false);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        students = new LinkedListStudent();
        teachers = new LinkedListTeacher();

        /*
         *  Reloading the past information uploaded by the user into the database.
         *  This information is saved inside the "StudentsInfo.txt", "TeachersInfo.txt" and "HistoryInfo.txt" RandomAccessFiles.
```

```java
         *  This information consists of the student's name, completed courses, current courses, teacher's name, history of actions, etc.
         */
        try {
            // Reloading the past information uploaded by the user about the students in the database
            RandomAccessFile studentsInfo = new RandomAccessFile("StudentsInfo.txt", "rw");
            studentsInfo.seek(0);
            String line = "x";
            while (true) {
                line = studentsInfo.readLine();
                if (line == null) break;

                String[] parts = line.split("_");

                if (parts[0].equals("Add")) {
                    Student temp = new Student();
                    temp.setName(parts[1]);
                    temp.setID(Integer.parseInt(parts[2]));
                    temp.setGrade(Integer.parseInt(parts[3]));
                    if (parts[4].equals("true")) temp.setIB(true);
                    else temp.setIB(false);
                    temp.setEmail(parts[5]);

                    students.addLast(temp);
                } else if (parts[0].equals("Edit")) {
                    Student stu = findStudent(Integer.parseInt(parts[2]));

                    LinkedListCourses courses = new LinkedListCourses();
                    for (int i = 3; i < parts.length; i ++) {
                        courses.addLast(new Course(parts[i]));
                    }

                    if (parts[1].equals("Completed Courses")) {
                        stu.setCompletedCourses(courses);
                    } else {
                        stu.setCurrentCourses(courses);
                    }
                } else if (parts[0].equals("Remove")) {
                    Student stu = findStudent(Integer.parseInt(parts[1]));
                    students.remove(stu);
                } else if (parts[0].equals("Timetable")) {
                    Student stu = findStudent(Integer.parseInt(parts[1]));

                    String[] timetable = new String[8];
                    for (int i = 2; i <= 9; i ++) timetable[i - 2] = parts[i];
                    stu.setTimetable(timetable);
                }
            }
```

```
            studentsInfo.close();

            // Reloading the past information uploaded by the user about the teachers in the database
            RandomAccessFile teachersInfo = new RandomAccessFile("TeachersInfo.txt", "rw");
            teachersInfo.seek(0);
            line = "x";
            while (true) {
                line = teachersInfo.readLine();
                if (line == null) break;

                String[] parts = line.split("_");

                if (parts[0].equals("Add")) {
                    Teacher temp = new Teacher(parts[1], Integer.parseInt(parts[2]));
                    temp.setEmail(parts[3]);

                    teachers.addLast(temp);
                } else if (parts[0].equals("Remove")) {
                    Teacher tch = findTeacher(Integer.parseInt(parts[1]));
                    teachers.remove(tch);
                }
            }
            teachersInfo.close();

            // Reloading the history of events that have occurred in the database
            RandomAccessFile historyInfo = new RandomAccessFile("HistoryInfo.txt", "rw");
            historyInfo.seek(0);
            line = "x";
            while (true) {
                line = historyInfo.readLine();
                if (line == null) break;

                DriverClass.addEventToHistory(line, false);
            }
            historyInfo.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
        }

        infoLabel = new MyTextLabel("Main Menu", SwingConstants.CENTER);
        infoLabel.setTextLabel(300, 40, getWidth() / 2 - 150, 0, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 18));

        currentStudentsCntLabel = new MyTextLabel("Current Number of Students: " + students.getSize(), SwingConstants.CENTER);
        currentStudentsCntLabel.setTextLabel(500, 50, getWidth() / 2 - 250, 300, Color.BLUE, null, new Font("Arial", Font.BOLD, 30));
        currentStudentsCntLabel.setOpaque(false);
```

```java

        currentTeachersCntLabel = new MyTextLabel("Current Number of Teachers: " + teachers.getSize(), SwingConstants.CENTER);
        currentTeachersCntLabel.setTextLabel(500, 50, getWidth() / 2 - 250, 360, Color.BLUE, null, new Font("Arial", Font.BOLD, 30));
        currentTeachersCntLabel.setOpaque(false);

        teachersNeededLabel = new MyTextLabel("<html>Number of teachers currently needed: " + 0 + "<br>(Note: 1 teacher can support 30 students)</html>", SwingConstants.CENTER);
        teachersNeededLabel.setTextLabel(500, 60, getWidth() / 2 - 250, 420, Color.RED, Color.YELLOW, new Font("Arial", Font.BOLD, 22));
        teachersNeededLabel.setOpaque(true);

        backToLoginScreenButton = new MyButton("Back");
        backToLoginScreenButton.setButton(150, 50, 0, 735);
        backToLoginScreenButton.addActionListener(this);

        quitButton = new MyButton("Quit");
        quitButton.setButton(150, 50, 515, 735);
        quitButton.addActionListener(this);

        addStudentButton = new MyButton("Add a student");
        addStudentButton.setButton(150, 50, 135, 70);
        addStudentButton.addActionListener(this);

        editCurrentStudentsButton = new MyButton("Edit current student(s)");
        editCurrentStudentsButton.setButton(165, 50, 314, 70);
        editCurrentStudentsButton.addActionListener(this);

        coursesCodeButton = new MyButton("<html>View all possible courses</html>");
        coursesCodeButton.setButton(165, 50, 494, 70);
        coursesCodeButton.setBackground(Color.ORANGE);
        coursesCodeButton.addActionListener(this);

        addTeacherButton = new MyButton("Add a teacher");
        addTeacherButton.setButton(150, 50, 690, 70);
        addTeacherButton.addActionListener(this);

        editCurrentTeachersButton = new MyButton("Edit current teacher(s)");
        editCurrentTeachersButton.setButton(165, 50, 873, 70);
        editCurrentTeachersButton.addActionListener(this);

        contactInformationButton = new MyButton("<html>Contact Information<br>of student(s) and teacher(s)</html>");
        contactInformationButton.setButton(150, 70, getWidth() / 2 - 75, 190);
        contactInformationButton.addActionListener(this);

        historyButton = new MyButton("History");
        historyButton.setButton(150, 50, getWidth() / 2 - 75, 550);
        historyButton.addActionListener(this);

```

```java
        imageCitationsButton = new MyButton("Image Citations");
        imageCitationsButton.setButton(150, 50, 1035, 735);
        imageCitationsButton.addActionListener(this);

        canvas.add(infoLabel);
        canvas.add(currentStudentsCntLabel);
        canvas.add(currentTeachersCntLabel);
        canvas.add(teachersNeededLabel);

        canvas.add(backToLoginScreenButton);
        canvas.add(quitButton);

        canvas.add(addStudentButton);
        canvas.add(editCurrentStudentsButton);

        canvas.add(addTeacherButton);
        canvas.add(editCurrentTeachersButton);

        canvas.add(coursesCodeButton);

        canvas.add(contactInformationButton);

        canvas.add(historyButton);

        canvas.add(imageCitationsButton);

        // Updating the current info about the # of students, # of teachers, and # of teachers required.
        updateCnt();
    }

    /**
     * Retrieving the Student associated with a given ID number
     * @param id The ID number of a Student
     * @return The Student associated with the given ID number
     */
    public Student findStudent(int id) {
        for (int i = 0; i < students.getSize(); i ++) {
            Student stu = students.get(0, students.getHead(), i);
            if (stu.getID() == id) return stu;
        }
        return null;
    }

    /**
     * Retrieving the Teacher associated with a given ID number
     * @param id The ID number of a Teacher
     * @return The Teacher associated with the given ID number
```

```java
     */
    public Teacher findTeacher(int id) {
        for (int i = 0; i < teachers.getSize(); i ++) {
            Teacher tch = teachers.get(0, teachers.getHead(), i);
            if (tch.getID() == id) return tch;
        }
        return null;
    }

    /**
     * Updating the current info about the # of students, # of teachers, and # of teachers required.
     */
    public void updateCnt() {
        currentStudentsCntLabel.setText("Current Number of Students: " + students.getSize());
        currentTeachersCntLabel.setText("Current Number of Teachers: " + teachers.getSize());

        int teachersNeeded = Math.max((int) Math.ceil(students.getSize() / (double)2) - teachers.getSize(), 0);
        teachersNeededLabel.setText("<html>Number of teachers currently needed: " + teachersNeeded + "<br>(Note: 1 teacher can support 30 students)</html>");
    }

    public void paint(Graphics g) {
        super.paintComponents(g);
        repaint();
    }

    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();

        if (button.equals(backToLoginScreenButton)) {   // User wants to go back to the login screen
            setVisible(false);
            DriverClass.displayLoginPage();
        } else if (button.equals(quitButton)) {   // User wants to quit or exit the program
            System.exit(0);
        } else if (button.equals(addStudentButton)) {   // User wants to add a student to the database
            setVisible(false);
            DriverClass.displayAddAStudentPage();
        } else if (button.equals(editCurrentStudentsButton)) {   // User wants to edit the added students
            setVisible(false);

            students.setOriginalList();
            if (EditCurrentStudentsPage.wasSorted())
                students.sortAlphabetically(true);

            DriverClass.displayEditCurrentStudentsPage();
        } else if (button.equals(coursesCodeButton)) {   // User wants to view all possible courses offered at my high school
            setVisible(false);
```

```
            DriverClass.displayViewCoursesPage();
        } else if (button.equals(addTeacherButton)) {   // User wants to add a teacher to the database
            setVisible(false);
            DriverClass.displayAddATeacherPage();
        } else if (button.equals(editCurrentTeachersButton)) {   // User wants to edit the added teachers
            setVisible(false);

            teachers.setOriginalList();
            if (EditCurrentTeachersPage.wasSorted())
                teachers.sortAlphabetically(true);

            DriverClass.displayEditCurrentTeachersPage();
        } else if (button.equals(contactInformationButton)) {   // User wants to view the contact information of the current students and teachers in the database.
            students.setOriginalList();
            teachers.setOriginalList();

            setVisible(false);
            DriverClass.displayContactInformationPage();
        } else if (button.equals(historyButton)) {   // User wants to view the history of events that have occurred while they used the program
            setVisible(false);
            DriverClass.displayHistoryPage();
        } else if (button.equals(imageCitationsButton)) {   // User wants to view the citations of the background images used in this program
            setVisible(false);
            DriverClass.displayImageCitationsPage();
        }
    }

    /**
     * Add a Student to the end of the "students" linked list
     * @param stu A Student
     */
    public void addStudent(Student stu) {
        students.addLast(stu);
    }

    /**
     * Add a Teacher to the end of the "teachers" linked list
     * @param tch A Teacher
     */
    public void addTeacher(Teacher tch) {
        teachers.addLast(tch);
    }

    /**
     * Get the list of current Students in this database
```

```
     * @return The list of current Students in this database
     */
    public LinkedListStudent getStudentsList() {
        return students;
    }

    /**
     * Get the list of current Teachers in this database
     * @return The list of current Teachers in this database
     */
    public LinkedListTeacher getTeachersList() {
        return teachers;
    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;

// Canvas for the "Add a student" page and "Add a teacher" page
public class AddAPersonCanvas extends JPanel {
    private static final long serialVersionUID = 1L;

    private BufferedImage background;

    /**
     * Initializes the values for a "AddAPersonCanvas" object
     */
    public AddAPersonCanvas() {
        try {
            background = ImageIO.read(this.getClass().getResource("/addAPersonBackground.jpg"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        setLayout(null);
    }

    /**
     * Setting up the background image for this canvas
     */
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(background, 0, 0, getWidth(), getHeight(), null);
    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import driverProgram.*;
import linkedList.*;
import objects.*;

public class AddAStudentPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private MyTextLabel necessaryInfoTextLabel, nameTextLabel, gradeTextLabel, inIBTextLabel;
    private MyTextField nameTextField;
    private MyButton backButton, addStudentButton;
    private static MyButton gr9Button, gr10Button, gr11Button, gr12Button, inIB_Button, notInIB_Button;

    private int grade;   // Selected grade for the student

    /* "inIB" is true if student is in pre-IB or IB, false otherwise.
     * "notInIB" returns true if the student is not in pre-IB or IB, false otherwise. */
    private boolean inIB, notInIB;

    /**
     * Initializes the values for a "AddAStudentPage" object
     * @param canvas The canvas for the "Add A Student" page
     */
    public AddAStudentPage(JPanel canvas) {
        super ("Add A Student");
        add(canvas);

        setSize(600, 830);
        setVisible(false);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        grade = 0;

        inIB = false; notInIB = false;

        necessaryInfoTextLabel = new MyTextLabel("<html>Note: Fields marked with '*' means<br>that they are necessary to fill out.</html>", SwingConstants.CENTER);
        necessaryInfoTextLabel.setTextLabel(280, 80, 155, 40, Color.BLACK, Color.ORANGE, new Font("Arial", Font.BOLD, 16));

        nameTextLabel = new MyTextLabel("(*) Full name:", SwingConstants.CENTER);
```

```
        nameTextLabel.setTextLabel(125, 20, 170, 190, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 18));

        nameTextField = new MyTextField();
        nameTextField.setTextField(250, 30, 170, 210);

        gradeTextLabel = new MyTextLabel("(*) Grade:", SwingConstants.CENTER);
        gradeTextLabel.setTextLabel(100, 20, 170, 260, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 18));

        gr9Button = new MyButton("Grade 9");
        gr9Button.setButton(78, 30, 170, 280);
        gr9Button.addActionListener(this);

        gr10Button = new MyButton("Grade 10");
        gr10Button.setButton(84, 30, 248, 280);
        gr10Button.addActionListener(this);

        gr11Button = new MyButton("Grade 11");
        gr11Button.setButton(84, 30, 332, 280);
        gr11Button.addActionListener(this);

        gr12Button = new MyButton("Grade 12");
        gr12Button.setButton(84, 30, 248, 310);
        gr12Button.addActionListener(this);

        inIBTextLabel = new MyTextLabel("<html>(*) Is this student currently in<br>the pre-IB or IB program?</html>", SwingConstants.CENTER);
        inIBTextLabel.setTextLabel(210, 35, 190, 370, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 14));

        inIB_Button = new MyButton("Yes");
        inIB_Button.setButton(80, 40, 190, 415);
        inIB_Button.addActionListener(this);

        notInIB_Button = new MyButton("No");
        notInIB_Button.setButton(80, 40, 320, 415);
        notInIB_Button.addActionListener(this);

        addStudentButton = new MyButton("Add this student");
        addStudentButton.setButton(130, 40, 227, 500);
        addStudentButton.addActionListener(this);

        backButton = new MyButton("Back");
        backButton.setButton(160, 50, 210, 735);
        backButton.addActionListener(this);

        canvas.add(necessaryInfoTextLabel);

        canvas.add(nameTextLabel);
        canvas.add(nameTextField);
```

```java

        canvas.add(gradeTextLabel);
        canvas.add(gr9Button);
        canvas.add(gr10Button);
        canvas.add(gr11Button);
        canvas.add(gr12Button);

        canvas.add(inIBTextLabel);
        canvas.add(inIB_Button);
        canvas.add(notInIB_Button);

        canvas.add(addStudentButton);

        canvas.add(backButton);
    }

    /**
     * Resetting the "grade", "inIB", and "notInIB" fields, clearing colours of buttons, clearing text from text fields
     */
    public void clearInfo() {
        grade = 0;
        inIB = false;
        notInIB = false;

        resetButtonColours();
        inIB_Button.setBackground(null);
        notInIB_Button.setBackground(null);
        nameTextField.setText("");
    }

    /**
     * Clearing the colours of the grade buttons
     */
    public static void resetButtonColours() {
        gr9Button.setBackground(null);
        gr10Button.setBackground(null);
        gr11Button.setBackground(null);
        gr12Button.setBackground(null);
    }

    public void paint(Graphics g) {
        super.paintComponents(g);
        repaint();
    }

    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();
```

```java

        if (button.equals(backButton)) {   // User wants to go back to the main menu page
            clearInfo();

            setVisible(false);
            DriverClass.displayMainMenu();
        } else if (button.equals(addStudentButton)) {   // User has chosen to add a student to this database program
            String name = nameTextField.getText().trim();
            String[] nameSplit = name.split(" ");

            if (!name.equals("") && grade != 0 && (inIB || notInIB)) {   // If the user filled in all necessary fields
                if (nameSplit.length == 1) {   // If the user has not entered a last name for the student
                    JOptionPane.showMessageDialog(this, "Please enter a last name for this student.", "Warning", JOptionPane.WARNING_MESSAGE);
                }
                else {   // User has successfully filled in all necessary field correctly

                    LinkedListStudent students = DriverClass.getStudents();
                    int id;

                    // Making sure that a unique ID is generated for the current student being added.
                    while (true) {
                        id = Person.generateID();
                        boolean flag = true;
                        for (int i = 0; i < students.getSize(); i ++) {
                            if (students.get(0, students.getHead(), i).getID() == id) {
                                flag = false;
                                break;
                            }
                        }
                        if (flag) break;
                    }

                    // Creating the student.
                    Student temp = new Student();
                    temp.setName(name);
                    temp.setID(id);
                    temp.setGrade(grade);
                    temp.setIB(inIB);
                    temp.generateEmail();

                    // Adding the student to the "students" linkedlist in the main menu.
                    DriverClass.addStudent(temp);

                    // This block of code updates the history page when a new student is added.
                    String str = temp.getName() + " has been added to the school as a grade " + temp.getGrade() + " student ";
                    if (temp.isInIB()) {
                        if (temp.getGrade() <= 10) str += "currently in the pre-IB program.";
```

```
                        else str += "currently in the IB program.";
                    } else str += "currently not in the pre-IB or IB program.";
                    DriverClass.addEventToHistory(str, true);

                    // Adding the student's info to the "StudentsInfo.txt" RandomAccessFile.
                    try {
                        RandomAccessFile studentsInfo = new RandomAccessFile("StudentsInfo.txt", "rw");
                        studentsInfo.seek(studentsInfo.length());
                        studentsInfo.writeBytes("Add_" + name + "_" + id + "_" + grade + "_" + inIB + "_" + temp.getEmail() + "\n");
                        studentsInfo.close();
                    } catch (IOException e1) {
                        e1.printStackTrace();
                    }

                    // Clearing the private field's info, textfields, and button colours to allow for another student to be added.
                    clearInfo();

                    JOptionPane.showMessageDialog(this, "Student added successfully.");
                }
            } else {   // User has not filled in all necessary fields
                JOptionPane.showMessageDialog(this, "Please make sure to fill all the necessary fields.", "Warning",
JOptionPane.WARNING_MESSAGE);
            }
        } else if (button.equals(inIB_Button)) {   // User has selected the student to be in the pre-IB or IB program
            notInIB_Button.setBackground(null);
            inIB_Button.setBackground(Color.GREEN);
            inIB = true;
        } else if (button.equals(notInIB_Button)) {   // User has selected the student to not be in the pre-IB or IB program
            inIB_Button.setBackground(null);
            notInIB_Button.setBackground(Color.RED);
            notInIB = true;
        } else if (button.equals(gr9Button)) {   // User has selected the student to be in grade 9
            grade = 9;
            resetButtonColours();
            gr9Button.setBackground(Color.GREEN);
        } else if (button.equals(gr10Button)) {   // User has selected the student to be in grade 10
            grade = 10;
            resetButtonColours();
            gr10Button.setBackground(Color.GREEN);
        } else if (button.equals(gr11Button)) {   // User has selected the student to be in grade 11
            grade = 11;
            resetButtonColours();
            gr11Button.setBackground(Color.GREEN);
        } else if (button.equals(gr12Button)) {   // User has selected the student to be in grade 12
            grade = 12;
            resetButtonColours();
            gr12Button.setBackground(Color.GREEN);
```

```
        }
    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import driverProgram.*;
import linkedList.*;
import objects.*;

public class EditCurrentStudentsPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private static LinkedListStudent studentsList;   // Linked list of students who are currently in the database

    private MyButton sortStudentsAlphabeticallyButton, removeThisStudentButton;
    private MyButton editCompletedCoursesButton, editCurrentCoursesButton, generateTimetableButton, displayTimetableButton;
    private MyButton backButton;

    private MyTextLabel choiceLabel;
    private MyTextLabel timetableSemester1Label, timetableSemester2Label;
    private MyTextLabel[] timetableDayLabels;
    private MyTextLabel[] timetableTimePeriods;
    private MyButton[][] timetableSemester1, timetableSemester2;

    private Choice studentsMenu;

    private Student curTimetableShownStudent;   // Stores the info of the student whose timetable is currently shown

    private static boolean sorted;   // Stores whether the list of students is currently sorted alphabetically or not

    /**
     * Initializes the values for a "EditCurrentStudentsPage" object
     */
    public EditCurrentStudentsPage() {
        super ("Edit current students");

        setLayout(null);
        setSize(800, 800);
        setVisible(false);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        sorted = false;

        curTimetableShownStudent = new Student();

```

```java
        studentsList = new LinkedListStudent();

        choiceLabel = new MyTextLabel("Select a student below", SwingConstants.CENTER);
        choiceLabel.setTextLabel(400, 40, getWidth() / 2 - 200, 20, Color.BLACK, null, new Font("Arial", Font.BOLD, 30));

        studentsMenu = new Choice();
        studentsMenu.setBounds(getWidth() / 2 - 100, 75, 200, 40);

        for (int i = 0; i < studentsList.getSize(); i ++) {
            studentsMenu.add(studentsList.get(0, studentsList.getHead(), i).getName());
        }

        timetableSemester1Label = new MyTextLabel("Semester 1 timetable", SwingConstants.CENTER);
        timetableSemester1Label.setTextLabel(370, 25, getWidth() / 2 - 150, 180, Color.BLACK, null, new Font("Arial", Font.BOLD, 20));
        timetableSemester2Label = new MyTextLabel("Semester 2 timetable", SwingConstants.CENTER);
        timetableSemester2Label.setTextLabel(370, 25, getWidth() / 2 - 150, 470, Color.BLACK, null, new Font("Arial", Font.BOLD, 20));

        timetableSemester1 = new MyButton[4][4];
        timetableSemester2 = new MyButton[4][4];

        int timetable1curX = getWidth() / 2 - 140, timetable1curY = 230;
        int timetable2curX = getWidth() / 2 - 140, timetable2curY = 520;

        for (int i = 0; i < 4; i ++) {
            for (int j = 0; j < 4; j ++) {
                timetableSemester1[i][j] = new MyButton("");
                timetableSemester1[i][j].setButton(90, 50, timetable1curX, timetable1curY);

                timetable1curX += 90;

                timetableSemester2[i][j] = new MyButton("");
                timetableSemester2[i][j].setButton(90, 50, timetable2curX, timetable2curY);

                timetable2curX += 90;
            }
            timetable1curX = getWidth() / 2 - 140;
            timetable1curY += 50;

            timetable2curX = getWidth() / 2 - 140;
            timetable2curY += 50;
        }

        timetableDayLabels = new MyTextLabel[8];
        int curX = getWidth() / 2 - 140, curY = 210;
        for (int i = 0; i < 8; i ++) {
            timetableDayLabels[i] = new MyTextLabel("Day " + (i + 1), SwingConstants.CENTER);
            timetableDayLabels[i].setTextLabel(90, 20, curX, curY, Color.BLACK, Color.YELLOW, new Font("Arial", Font.PLAIN, 14));
```

```
            curX += 90;
        }
        curX = getWidth() / 2 - 140; curY = 500;
        for (int i = 4; i < 8; i ++) {
            timetableDayLabels[i] = new MyTextLabel("Day " + (i - 4 + 1), SwingConstants.CENTER);
            timetableDayLabels[i].setTextLabel(90, 20, curX, curY, Color.BLACK, Color.YELLOW, new Font("Arial", Font.PLAIN, 14));
            curX += 90;
        }

        timetableTimePeriods = new MyTextLabel[8];
        curX = getWidth() / 2 - 220; curY = 230;
        for (int i = 0; i < 4; i ++) {
            if (i == 0) timetableTimePeriods[i] = new MyTextLabel("08:45-10:05", SwingConstants.CENTER);
            else if (i == 1) timetableTimePeriods[i] = new MyTextLabel("10:10-11:25", SwingConstants.CENTER);
            else if (i == 2) timetableTimePeriods[i] = new MyTextLabel("12:25-13:45", SwingConstants.CENTER);
            else timetableTimePeriods[i] = new MyTextLabel("13:50-15:00", SwingConstants.CENTER);
            timetableTimePeriods[i].setTextLabel(80, 50, curX, curY, Color.BLACK, Color.YELLOW, new Font("Arial", Font.PLAIN, 14));
            curY += 50;
        }
        curX = getWidth() / 2 - 220; curY = 520;
        for (int i = 4; i < 8; i ++) {
            if (i - 4 == 0) timetableTimePeriods[i] = new MyTextLabel("08:45-10:05", SwingConstants.CENTER);
            else if (i - 4 == 1) timetableTimePeriods[i] = new MyTextLabel("10:10-11:25", SwingConstants.CENTER);
            else if (i - 4 == 2) timetableTimePeriods[i] = new MyTextLabel("12:25-13:45", SwingConstants.CENTER);
            else timetableTimePeriods[i] = new MyTextLabel("13:50-15:00", SwingConstants.CENTER);
            timetableTimePeriods[i].setTextLabel(80, 50, curX, curY, Color.BLACK, Color.YELLOW, new Font("Arial", Font.PLAIN, 14));
            curY += 50;
        }

        backButton = new MyButton("Back");
        backButton.setButton(160, 50, 0, 713);
        backButton.addActionListener(this);

        sortStudentsAlphabeticallyButton = new MyButton("<html>Sort list of student(s)<br>alphabetically</html>");
        sortStudentsAlphabeticallyButton.setButton(190, 40, 90, 65);
        sortStudentsAlphabeticallyButton.setBackground(Color.CYAN);
        sortStudentsAlphabeticallyButton.addActionListener(this);

        removeThisStudentButton = new MyButton("Remove this student");
        removeThisStudentButton.setButton(190, 30, getWidth() / 2 + 120, 70);
        removeThisStudentButton.setBackground(Color.RED);
        removeThisStudentButton.addActionListener(this);

        editCompletedCoursesButton = new MyButton("<html>Edit this student's<br>completed courses</html>");
        editCompletedCoursesButton.setButton(190, 50, 90, 110);
        editCompletedCoursesButton.addActionListener(this);

```

```
        editCurrentCoursesButton = new MyButton("<html>Edit this student's<br>current courses</html>");
        editCurrentCoursesButton.setButton(190, 50, getWidth() / 2 - 95, 110);
        editCurrentCoursesButton.addActionListener(this);

        generateTimetableButton = new MyButton("<html>Randomly generate this<br>student's current timetable</html>");
        generateTimetableButton.setButton(190, 50, getWidth() / 2 + 120, 110);
        generateTimetableButton.addActionListener(this);

        displayTimetableButton = new MyButton("<html>Display this student's<br>current timetable</html>");
        displayTimetableButton.setButton(110, 110, 50, 400);
        displayTimetableButton.setBackground(Color.GREEN);
        displayTimetableButton.addActionListener(this);

        add(choiceLabel);

        add(studentsMenu);

        add(removeThisStudentButton);

        add(sortStudentsAlphabeticallyButton);
        add(editCompletedCoursesButton);
        add(editCurrentCoursesButton);
        add(generateTimetableButton);
        add(displayTimetableButton);

        add(timetableSemester1Label);
        add(timetableSemester2Label);
        timetableSemester1Label.setVisible(false);
        timetableSemester2Label.setVisible(false);

        for (int i = 0; i < 8; i ++) {
            add(timetableDayLabels[i]);
            add(timetableTimePeriods[i]);
            timetableDayLabels[i].setVisible(false);
            timetableTimePeriods[i].setVisible(false);
        }

        for (int i = 0; i < 4; i ++) {
            for (int j = 0; j < 4; j ++) {
                add(timetableSemester1[i][j]);
                add(timetableSemester2[i][j]);
                timetableSemester1[i][j].setVisible(false);
                timetableSemester2[i][j].setVisible(false);
            }
        }

        removeThisStudentButton.setVisible(false);
```

```java

        editCompletedCoursesButton.setVisible(false);
        editCurrentCoursesButton.setVisible(false);
        generateTimetableButton.setVisible(false);
        displayTimetableButton.setVisible(false);

        add(backButton);
    }

    /**
     * Receiving the list of Students in the database from the main menu page and setting up the page layout from this list of Students
     * @param list The list of Students currently in the database
     */
    public void setStudentsList(LinkedListStudent list) {
        studentsList = list;

        remove(studentsMenu);

        studentsMenu = new Choice();
        studentsMenu.setBounds(getWidth() / 2 - 100, 75, 200, 40);

        for (int i = 0; i < studentsList.getSize(); i ++) {
            studentsMenu.add((i + 1) + ". " + studentsList.get(0, studentsList.getHead(), i).getName());
        }

        add(studentsMenu);

        if (studentsList.getSize() > 0) {   // List of students is empty, so hide all of the buttons and labels which are responsible for generating a student's timetable
            sortStudentsAlphabeticallyButton.setVisible(true);
            removeThisStudentButton.setVisible(true);

            timetableSemester1Label.setVisible(true);
            timetableSemester2Label.setVisible(true);

            for (int i = 0; i < 8; i ++) {
                timetableDayLabels[i].setVisible(true);
                timetableTimePeriods[i].setVisible(true);
            }

            for (int i = 0; i < 4; i ++) {
                for (int j = 0; j < 4; j ++) {
                    timetableSemester1[i][j].setVisible(true);
                    timetableSemester2[i][j].setVisible(true);
                }
            }

```

```
            editCompletedCoursesButton.setVisible(true);
            editCurrentCoursesButton.setVisible(true);
            generateTimetableButton.setVisible(true);
            displayTimetableButton.setVisible(true);
        } else {   // List of students is not empty, so display all of the buttons and labels necessary in generating a student's timetable
            sortStudentsAlphabeticallyButton.setVisible(false);
            removeThisStudentButton.setVisible(false);

            timetableSemester1Label.setVisible(false);
            timetableSemester2Label.setVisible(false);

            for (int i = 0; i < 8; i ++) {
                timetableDayLabels[i].setVisible(false);
                timetableTimePeriods[i].setVisible(false);
            }

            for (int i = 0; i < 4; i ++) {
                for (int j = 0; j < 4; j ++) {
                    timetableSemester1[i][j].setVisible(false);
                    timetableSemester2[i][j].setVisible(false);
                }
            }

            editCompletedCoursesButton.setVisible(false);
            editCurrentCoursesButton.setVisible(false);
            generateTimetableButton.setVisible(false);
            displayTimetableButton.setVisible(false);
        }
    }

    /**
     * Returns true if the list of Students on this page is currently sorted alphabetically
     * @return "True" if the list of Students on this page is currently sorted alphabetically, "false" otherwise
     */
    public static boolean wasSorted() {
        return sorted;
    }

    /**
     * Receives a string from the list of Students and returns the Student associated with that string
     * @param str The selected string from the Student's pop-up menu of choices
     * @return The Student associated with the selected string in the Student's pop-up menu of choices
     */
    private Student findStudent(String str) {
        int idx = Character.getNumericValue(str.charAt(0)) - 1;   // The index of the student in the list
        Student stu = studentsList.get(0, studentsList.getHead(), idx);
        return stu;
```

```
    }

    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();

        if (button.equals(backButton)) {   // User wants to go back to the main menu page
            setVisible(false);

            studentsList.sortAlphabetically(false);
            DriverClass.displayMainMenu();
        } else if (button.equals(sortStudentsAlphabeticallyButton)) {   // User wants to either sort the list of students alphabetically or unsort it back into its original order
            if (!sorted) {   // Current list of students is not sorted alphabetically, so sort the list of students alphabetically
                studentsList.sortAlphabetically(true);
                setStudentsList(studentsList);
                sorted = true;

                sortStudentsAlphabeticallyButton.setBackground(Color.ORANGE);
                sortStudentsAlphabeticallyButton.setText("<html>Unsort the list back into<br>original order</html>");
            } else {   // Current list of students is sorted alphabetically, so unsort the list of students back into its original order
                studentsList.sortAlphabetically(false);
                setStudentsList(studentsList);
                sorted = false;

                sortStudentsAlphabeticallyButton.setBackground(Color.CYAN);
                sortStudentsAlphabeticallyButton.setText("<html>Sort list of student(s)<br>alphabetically</html>");
            }
        } else if (button.equals(removeThisStudentButton)) {   // User wants to remove the currently selected student
            Student stu = findStudent(studentsMenu.getSelectedItem());

            // Removing the student from the "students" linked list in the main menu.
            studentsList.remove(stu);
            setStudentsList(studentsList);
            studentsList.updateOriginalList(stu);

            // Adding an event to the history page about the student being removed.
            DriverClass.addEventToHistory(stu.getName() + " has been removed as a student in this school.", true);

            // If the student being removed is the student whose timetable is currently shown, clear the timetable buttons and labels.
            if (stu == curTimetableShownStudent) {
                timetableSemester1Label.setText("Semester 1 timetable");
                timetableSemester2Label.setText("Semester 2 timetable");
                for (int i = 0; i < 4; i ++) {
                    for (int j = 0; j < 4; j ++) {
                        timetableSemester1[i][j].setText("");
                        timetableSemester2[i][j].setText("");
                    }
```

```
                }
            }

            // Inform the "StudentsInfo.txt" RandomAccessFile about removing the selected student
            try {
                RandomAccessFile studentsInfo = new RandomAccessFile("StudentsInfo.txt", "rw");
                studentsInfo.seek(studentsInfo.length());
                studentsInfo.writeBytes("Remove_" + stu.getID() + "\n");
                studentsInfo.close();
            } catch (FileNotFoundException e1) {
                e1.printStackTrace();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        } else if (button.equals(editCompletedCoursesButton)) {   // User wants to edit the selected student's completed courses
            Student stu = findStudent(studentsMenu.getSelectedItem());
            stu.displayCoursesPage(true);
        } else if (button.equals(editCurrentCoursesButton)) {   // User wants to edit the selected student's current courses
            Student stu = findStudent(studentsMenu.getSelectedItem());
            stu.displayCoursesPage(false);
        } else if (button.equals(generateTimetableButton)) {   // User wants to generate the selected student's timetable
            Student stu = findStudent(studentsMenu.getSelectedItem());

            if (stu.getCurrentCoursesList().getSize() != 8) {
                JOptionPane.showMessageDialog(this, "You must select 8 current courses for student '" +
studentsMenu.getSelectedItem().substring(3) + "' in order to generate their timetable.", "Warning", JOptionPane.WARNING_MESSAGE);
            }
            else {
                String[] parts = studentsMenu.getSelectedItem().split(" ");
                String firstName = parts[1];

                timetableSemester1Label.setText(firstName + "'s semester 1 timetable");
                timetableSemester2Label.setText(firstName + "'s semester 2 timetable");

                add(timetableSemester1Label);
                add(timetableSemester2Label);

                generateTimetable(stu);

                curTimetableShownStudent = stu;
            }
        } else if (button.equals(displayTimetableButton)) {   // User wants to display the selected student's timetable
            Student stu = findStudent(studentsMenu.getSelectedItem());

            String[] parts = studentsMenu.getSelectedItem().split(" ");
            String firstName = parts[1];

```

```
            timetableSemester1Label.setText(firstName + "'s semester 1 timetable");
            timetableSemester2Label.setText(firstName + "'s semester 2 timetable");

            displayTimetable(stu.getTimetable());

            curTimetableShownStudent = stu;
        }
    }

    /**
     * Generating a random timetable for the selected Student and displaying it
     * @param stu The Student whose timetable is to be randomly generated and displayed
     */
    public void generateTimetable(Student stu) {
        LinkedListCourses list = stu.getCurrentCoursesList();  // List of the student's current courses

        boolean[] visited = new boolean[8];  // Maximum of 8 courses in a timetable

        /* Each cell in this "timetable" array represents a unique course.
         * The ordering of these courses represent the ordering of the student's timetable. */
        String[] timetable = new String[8];  // Maximum of 8 courses in a timetable

        // Algorithm used to randomly generate the student's timetable and store it in the "timetable" array
        for (int i = 0; i < 8; i ++) {  // 8 Courses to be randomly assigned to student's timetable
            while (true) {  // This "while" statement will continue to be "true" until an empty spot is found in "list"
                int x = (int)(Math.random() * 8);  // Generating a random number from 0 to 7 (representing all possible indices in "list")
                if (!visited[x]) {  // "visited[x]" is "false" when "list.get(x)" has not yet been assigned to "timetable" array array
                    visited[x] = true;  // Mark index 'x' of "list" as occupied in the "timetable" array array

                    // Getting the course code associated with index 'x' from "list" and assigning it to "timetable[i]"
                    timetable[i] = list.get(0, list.getHead(), x).getCourseCode();

                    if (timetable[i].equals("Grade 12 Spare #1") || timetable[i].equals("Grade 12 Spare #2")) { // Special case
                        timetable[i] = "Spare";
                    }
                    break;
                }
            }
        }

        stu.setTimetable(timetable);  // Set this student's current timetable

        displayTimetable(timetable);  // Display the student's current timetable

        // Adding the info of the student's timetable into the "StudentsInfo.txt" RandomAccessFile.
        try {
            RandomAccessFile studentsInfo = new RandomAccessFile("StudentsInfo.txt", "rw");
```

```
            studentsInfo.seek(studentsInfo.length());

            String str = "Timetable_" + stu.getID() + "_";
            for (int i = 0; i < 8; i ++) str += timetable[i] + "_";

            studentsInfo.writeBytes(str + "\n");
            studentsInfo.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Displayed the selected Student's current timetable
     * @param arr The timetable array representing the ordering of the Student's current courses
     */
    public void displayTimetable(String[] arr) {
        int curRow1 = 0, curRow2 = 1, curRow3 = 2, curRow4 = 3;
        for (int i = 0; i < 4; i ++) {
            timetableSemester1[curRow1][i].setText(arr[0]);
            curRow1 ++;

            timetableSemester1[curRow2][i].setText(arr[1]);
            curRow2 ++;
            if (curRow2 == 4) curRow2 = 0;

            timetableSemester1[curRow3][i].setText(arr[2]);
            curRow3 ++;
            if (curRow3 == 4) curRow3 = 0;

            timetableSemester1[curRow4][i].setText(arr[3]);
            curRow4 ++;
            if (curRow4 == 4) curRow4 = 0;
        }

        curRow1 = 0; curRow2 = 1; curRow3 = 2; curRow4 = 3;
        for (int i = 0; i < 4; i ++) {
            timetableSemester2[curRow1][i].setText(arr[4]);
            curRow1 ++;

            timetableSemester2[curRow2][i].setText(arr[5]);
            curRow2 ++;
            if (curRow2 == 4) curRow2 = 0;

            timetableSemester2[curRow3][i].setText(arr[6]);
```

```
            curRow3 ++;
            if (curRow3 == 4) curRow3 = 0;

            timetableSemester2[curRow4][i].setText(arr[7]);
            curRow4 ++;
            if (curRow4 == 4) curRow4 = 0;
        }
    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import driverProgram.*;
import linkedList.*;
import objects.*;

public class StudentCoursesPage extends JFrame implements ActionListener, MouseListener {
    private static final long serialVersionUID = 1L;

    private MyButton finishButton;

    // All of the courses in our school stored in a 29 x 7 array called "courses" of type "String"
    private final String[][] courses =
    {{"", "Grade 9", "Grade 10", "Grade 11", "Grade 12", "Grade 12", "Grade 12"},
    {"English", "<html>ENG1D1 - Grade 9 English<br>Prerequisite(s): none</html>", "<html>ENG2D1 - Grade 10 English<br>Prerequisite(s): ENG1D1 or ENG1D7</html>", "<html>ENG3U1 - Grade 11 English<br>Prerequisite(s): ENG2D1 or ENG2D7</html>", "<html>ENG4U1 - Grade 12 English<br>Prerequisite(s): ENG3U1 or ENG3U7</html>", "<html>ETS4U1 - Grade 12 Film Studies<br>Prerequisite(s): ENG3U1 or ENG3U7</html>", ""},
    {"Math", "<html>MTH1W1 - Grade 9 Mathematics<br>Prerequisite(s): none</html>", "<html>MPM2D1 - Grade 10 Principles of Mathematics<br>Prerequisite(s): MTH1W1 or MPM1D7</html>", "<html>MCR3U1 - Grade 11 Functions<br>Prerequisite(s): MPM2D1 or MPM2D7</html>", "<html>MHF4U1 - Grade 11 Advanced Functions<br>Prerequisite(s): MCR3U1 or MCR3U7</html>", "<html>MCV4U1 - Grade 12 Calculus and Vectors<br>Prerequisite(s): MHF4U1 or MHF4U7</html>", ""},
    {"Science", "<html>SNC1D1 - Grade 9 Science<br>Prerequisite(s): none</html>", "<html>SNC2D1 - Grade 10 Science<br>Prerequisite(s): SNC1D1 or SNC1D7</html>", "", "", "", ""},
    {"Computer Studies", "", "<html>ICS2O1 - Grade 10 Introduction to Computer Studies<br>Prerequisite(s): none</html>", "<html>ICS3U1 - Grade 11 Introduction to Computer Science<br>Prerequisite(s): none</html>", "<html>ICS4U1 - Grade 12 Computer Science<br>Prerequisite(s): ICS3U1 or ICS3U7</html>", "", ""},
    {"Chemistry", "", "", "<html>SCH3U1 - Grade 11 Chemistry<br>Prerequisite(s): SNC2D1 or SNC2D7</html>", "<html>SCH4U1 - Grade 12 Chemistry<br>Prerequisite(s): SCH3U1 or SCH3U7</html>", "", ""},
    {"Physics", "", "", "<html>SPH3U1 - Grade 11 Physics<br>Prerequisite(s): SNC2D1 or SNC2D7</html>", "<html>SPH4U1 - Grade 12 Physics<br>Prerequisite(s): SPH3U1 or SPH3U7</html>", "", ""},
    {"Biology", "", "", "<html>SBI3U1 - Grade 11 Biology<br>Prerequisite(s): SNC2D1 or SNC2D7</html>", "<html>SBI4U1 - Grade 12 Biology<br>Prerequisite(s): SBI3U1 or SBI3U7</html>", "", ""},
    {"French", "<html>FSF1D1 - Grade 9 Core French<br>Prerequisite(s): none</html>", "<html>FSF2D1 - Grade 10 Core French<br>Prerequisite(s): FSF1D1 or FSF1D7</html>", "<html>FSF3U1 - Grade 11 Core French<br>Prerequisite(s): FSF2D1 or FSF2D7</html>", "<html>FSF4U1 - Grade 12 Core French<br>Prerequisite(s): FSF3U1 or FSF3U7</html>", "", ""},
    {"Gym (Males)", "<html>PPL1OM - Grade 9 Healthy Active Living Education (M)<br>Prerequisite(s): none</html>", "<html>PPL2OM - Grade 10 Healthy Active Living Education (M)<br>Prerequisite(s): none</html>", "<html>PPL3OM - Grade 11 Healthy Active Living Education (M)<br>Prerequisite(s): none</html>", "<html>PPL4OM - Grade 12 Healthy Active Living Education (M)<br>Prerequisite(s): none</html>", "", ""},
    {"Gym (Females)", "<html>PPL1OF - Grade 9 Healthy Active Living Education (F)<br>Prerequisite(s): none</html>", "<html>PPL2OF - Grade 10 Healthy Active Living Education (F)<br>Prerequisite(s): none</html>", "<html>PPL3OF - Grade 11 Healthy Active Living Education (F)<br>Prerequisite(s): none</html>", "<html>PPL4OF - Grade 12 Healthy Active Living Education (F)<br>Prerequisite(s): none</html>", "", ""},
    {"World Studies", "<html>CGC1D1 - Grade 9 Issues in Canadian Geography<br>Prerequisite(s): none</html>", "<html>CHC2D1 - Grade 10 Canadian History since World War I<br>Prerequisite(s): none</html>", "<html>CLU3M1 - Grade 11 Understanding Canadian Law<br>Prerequisite(s):
```

```
CHC2D1</html>", "<html>CLN4U1 - Grade 12 Canadian and International Law<br>Prerequisite(s): CLU3M1</html>", "", ""},
    {"Art", "<html>NAC101 - Grade 9 Expressing Aboriginal Cultures<br>Prerequisite(s): none</html>", "<html>AVI2O1 - Grade 10 Visual Arts<br>Prerequisite(s): none</html>", "<html>AVI3M1 - Grade 11 Visual Arts<br>Prerequisite(s): AVI2O1</html>", "<html>AVI4M1 - Grade 12 Visual Arts<br>Prerequisite(s): AVI3M1</html>", "", ""},
    {"Business Studies", "<html>BTT1O1 - Grade 9 Information and Communication Technology in Business<br>Prerequisite(s): none</html>", "<html>BBI2O1 - Grade 10 Introduction to Business<br>Prerequisite(s): none</html>", "<html>BAF3M1 - Grade 11 Financial Accounting Fundamentals<br>Prerequisite(s): none</html>", "<html>BAT4M1 - Grade 12 Financial Accounting Principles<br>Prerequisite(s): BAF3M1</html>", "<html>BBB4M1 - Grade 12 International Business Fundamentals<br>Prerequisite(s): BAF3M1</html>", ""},
    {"Technological Education", "<html>TIJ1O1 - Grade 9 Exploring Technologies<br>Prerequisite(s): none</html>", "<html>TEJ2O1 - Grade 10 Computer Technology<br>Prerequisite(s): none</html>", "<html>TEJ3M1 - Grade 11 Computer Engineering Technology<br>Prerequisite(s): none</html>", "<html>TEJ4M1 - Grade 12 Computer Engineering Technology<br>Prerequisite(s): TEJ3M1</html>", "", ""},
    {"", "", "", "", "", "", ""},
    {"English (IB)", "<html>ENG1D7 - Grade 9 English (IB)<br>Prerequisite(s): none</html>", "<html>ENG2D7 - Grade 10 English (IB)<br>Prerequisite(s): ENG1D7</html>", "<html>ENG3U7 - Grade 11 English (IB)<br>Prerequisite(s): ENG2D7</html>", "<html>ENG4U7 - Grade 12 English (IB)<br>Prerequisite(s): ENG3U7</html>", "<html>ETS4U7 - Grade 12 Studies in Literature (IB)<br>Prerequisite(s): ENG3U7</html>", ""},
    {"Math (IB)", "<html>MPM1D7 - Grade 9 Principles of Mathematics (IB)<br>Prerequisite(s): none</html>", "<html>MPM2D7 - Grade 10 Principles of Mathematics (IB)<br>Prerequisite(s): MPM1D7</html>", "<html>MCR3U7 - Grade 11 Functions (IB)<br>Prerequisite(s): MPM2D7</html>", "<html>MDM4U7 - Grade 12 Mathematics of Data Management (IB)<br>Prerequisite(s): MCR3U7</html>", "<html>MHF4U7 - Grade 12 Advanced Functions (IB)<br>Prerequisite(s): MCR3U7</html>", "<html>MCV4U7 - Grade 12 Calculus and Vectors (IB)<br>Prerequisite(s): MCR3U7</html>"},
    {"Science (IB)", "<html>SNC1D7 - Grade 9 Science (IB)<br>Prerequisite(s): none</html>", "<html>SNC2D7 - Grade 10 Science (IB)<br>Prerequisite(s): SNC1D7</html>", "", "", "", ""},
    {"Computer Studies (IB)", "", "<html>ICS2O7 - Grade 10 Introduction to Computer Studies (IB)<br>Prerequisite(s): none</html>", "<html>ICS3U7 - Grade 11 Introduction to Computer Science (IB)<br>Prerequisite(s): ICS2O7</html>", "<html>ICS4U7 - Grade 12 Computer Science (IB)<br>Prerequisite(s): ICS3U7</html>", "", ""},
    {"Chemistry (IB)", "", "", "<html>SCH3U7 - Grade 11 Chemistry (IB)<br>Prerequisite(s): SNC2D7</html>", "<html>SCH4UP - Grade 12 Chemistry (IB)<br>Prerequisite(s): SCH3U7</html>", "", ""},
    {"Physics (IB)", "", "", "<html>SPH3U7 - Grade 11 Physics (IB)<br>Prerequisite(s): SNC2D7</html>", "<html>SPH4U7 - Grade 12 Physics (IB)<br>Prerequisite(s): SPH3U7</html>", "", ""},
    {"Biology (IB)", "", "", "<html>SBI3U7 - Grade 11 Biology (IB)<br>Prerequisite(s): SNC2D7</html>", "<html>SBI4U7 - Grade 12 Biology (IB)<br>Prerequisite(s): SBI3U7</html>", "", ""},
    {"French (IB)", "<html>FSF1D7 - Grade 9 Core French (IB)<br>Prerequisite(s): none</html>", "<html>FSF2D7 - Grade 10 Core French (IB)<br>Prerequisite(s): FSF1D7</html>", "<html>FSF3U7 - Grade 11 Core French (IB)<br>Prerequisite(s): FSF2D7</html>", "<html>FSF4U7 - Grade 12 Core French (IB)<br>Prerequisite(s): FSF3U7</html>", "", ""},
    {"Geography (IB)", "<html>CGC1D7 - Grade 9 Issues in Canadian Geography (IB)<br>Prerequisite(s): none</html>", "", "<html>CGF3M7 - Grade 11 Forces of Nature: Physical Processes and Disasters (IB)<br>Prerequisite(s): CGC1D7</html>", "<html>CGW4U7 - Grade 12 World Issues: A Geographic Analysis (IB)<br>Prerequisite(s): CGF3M7</html>", "", ""},
    {"History & Economics (IB)", "", "<html>CHC2D7 - Grade 10 Canadian History since World War I (IB)<br>Prerequisite(s): none</html>", "<html>CIE3M7 - Grade 11 The Individual & the Economy (IB)<br>Prerequisite(s): CHC2D7</html>", "<html>CIA4U7 - Grade 12 Analysing Current Economic Issues (IB)<br>Prerequisite(s): CIE3M7</html>", "<html>BBB4M7 - Grade 12 Economics (IB HL)<br>Prerequisite(s): CIA4U7</html>", ""},
    {"Theory Of Knowledge (IB)", "", "", "", "<html>HZT4U7 - Grade 12 Philosophy: Questions and Theories (IB)<br>Prerequisite(s): ENG2D7</html>", "", ""},
    {"", "", "", "", "", "", ""},
    {"Spares", "", "", "", "Grade 12 Spare #1", "Grade 12 Spare #2", ""}};

    private Button[][] buttonArr;   // Array of buttons that will be used to represent the course codes of each course
    private boolean[][] completedCourses, currentCourses, possibleCourses;   // Arrays to store the student's completed courses, current courses, and their possible courses to take
```

```java

    private boolean userHasClickedCompletedCourses;   // This variable is "true" if the user has clicked the "Edit this student's completed courses" button. Otherwise, this variable is false.

    private MyTextLabel infoLabel;

    private LinkedListCourses completedCoursesList, currentCoursesList;

    private Student stu;   // The student associated with this page

    private JPanel canvas;

    /**
     * Initializes the values for a "StudentCoursesPage" object
     * @param str The title of this "StudentCoursesPage" object
     * @param s The Student associated with this "StudentCoursesPage" object
     */
    public StudentCoursesPage(String str, Student s) {
        super (str);

        setSize(1100, 800);
        setVisible(false);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        stu = s;

        canvas = new JPanel();
        canvas.setLayout(new GridLayout(0, 7));

        buttonArr = new Button[courses.length][courses[0].length];
        completedCourses = new boolean[courses.length][courses[0].length];
        currentCourses = new boolean[courses.length][courses[0].length];
        possibleCourses = new boolean[courses.length][courses[0].length];

        userHasClickedCompletedCourses = false;

        completedCoursesList = new LinkedListCourses();
        currentCoursesList = new LinkedListCourses();

        infoLabel = new MyTextLabel("<html>Orange button(s) represent the student's completed course(s). Green button(s) represent the student's current course(s).<br>Red button(s) represent the course(s) that the student cannot select because they do not have the prerequisites for it.<br>Left click on a button to select or deselect a course. Right click on a button to view its course description and prerequisites.</html>", SwingConstants.CENTER);
        infoLabel.setTextLabel(400, 40, 0, 0, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 14));

```

```java
        finishButton = new MyButton("Finish");
        finishButton.addActionListener(this);

        add(canvas);

        add(infoLabel, BorderLayout.NORTH);
        add(finishButton, BorderLayout.EAST);
    }

    public void paint(Graphics g) {
        super.paintComponents(g);
        repaint();
    }

    /**
     * Setting whether or not the user has clicked "Edit this student's completed courses" button or the "Edit this student's current courses" button
     * @param b "True" if the user has clicked the "Edit this student's completed courses" button, "false" if the user has clicked the "Edit this student's current courses" button
     */
    public void setBool(boolean b) {
        userHasClickedCompletedCourses = b;
    }

    /**
     * Set the 2D array of completed courses for this Student
     * @param c The linked list of courses representing all of this Student's completed courses
     */
    public void setCompletedCourses(LinkedListCourses c) {  // 'c' is a doubly linked list of courses
        completedCourses = new boolean[courses.length][courses[0].length];  // Each cell in this 2D array relates to a specific available or non-available course

        for (int a = 0; a < c.getSize(); a ++) {  // Looping through the elements of linked list 'c'
            String courseCode = c.get(0, c.getHead(), a).getCourseCode();  // Getting the course code from linked list 'c' at index 'a'

            // Nested 'for' loop that loops through all cells of the "completedCourses" 2D boolean array to find which cell is related to the "courseCode" String
            outerloop:
            for (int i = 1; i < courses.length; i ++) {
                for (int j = 1; j < courses[0].length; j ++) {
                    if (courses[i][j].length() == 0) continue;  // If there is no text in a particular cell, then this cell is empty and does not contain a course

                    String check;
                    // Finding the course code associated with a particular cell and assigning it to the "check" variable
                    if (i == courses.length - 1) check = courses[i][j];
                    else check = courses[i][j].substring(6, 12);
```

```java

                    // If the course code of linked list 'c' matches the cell's course code with row 'i' and column 'j', then mark this cell as a completed course and exit "outerloop"
                    if (courseCode.equals(check)) {
                        completedCourses[i][j] = true;
                        break outerloop;
                    }
                }
            }
        }

        completedCoursesList = c;  // Along with setting the "completedCourses" 2D boolean array, set the "completedCoursesList" as well
    }

    /**
     * Set the 2D array of current courses for this Student
     * @param c The linked list of courses representing all of this Student's current courses
     */
    public void setCurrentCourses(LinkedListCourses c) {
        currentCourses = new boolean[courses.length][courses[0].length];

        for (int a = 0; a < c.getSize(); a ++) {
            String courseCode = c.get(0, c.getHead(), a).getCourseCode();

            outerloop:
            for (int i = 1; i < courses.length; i ++) {
                for (int j = 1; j < courses[0].length; j ++) {
                    if (courses[i][j].length() == 0) continue;

                    String check;
                    if (i == courses.length - 1) check = courses[i][j];
                    else check = courses[i][j].substring(6, 12);
                    if (courseCode.equals(check)) {
                        currentCourses[i][j] = true;
                        break outerloop;
                    }
                }
            }
        }

        currentCoursesList = c;
    }

    /**
     * Get the 2D array representing this Student's completed courses
     * @return The 2D array representing this Student's completed courses
     */
```

```java
    public boolean[][] getCompletedCourses() {
        return completedCourses;
    }

    /**
     * Get the 2D array representing this Student's current courses
     * @return The 2D array representing this Student's current courses
     */
    public boolean[][] getCurrentCourses() {
        return currentCourses;
    }

    /**
     * Get this Student's completed courses list
     * @return This Student's completed courses list
     */
    public LinkedListCourses getCompletedCoursesList() {
        return completedCoursesList;
    }

    /**
     * Get this Student's current courses list
     * @return This Student's current courses list
     */
    public LinkedListCourses getCurrentCoursesList() {
        return currentCoursesList;
    }

    /**
     * Setting the page by: Highlighting the student's completed courses with orange buttons, highlighting the student's current courses with green buttons,
     * highlighting the courses that the student is not allowed to take with red buttons, and highlighting the student's possible courses to select with white buttons.
     * All other buttons are used for information and have no purpose when they are clicked (brown, black, and teal buttons)
     */
    public void setPage() {
        if (stu.getGrade() == 12) {   // If student is currently in grade 12, they are allowed to select spares as their courses
            possibleCourses[courses.length - 1][4] = true;
            possibleCourses[courses.length - 1][5] = true;
        }

        // Nested "for" loop used to locate all of the buttons with course codes on them
        for (int i = 1; i < courses.length - 1; i ++) {
            for (int j = 1; j < courses[0].length; j ++) {

                // If the student is not currently in the pre-IB or IB program , they cannot select a pre-IB or IB course as one of their current courses.
```

```java
                if (i >= 16 && j <= 26) {
                    if (!stu.isInIB()) {
                        possibleCourses[i][j] = false;
                        break;
                    }
                }

                if (courses[i][j].length() != 0) {
                    String[] str = courses[i][j].split(" ");

                    boolean flag = false;

                    // Locating the prerequisites of the current course to see if the student has the prerequisites to take this course
                    for (int k = 2; k < str.length; k ++) {
                        if (str[k].equals("none</html>")) {   // If there are no prerequisites for the course, then any student is allowed to take it
                            flag = true;
                            break;
                        }

                        if (str[k].length() >= 6 && Character.isDigit(str[k].charAt(3)) && Character.isDigit(str[k].charAt(5))) {   // If the current button is a valid course button
                            if (str[k].length() > 6) str[k] = str[k].substring(0, 6);


                            if (completedCoursesList.contains(str[k])) {   // If the student has one of the prerequisites for the current course, they are allowed to take it
                                flag = true;
                            }
                        }
                    }

                    possibleCourses[i][j] = flag;
                }
            }
        }

        // Displaying the top row of the buttons
        buttonArr[0][0] = new Button(courses[0][0]);
        canvas.add(buttonArr[0][0]);
        for (int i = 1; i < courses[0].length; i ++) {
            buttonArr[0][i] = new Button(courses[0][i]);
            buttonArr[0][i].setBackground(Color.CYAN);
            canvas.add(buttonArr[0][i]);
        }

        // Displaying the colours of the buttons based on the student's course selection
```

```java
        for (int i = 1; i < courses.length; i ++) {
            for (int j = 0; j < courses[0].length; j ++) {
                if (courses[i][j].length() == 0) {
                    buttonArr[i][j] = new Button("");
                    buttonArr[i][j].setBackground(Color.BLACK);
                    buttonArr[i][j].addActionListener(this);
                    buttonArr[i][j].addMouseListener(this);
                } else if (j == 0) {
                    buttonArr[i][j] = new Button(courses[i][j]);
                    buttonArr[i][j].setBackground(new Color(222,184,135));
                } else {
                    if (i == courses.length - 1) {
                        buttonArr[i][j] = new Button(courses[i][j]);
                        if (userHasClickedCompletedCourses || stu.getGrade() != 12) buttonArr[i][j].setBackground(Color.RED);
                    }
                    else buttonArr[i][j] = new Button(courses[i][j].substring(6, 12));

                    if (completedCourses[i][j]) buttonArr[i][j].setBackground(Color.ORANGE);
                    else if (currentCourses[i][j]) buttonArr[i][j].setBackground(Color.GREEN);
                    else if (!userHasClickedCompletedCourses) {
                        if (courses[i][j].charAt(11) == '7' && !stu.isInIB()) buttonArr[i][j].setBackground(Color.RED);
                        else if (!possibleCourses[i][j]) buttonArr[i][j].setBackground(Color.RED);
                    }
                    buttonArr[i][j].addActionListener(this);
                    buttonArr[i][j].addMouseListener(this);
                }
                canvas.add(buttonArr[i][j]);
            }
        }
        add(canvas);
    }

    // Find the course related to a certain button
    /**
     * Find the Course associated with a certain button
     * @param b The button that is used to find the Course associated with it
     * @return the Course associated with button 'b'
     */
    public String findButtonString(Button b) {
        for (int i = 1; i < courses.length; i ++) {
            for (int j = 1; j < courses[0].length; j ++) {
                if (buttonArr[i][j] == b) return courses[i][j];
            }
        }
        return null;
    }

```

```java
    /**
     * Remove all buttons from the canvas
     */
    public void resetButtons() {
        for (int i = 0; i < courses.length; i ++) {
            for (int j = 0; j < courses[0].length; j ++) {
                canvas.remove(buttonArr[i][j]);
            }
        }
    }

    /**
     * Setting the student's linked lists for their completed and current courses
     */
    public void setLists() {
        completedCoursesList = new LinkedListCourses();
        currentCoursesList = new LinkedListCourses();

        for (int i = 1; i < courses.length; i ++) {
            for (int j = 1; j < courses[0].length; j ++) {
                if (completedCourses[i][j]) {
                    completedCoursesList.addLast(new Course(courses[i][j].substring(6, 12)));
                }
                if (currentCourses[i][j]) {
                    if (i == courses.length - 1) currentCoursesList.addLast(new Course(courses[i][j]));
                    else currentCoursesList.addLast(new Course(courses[i][j].substring(6, 12)));
                }
            }
        }
    }

    public void actionPerformed(ActionEvent e) {
        Component button = (Component) e.getSource();

        if (button.equals(finishButton)) {   // The user has finished updating the student's completed/current courses and wants to exit the page
            setLists();
            resetButtons();
            setVisible(false);
            if (userHasClickedCompletedCourses) JOptionPane.showMessageDialog(this, "The student's completed courses has been updated successfully.");
            else JOptionPane.showMessageDialog(this, "The student's current courses has been updated successfully.");

            if (userHasClickedCompletedCourses) {   // User has initially chosen to update the student's completed courses

                // Adding the event of updating the student's completed courses to the history page.
                String str = stu.getName() + "'s completed courses has been updated to: {";
```

```
                for (int i = 0; i < completedCoursesList.getSize(); i ++) {
                    str += completedCoursesList.get(0, completedCoursesList.getHead(), i).getCourseCode();
                    if (i != completedCoursesList.getSize() - 1) str += ", ";
                }
                str += "}.";
                DriverClass.addEventToHistory(str, true);

                // Adding the event of updating the student's completed courses to the "StudentsInfo.txt" RandomAccessFile.
                try {
                    RandomAccessFile studentsInfo = new RandomAccessFile("StudentsInfo.txt", "rw");
                    studentsInfo.seek(studentsInfo.length());

                    str = "Edit_Completed Courses_" + stu.getID() + "_";
                    for (int i = 0; i < completedCoursesList.getSize(); i ++) {
                        str += completedCoursesList.get(0, completedCoursesList.getHead(), i).getCourseCode() + "_";
                    }
                    studentsInfo.writeBytes(str + "\n");
                    studentsInfo.close();
                } catch (FileNotFoundException e1) {
                    e1.printStackTrace();
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            } else {   // User has initially chosen to update the student's current courses

                // Adding the event of updating the student's current courses to the history page.
                String str = stu.getName() + "'s current courses has been updated to: {";
                for (int i = 0; i < currentCoursesList.getSize(); i ++) {
                    str += currentCoursesList.get(0, currentCoursesList.getHead(), i).getCourseCode();
                    if (i != currentCoursesList.getSize() - 1) str += ", ";
                }
                str += "}.";
                DriverClass.addEventToHistory(str, true);

                // Adding the event of updating the student's current courses to the "StudentsInfo.txt" RandomAccessFile.
                try {
                    RandomAccessFile studentsInfo = new RandomAccessFile("StudentsInfo.txt", "rw");
                    studentsInfo.seek(studentsInfo.length());

                    str = "Edit_Current Courses_" + stu.getID() + "_";
                    for (int i = 0; i < currentCoursesList.getSize(); i ++) {
                        str += currentCoursesList.get(0, currentCoursesList.getHead(), i).getCourseCode() + "_";
                    }
                    studentsInfo.writeBytes(str + "\n");
                    studentsInfo.close();
                } catch (FileNotFoundException e1) {
                    e1.printStackTrace();
```

```
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        } else {   // User has clicked on a button that was not the "finish" button

            boolean appropriateButton = false;   // Used to denote whether the user's clicked button is appropriate. (Black-coloured buttons are not appropriate buttons)

            if (userHasClickedCompletedCourses) {   // User has initially chosen to update the student's completed courses
                outerloop: for (int i = 1; i < courses.length; i ++) {
                    for (int j = 1; j < courses[0].length; j ++) {
                        if (courses[i][j].length() != 0 && button.equals(buttonArr[i][j])) {

                            if (i == courses.length - 1) {
                                JOptionPane.showMessageDialog(this, "It is impossible to have completed a spare in grades 9, 10, and 11.", "Warning", JOptionPane.WARNING_MESSAGE);
                            }
                            else if (currentCourses[i][j]) {
                                JOptionPane.showMessageDialog(this, "This action cannot be completed because this student is currently taking this course.", "Warning", JOptionPane.WARNING_MESSAGE);
                            }
                            else if (completedCourses[i][j]) {
                                button.setBackground(null);
                                completedCourses[i][j] = false;
                            } else {
                                button.setBackground(Color.ORANGE);
                                completedCourses[i][j] = true;
                            }
                            appropriateButton = true;
                            break outerloop;
                        }
                    }
                }
            } else {   // User has initially chosen to update the student's current courses
                outerloop: for (int i = 1; i < courses.length; i ++) {
                    for (int j = 1; j < courses[0].length; j ++) {
                        if (courses[i][j].length() != 0 && button.equals(buttonArr[i][j])) {
                            if (completedCourses[i][j]) {
                                JOptionPane.showMessageDialog(this, "Students cannot enroll in courses that they have completed.", "Warning", JOptionPane.WARNING_MESSAGE);
                            } else if (!possibleCourses[i][j]) {
                                if (i >= 16 && i <= 26 && !stu.isInIB()) JOptionPane.showMessageDialog(this, "This student cannot take this course because they are currently not in the pre-IB/IB program.", "Warning", JOptionPane.WARNING_MESSAGE);
                                else {
                                    if (i == courses.length - 1 && stu.getGrade() != 12) {
                                        JOptionPane.showMessageDialog(this, "Only grade 12 students can select spares.", "Warning",
```

```
JOptionPane.WARNING_MESSAGE);
                                    }
                                    else {
                                        JOptionPane.showMessageDialog(this, "This student does not have the prerequisite(s) for this course yet.", "Warning", JOptionPane.WARNING_MESSAGE);
                                    }
                                }
                            } else if (!currentCourses[i][j]){
                                button.setBackground(Color.GREEN);
                                currentCourses[i][j] = true;
                            } else if (currentCourses[i][j]) {
                                button.setBackground(null);
                                currentCourses[i][j] = false;
                                appropriateButton = true;
                            }
                            appropriateButton = true;
                            break outerloop;
                        }
                    }
                }
            }

            if (!appropriateButton) JOptionPane.showMessageDialog(this, "That is not an appropriate course to choose.", "Warning", JOptionPane.WARNING_MESSAGE);
        }
    }

    /**
     * This method is so if the user right clicks on a button, they can view the button's Course description and prerequisites needed for the Course.
     */
    @Override
    public void mouseClicked(MouseEvent e) {
        Button button = (Button) e.getSource();
        if(SwingUtilities.isRightMouseButton(e)) {
            String str = findButtonString(button);
            if (str.length() == 0) JOptionPane.showMessageDialog(this, "That is not an appropriate course.", "Warning", JOptionPane.WARNING_MESSAGE);
            else JOptionPane.showMessageDialog(this, str);
        }
    }

    @Override
    public void mousePressed(MouseEvent e) {

    }

```

```
    @Override
    public void mouseReleased(MouseEvent e) {

    }

    @Override
    public void mouseEntered(MouseEvent e) {

    }

    @Override
    public void mouseExited(MouseEvent e) {

    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import driverProgram.*;
import objects.*;

public class ViewCoursesPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private MyButton backButton;

    // All of the courses in our school stored in a 29 x 7 array called "courses" of type "String"
    private final String[][] courses =
    {{"", "Grade 9", "Grade 10", "Grade 11", "Grade 12", "Grade 12", "Grade 12"},
    {"English", "<html>ENG1D1 - Grade 9 English<br>Prerequisite(s): none</html>", "<html>ENG2D1 - Grade 10 English<br>Prerequisite(s): ENG1D1 or ENG1D7</html>", "<html>ENG3U1 - Grade 11 English<br>Prerequisite(s): ENG2D1 or ENG2D7</html>", "<html>ENG4U1 - Grade 12 English<br>Prerequisite(s): ENG3U1 or ENG3U7</html>", "<html>ETS4U1 - Grade 12 Film Studies<br>Prerequisite(s): ENG3U1 or ENG3U7</html>", ""},
    {"Math", "<html>MTH1W1 - Grade 9 Mathematics<br>Prerequisite(s): none</html>", "<html>MPM2D1 - Grade 10 Principles of Mathematics<br>Prerequisite(s): MTH1W1 or MPM1D7</html>", "<html>MCR3U1 - Grade 11 Functions<br>Prerequisite(s): MPM2D1 or MPM2D7</html>", "<html>MHF4U1 - Grade 11 Advanced Functions<br>Prerequisite(s): MCR3U1 or MCR3U7</html>", "<html>MCV4U1 - Grade 12 Calculus and Vectors<br>Prerequisite(s): MHF4U1 or MHF4U7</html>", ""},
    {"Science", "<html>SNC1D1 - Grade 9 Science<br>Prerequisite(s): none</html>", "<html>SNC2D1 - Grade 10 Science<br>Prerequisite(s): SNC1D1 or SNC1D7</html>", "", "", "", ""},
    {"Computer Studies", "", "<html>ICS2O1 - Grade 10 Introduction to Computer Studies<br>Prerequisite(s): none</html>", "<html>ICS3U1 - Grade 11 Introduction to Computer Science<br>Prerequisite(s): none</html>", "<html>ICS4U1 - Grade 12 Computer Science<br>Prerequisite(s): ICS3U1 or ICS3U7</html>", "", ""},
    {"Chemistry", "", "", "<html>SCH3U1 - Grade 11 Chemistry<br>Prerequisite(s): SNC2D1 or SNC2D7</html>", "<html>SCH4U1 - Grade 12 Chemistry<br>Prerequisite(s): SCH3U1 or SCH3U7</html>", "", ""},
    {"Physics", "", "", "<html>SPH3U1 - Grade 11 Physics<br>Prerequisite(s): SNC2D1 or SNC2D7</html>", "<html>SPH4U1 - Grade 12 Physics<br>Prerequisite(s): SPH3U1 or SPH3U7</html>", "", ""},
    {"Biology", "", "", "<html>SBI3U1 - Grade 11 Biology<br>Prerequisite(s): SNC2D1 or SNC2D7</html>", "<html>SBI4U1 - Grade 12 Biology<br>Prerequisite(s): SBI3U1 or SBI3U7</html>", "", ""},
    {"French", "<html>FSF1D1 - Grade 9 Core French<br>Prerequisite(s): none</html>", "<html>FSF2D1 - Grade 10 Core French<br>Prerequisite(s): FSF1D1 or FSF1D7</html>", "<html>FSF3U1 - Grade 11 Core French<br>Prerequisite(s): FSF2D1 or FSF2D7</html>", "<html>FSF4U1 - Grade 12 Core French<br>Prerequisite(s): FSF3U1 or FSF3U7</html>", "", ""},
    {"Gym (Males)", "<html>PPL1OM - Grade 9 Healthy Active Living Education (M)<br>Prerequisite(s): none</html>", "<html>PPL2OM - Grade 10 Healthy Active Living Education (M)<br>Prerequisite(s): none</html>", "<html>PPL3OM - Grade 11 Healthy Active Living Education (M)<br>Prerequisite(s): none</html>", "<html>PPL4OM - Grade 12 Healthy Active Living Education (M)<br>Prerequisite(s): none</html>", "", ""},
    {"Gym (Females)", "<html>PPL1OF - Grade 9 Healthy Active Living Education (F)<br>Prerequisite(s): none</html>", "<html>PPL2OF - Grade 10 Healthy Active Living Education (F)<br>Prerequisite(s): none</html>", "<html>PPL3OF - Grade 11 Healthy Active Living Education (F)<br>Prerequisite(s): none</html>", "<html>PPL4OF - Grade 12 Healthy Active Living Education (F)<br>Prerequisite(s): none</html>", "", ""},
    {"World Studies", "<html>CGC1D1 - Grade 9 Issues in Canadian Geography<br>Prerequisite(s): none</html>", "<html>CHC2D1 - Grade 10 Canadian History since World War I<br>Prerequisite(s): none</html>", "<html>CLU3M1 - Grade 11 Understanding Canadian Law<br>Prerequisite(s): CHC2D1</html>", "<html>CLN4U1 - Grade 12 Canadian and International Law<br>Prerequisite(s): CLU3M1</html>", "", ""},
    {"Art", "<html>NAC101 - Grade 9 Expressing Aboriginal Cultures<br>Prerequisite(s): none</html>", "<html>AVI2O1 - Grade 10 Visual
```

```
Arts<br>Prerequisite(s): none</html>", "<html>AVI3M1 - Grade 11 Visual Arts<br>Prerequisite(s): AVI2O1</html>", "<html>AVI4M1 - Grade 12 Visual Arts<br>Prerequisite(s): AVI3M1</html>", "", ""},
    {"Business Studies", "<html>BTT1O1 - Grade 9 Information and Communication Technology in Business<br>Prerequisite(s): none</html>", "<html>BBI2O1 - Grade 10 Introduction to Business<br>Prerequisite(s): none</html>", "<html>BAF3M1 - Grade 11 Financial Accounting Fundamentals<br>Prerequisite(s): none</html>", "<html>BAT4M1 - Grade 12 Financial Accounting Principles<br>Prerequisite(s): BAF3M1</html>", "<html>BBB4M1 - Grade 12 International Business Fundamentals<br>Prerequisite(s): BAF3M1</html>", ""},
    {"Technological Education", "<html>TIJ1O1 - Grade 9 Exploring Technologies<br>Prerequisite(s): none</html>", "<html>TEJ2O1 - Grade 10 Computer Technology<br>Prerequisite(s): none</html>", "<html>TEJ3M1 - Grade 11 Computer Engineering Technology<br>Prerequisite(s): none</html>", "<html>TEJ4M1 - Grade 12 Computer Engineering Technology<br>Prerequisite(s): TEJ3M1</html>", "", ""},
    {"", "", "", "", "", "", ""},
    {"English (IB)", "<html>ENG1D7 - Grade 9 English (IB)<br>Prerequisite(s): none</html>", "<html>ENG2D7 - Grade 10 English (IB)<br>Prerequisite(s): ENG1D7</html>", "<html>ENG3U7 - Grade 11 English (IB)<br>Prerequisite(s): ENG2D7</html>", "<html>ENG4U7 - Grade 12 English (IB)<br>Prerequisite(s): ENG3U7</html>", "<html>ETS4U7 - Grade 12 Studies in Literature (IB)<br>Prerequisite(s): ENG3U7</html>", ""},
    {"Math (IB)", "<html>MPM1D7 - Grade 9 Principles of Mathematics (IB)<br>Prerequisite(s): none</html>", "<html>MPM2D7 - Grade 10 Principles of Mathematics (IB)<br>Prerequisite(s): MPM1D7</html>", "<html>MCR3U7 - Grade 11 Functions (IB)<br>Prerequisite(s): MPM2D7</html>", "<html>MDM4U7 - Grade 12 Mathematics of Data Management (IB)<br>Prerequisite(s): MCR3U7</html>", "<html>MHF4U7 - Grade 12 Advanced Functions (IB)<br>Prerequisite(s): MCR3U7</html>", "<html>MCV4U7 - Grade 12 Calculus and Vectors (IB)<br>Prerequisite(s): MCR3U7</html>"},
    {"Science (IB)", "<html>SNC1D7 - Grade 9 Science (IB)<br>Prerequisite(s): none</html>", "<html>SNC2D7 - Grade 10 Science (IB)<br>Prerequisite(s): SNC1D7</html>", "", "", "", ""},
    {"Computer Studies (IB)", "", "<html>ICS2O7 - Grade 10 Introduction to Computer Studies (IB)<br>Prerequisite(s): none</html>", "<html>ICS3U7 - Grade 11 Introduction to Computer Science (IB)<br>Prerequisite(s): ICS2O7</html>", "<html>ICS4U7 - Grade 12 Computer Science (IB)<br>Prerequisite(s): ICS3U7</html>", "", ""},
    {"Chemistry (IB)", "", "", "<html>SCH3U7 - Grade 11 Chemistry (IB)<br>Prerequisite(s): SNC2D7</html>", "<html>SCH4UP - Grade 12 Chemistry (IB)<br>Prerequisite(s): SCH3U7</html>", "", ""},
    {"Physics (IB)", "", "", "<html>SPH3U7 - Grade 11 Physics (IB)<br>Prerequisite(s): SNC2D7</html>", "<html>SPH4U7 - Grade 12 Physics (IB)<br>Prerequisite(s): SPH3U7</html>", "", ""},
    {"Biology (IB)", "", "", "<html>SBI3U7 - Grade 11 Biology (IB)<br>Prerequisite(s): SNC2D7</html>", "<html>SBI4U7 - Grade 12 Biology (IB)<br>Prerequisite(s): SBI3U7</html>", "", ""},
    {"French (IB)", "<html>FSF1D7 - Grade 9 Core French (IB)<br>Prerequisite(s): none</html>", "<html>FSF2D7 - Grade 10 Core French (IB)<br>Prerequisite(s): FSF1D7</html>", "<html>FSF3U7 - Grade 11 Core French (IB)<br>Prerequisite(s): FSF2D7</html>", "<html>FSF4U7 - Grade 12 Core French (IB)<br>Prerequisite(s): FSF3U7</html>", "", ""},
    {"Geography (IB)", "<html>CGC1D7 - Grade 9 Issues in Canadian Geography (IB)<br>Prerequisite(s): none</html>", "", "<html>CGF3M7 - Grade 11 Forces of Nature: Physical Processes and Disasters (IB)<br>Prerequisite(s): CGC1D7</html>", "<html>CGW4U7 - Grade 12 World Issues: A Geographic Analysis (IB)<br>Prerequisite(s): CGF3M7</html>", "", ""},
    {"History & Economics (IB)", "", "<html>CHC2D7 - Grade 10 Canadian History since World War I (IB)<br>Prerequisite(s): none</html>", "<html>CIE3M7 - Grade 11 The Individual & the Economy (IB)<br>Prerequisite(s): CHC2D7</html>", "<html>CIA4U7 - Grade 12 Analysing Current Economic Issues (IB)<br>Prerequisite(s): CIE3M7</html>", "<html>BBB4M7 - Grade 12 Economics (IB HL)<br>Prerequisite(s): CIA4U7</html>", ""},
    {"Theory Of Knowledge (IB)", "", "", "", "<html>HZT4U7 - Grade 12 Philosophy: Questions and Theories (IB)<br>Prerequisite(s): ENG2D7</html>", "", ""},
    {"", "", "", "", "", "", ""},
    {"Spares", "", "", "", "Grade 12 Spare #1", "Grade 12 Spare #2", ""}};

    private Button[][] buttonArr = new Button[courses.length][courses[0].length];   // The buttons necessary in displaying all of the courses

    private MyTextLabel infoLabel;

    /**
```

```java
     * Initializes the values for a "ViewCoursesPage" object
     */
    public ViewCoursesPage() {
        super ("View All Possible Courses Offered");

        setSize(1100, 800);
        setVisible(false);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        JPanel canvas = new JPanel();
        canvas.setLayout(new GridLayout(0, 7));

        // This block of code sets up the top row of buttons for this page
        buttonArr[0][0] = new Button(courses[0][0]);
        canvas.add(buttonArr[0][0]);
        for (int i = 1; i < courses[0].length; i ++) {
            buttonArr[0][i] = new Button(courses[0][i]);
            buttonArr[0][i].setBackground(Color.CYAN);
            canvas.add(buttonArr[0][i]);
        }

        // Setting up all other buttons in the page
        for (int i = 1; i < courses.length; i ++) {
            for (int j = 0; j < courses[i].length; j ++) {
                if (j == 0) {
                    if (courses[i][j].length() == 0) {
                        buttonArr[i][j] = new Button("");
                        buttonArr[i][j].setBackground(Color.BLACK);
                    } else {
                        buttonArr[i][j] = new Button(courses[i][j]);
                        buttonArr[i][j].setBackground(new Color(222,184,135));
                    }
                } else {
                    if (courses[i][j].length() == 0) {
                        buttonArr[i][j] = new Button("");
                        buttonArr[i][j].setBackground(Color.BLACK);
                    }
                    else {
                        if (i == courses.length - 1) buttonArr[i][j] = new Button(courses[i][j]);
                        else buttonArr[i][j] = new Button(courses[i][j].substring(6, 12));
                    }

                    buttonArr[i][j].addActionListener(this);
                }
                canvas.add(buttonArr[i][j]);
```

```
            }
        }

        infoLabel = new MyTextLabel("Click on a course code to view its course description.", SwingConstants.CENTER);
        infoLabel.setTextLabel(400, 40, 0, 0, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 14));

        backButton = new MyButton("Back");
        backButton.addActionListener(this);

        add(infoLabel, BorderLayout.NORTH);
        add(backButton, BorderLayout.WEST);
        add(canvas);
    }

    public void paint(Graphics g) {
        super.paintComponents(g);
        repaint();
    }

    public void actionPerformed(ActionEvent e) {
        Component button = (Component) e.getSource();

        if (button.equals(backButton)) {   // User wants to go back to the main menu page
            setVisible(false);
            DriverClass.displayMainMenu();
        } else {   // User has clicked on one of the other buttons to possibly view its course description and prerequisites
            for (int i = 0; i < courses.length; i ++) {
                for (int j = 0; j < courses[i].length; j ++) {
                    if (button.equals(buttonArr[i][j])) {
                        if (courses[i][j].length() == 0) JOptionPane.showMessageDialog(this, "That is not an appropriate course.", "Warning", JOptionPane.WARNING_MESSAGE);
                        else JOptionPane.showMessageDialog(this, courses[i][j]);
                    }
                }
            }
        }
    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import driverProgram.*;
import linkedList.*;
import objects.*;

public class AddATeacherPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private MyTextLabel necessaryInfoTextLabel, nameTextLabel;

    private MyTextField nameTextField;

    private MyButton backButton, addTeacherButton;

    /**
     * Initializes the values for a "AddATeacherPage" object
     * @param canvas The canvas for the "Add A Teacher" page
     */
    public AddATeacherPage(JPanel canvas) {
        super ("Add A Teacher");
        add(canvas);

        setSize(600, 830);
        setVisible(false);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        necessaryInfoTextLabel = new MyTextLabel("<html>Note: Fields marked with '*' means<br>that they are necessary to fill out.</html>", SwingConstants.CENTER);
        necessaryInfoTextLabel.setTextLabel(280, 80, 155, 40, Color.BLACK, Color.ORANGE, new Font("Arial", Font.BOLD, 16));

        nameTextLabel = new MyTextLabel("(*) Full name:", SwingConstants.CENTER);
        nameTextLabel.setTextLabel(125, 20, 170, 190, Color.BLACK, Color.YELLOW, new Font("Arial", Font.BOLD, 18));

        nameTextField = new MyTextField();
        nameTextField.setTextField(250, 30, 170, 210);

        addTeacherButton = new MyButton("Add this teacher");
        addTeacherButton.setButton(130, 40, 227, 260);
        addTeacherButton.addActionListener(this);

```

```java
        backButton = new MyButton("Back");
        backButton.setButton(150, 50, 210, 735);
        backButton.addActionListener(this);

        canvas.add(necessaryInfoTextLabel);

        canvas.add(nameTextLabel);
        canvas.add(nameTextField);

        canvas.add(addTeacherButton);

        canvas.add(backButton);
    }

    public void paint(Graphics g) {
        super.paintComponents(g);
        repaint();
    }

    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();

        if (button.equals(backButton)) {   // User wants to go back to the main menu page
            nameTextField.setText("");

            setVisible(false);
            DriverClass.displayMainMenu();
        } else if (button.equals(addTeacherButton)) {   // User has chosen to add a teacher to this database program
            String name = nameTextField.getText().trim();
            String[] nameSplit = name.split(" ");

            if (name.equals("")) {   // If user did not enter anything in the "name" text field
                JOptionPane.showMessageDialog(this, "Please fill out all necessary fields.", "Warning", JOptionPane.WARNING_MESSAGE);
            }
            else if (nameSplit.length == 1) {   // If the user did not enter a last name for the teacher
                JOptionPane.showMessageDialog(this, "Please enter a last name for this teacher.", "Warning", JOptionPane.WARNING_MESSAGE);
            }
            else {   // User has successfully filled in all necessary fields correctly
                LinkedListTeacher teachers = DriverClass.getTeachers();
                int id;

                // Making sure that a unique ID is generated for the current teacher being added.
                while (true) {
                    id = Person.generateID();
                    boolean flag = true;
                    for (int i = 0; i < teachers.getSize(); i ++) {
                        if (teachers.get(0, teachers.getHead(), i).getID() == id) {
```

```java
                        flag = false;
                        break;
                    }
                }
                if (flag) break;
            }

            // Setting the name, id, and email of the added teacher.
            Teacher temp = new Teacher(name, id);
            temp.generateEmail();

            // Adding the teacher to the "teachers" linkedlist in the main menu.
            DriverClass.addTeacher(temp);

            // Updating the history page when a new teacher is added.
            DriverClass.addEventToHistory(name + " has been added as a teacher in this school.", true);

            // Adding the teacher's info to the "TeachersInfo.txt" RandomAccessFile.
            try {
                RandomAccessFile teachersInfo = new RandomAccessFile("TeachersInfo.txt", "rw");
                teachersInfo.seek(teachersInfo.length());
                teachersInfo.writeBytes("Add_" + name + "_" + id + "_" + temp.getEmail() + "\n");
                teachersInfo.close();
            } catch (IOException e1) {
                e1.printStackTrace();
            }

            // Clearing the screen and informing the user that the teacher has been successfully added.
            nameTextField.setText("");
            JOptionPane.showMessageDialog(this, "Teacher added successfully.");
        }
    }
}
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import driverProgram.*;
import linkedList.*;
import objects.*;

public class EditCurrentTeachersPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private MyTextLabel choiceLabel;

    private static LinkedListTeacher teachersList;   // Linked list of teachers who are currently in the database

    private MyButton sortTeachersAlphabeticallyButton, removeThisTeacherButton;
    private MyButton backButton;

    private Choice teachersMenu;

    private static boolean sorted;   // Stores whether the list of teachers is currently sorted alphabetically or not

    /**
     * Initializes the values for a "EditCurrentTeachersPage" object
     */
    public EditCurrentTeachersPage() {
        super ("Edit current teachers");

        setLayout(null);
        setSize(800, 800);
        setVisible(false);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        sorted = false;

        teachersList = new LinkedListTeacher();

        choiceLabel = new MyTextLabel("Select a teacher below", SwingConstants.CENTER);
        choiceLabel.setTextLabel(400, 40, getWidth() / 2 - 200, 20, Color.BLACK, null, new Font("Arial", Font.BOLD, 30));

        teachersMenu = new Choice();
        teachersMenu.setBounds(getWidth() / 2 - 100, 75, 200, 40);

        for (int i = 0; i < teachersList.getSize(); i ++) {
```

```
            teachersMenu.add(teachersList.get(0, teachersList.getHead(), i).getName());
        }

        backButton = new MyButton("Back");
        backButton.setButton(160, 50, 0, 713);
        backButton.addActionListener(this);

        sortTeachersAlphabeticallyButton = new MyButton("<html>Sort list of teacher(s)<br>alphabetically</html>");
        sortTeachersAlphabeticallyButton.setButton(190, 40, 90, 65);
        sortTeachersAlphabeticallyButton.setBackground(Color.CYAN);
        sortTeachersAlphabeticallyButton.addActionListener(this);

        removeThisTeacherButton = new MyButton("Remove this teacher");
        removeThisTeacherButton.setButton(190, 30, getWidth() / 2 + 120, 70);
        removeThisTeacherButton.setBackground(Color.RED);
        removeThisTeacherButton.addActionListener(this);

        add(choiceLabel);
        add(teachersMenu);

        add(sortTeachersAlphabeticallyButton);
        add(removeThisTeacherButton);

        add(backButton);

        sortTeachersAlphabeticallyButton.setVisible(false);
        removeThisTeacherButton.setVisible(false);
    }

    /**
     * Receiving the list of Teachers in the database from the main menu page and setting up the page layout from this list of Teachers
     * @param list The list of Teachers currently in the database
     */
    public void setTeachersList(LinkedListTeacher list) {
        teachersList = list;

        remove(teachersMenu);

        teachersMenu = new Choice();
        teachersMenu.setBounds(getWidth() / 2 - 100, 75, 200, 40);

        for (int i = 0; i < teachersList.getSize(); i ++) {
            teachersMenu.add((i + 1) + ". " + teachersList.get(0, teachersList.getHead(), i).getName());
        }

        add(teachersMenu);

```

```java
        if (teachersList.getSize() > 0) {   // List of teachers is not empty, so show the "Sort alphabetically" and "Remove this teacher" button
            sortTeachersAlphabeticallyButton.setVisible(true);
            removeThisTeacherButton.setVisible(true);
        } else {   // List of teachers is empty, so hide the "Sort alphabetically" and "Remove this teacher" button
            sortTeachersAlphabeticallyButton.setVisible(false);
            removeThisTeacherButton.setVisible(false);
        }
    }

    /**
     * Returns true if the list of Teachers on this page is currently sorted alphabetically
     * @return "True" if the list of Teachers on this page is currently sorted alphabetically, "false" otherwise
     */
    public static boolean wasSorted() {
        return sorted;
    }

    /**
     * Receives a string from the list of Teachers and returns the Teacher associated with that string
     * @param str The selected string from the Teacher's pop-up menu of choices
     * @return The Teacher associated with the selected string in the Teacher's pop-up menu of choices
     */
    private Teacher findTeacher(String str) {
        int idx = Character.getNumericValue(str.charAt(0)) - 1;
        Teacher tch = teachersList.get(0, teachersList.getHead(), idx);
        return tch;
    }

    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();

        if (button.equals(backButton)) {   // User wants to go back to the main menu page
            setVisible(false);

            teachersList.sortAlphabetically(false);
            DriverClass.displayMainMenu();
        } else if (button.equals(sortTeachersAlphabeticallyButton)) {   // User wants to either sort the list of teachers alphabetically or unsort it back into its original order
            if (!sorted) {   // Current list of teachers is not sorted alphabetically, so sort the list of teachers alphabetically
                teachersList.sortAlphabetically(true);
                setTeachersList(teachersList);
                sorted = true;

                sortTeachersAlphabeticallyButton.setBackground(Color.ORANGE);
                sortTeachersAlphabeticallyButton.setText("<html>Unsort the list back into<br>original order</html>");
            } else {   // Current list of teachers is sorted alphabetically, so unsort the list of teachers back into its original order
```

```
                teachersList.sortAlphabetically(false);
                setTeachersList(teachersList);
                sorted = false;

                sortTeachersAlphabeticallyButton.setBackground(Color.CYAN);
                sortTeachersAlphabeticallyButton.setText("<html>Sort list of teacher(s)<br>alphabetically</html>");
            }
        } else if (button.equals(removeThisTeacherButton)) {   // User wants to remove the currently selected teacher
            Teacher tch = findTeacher(teachersMenu.getSelectedItem());

            // Removing the teacher from the "teachers" linked list in the main menu.
            teachersList.remove(tch);
            setTeachersList(teachersList);
            teachersList.updateOriginalList(tch);

            // Adding an event to the history page about the teacher being removed.
            DriverClass.addEventToHistory(tch.getName() + " has been removed as a teacher in this school.", true);

            // Inform the "TeachersInfo.txt" RandomAccessFile about removing the selected teacher
            try {
                RandomAccessFile teachersInfo = new RandomAccessFile("TeachersInfo.txt", "rw");
                teachersInfo.seek(teachersInfo.length());
                teachersInfo.writeBytes("Remove_" + tch.getID() + "\n");
                teachersInfo.close();
            } catch (FileNotFoundException e1) {
                e1.printStackTrace();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }
    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import driverProgram.*;
import linkedList.*;
import objects.*;

public class ContactInformationPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private LinkedListStudent studentsList;   // Linked list of Students who are currently in the database
    private LinkedListTeacher teachersList;   // Linked list of Teachers who are currently in the database

    private Choice studentsMenu, teachersMenu;

    private MyTextLabel infoLabel, studentsListLabel, teachersListLabel;

    private MyTextLabel studentNameLabel, studentNameLabelAns, studentIDLabel, studentIDLabelAns, studentGradeLabel, studentGradeLabelAns, studentIsInIBLabel, studentIsInIBLabelAns, studentEmailLabel, studentEmailLabelAns;
    private MyTextLabel teacherNameLabel, teacherNameLabelAns, teacherIDLabel, teacherIDLabelAns, teacherEmailLabel, teacherEmailLabelAns;

    private MyButton viewStudentInfoButton, viewTeacherInfoButton, backButton;

    /**
     * Initializes the values for a "ContactInformationPage" object
     */
    public ContactInformationPage() {
        super ("Contact Information Page");

        setLayout(null);
        setSize(800, 800);
        setVisible(false);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        studentsList = new LinkedListStudent();

        infoLabel = new MyTextLabel("Contact Information Page", SwingConstants.CENTER);
        infoLabel.setTextLabel(400, 40, getWidth() / 2 - 200, 20, Color.BLACK, null, new Font("Arial", Font.BOLD, 30));

        studentsListLabel = new MyTextLabel("Students", SwingConstants.CENTER);
        studentsListLabel.setTextLabel(200, 60, 100, 75, Color.BLACK, null, new Font("Arial", Font.BOLD, 24));

        studentsMenu = new Choice();
        studentsMenu.setBounds(100, 135, 200, 40);
```

```java

        viewStudentInfoButton = new MyButton("View this student's info");
        viewStudentInfoButton.setButton(200, 40, 100, 160);
        viewStudentInfoButton.addActionListener(this);

        teachersListLabel = new MyTextLabel("Teachers", SwingConstants.CENTER);
        teachersListLabel.setTextLabel(200, 60, 500, 75, Color.BLACK, null, new Font("Arial", Font.BOLD, 24));

        teachersMenu = new Choice();
        teachersMenu.setBounds(500, 135, 200, 40);

        viewTeacherInfoButton = new MyButton("View this teacher's info");
        viewTeacherInfoButton.setButton(200, 40, 500, 160);
        viewTeacherInfoButton.addActionListener(this);

        studentNameLabel = new MyTextLabel("Name", SwingConstants.CENTER);
        studentNameLabel.setTextLabel(200, 20, 100, 220, Color.BLACK, null, new Font("Arial", Font.BOLD, 18));

        studentNameLabelAns = new MyTextLabel("", SwingConstants.CENTER);
        studentNameLabelAns.setTextLabel(200, 20, 100, 240, Color.BLACK, null, new Font("Arial", Font.PLAIN, 16));

        studentIDLabel = new MyTextLabel("ID", SwingConstants.CENTER);
        studentIDLabel.setTextLabel(200, 20, 100, 300, Color.BLACK, null, new Font("Arial", Font.BOLD, 18));

        studentIDLabelAns = new MyTextLabel("", SwingConstants.CENTER);
        studentIDLabelAns.setTextLabel(200, 20, 100, 320, Color.BLACK, null, new Font("Arial", Font.PLAIN, 16));

        studentGradeLabel = new MyTextLabel("Grade", SwingConstants.CENTER);
        studentGradeLabel.setTextLabel(200, 20, 100, 380, Color.BLACK, null, new Font("Arial", Font.BOLD, 18));

        studentGradeLabelAns = new MyTextLabel("", SwingConstants.CENTER);
        studentGradeLabelAns.setTextLabel(200, 20, 100, 400, Color.BLACK, null, new Font("Arial", Font.PLAIN, 16));

        studentIsInIBLabel = new MyTextLabel("In pre-IB or IB?", SwingConstants.CENTER);
        studentIsInIBLabel.setTextLabel(200, 20, 100, 460, Color.BLACK, null, new Font("Arial", Font.BOLD, 18));

        studentIsInIBLabelAns = new MyTextLabel("", SwingConstants.CENTER);
        studentIsInIBLabelAns.setTextLabel(200, 20, 100, 480, Color.BLACK, null, new Font("Arial", Font.PLAIN, 16));

        studentEmailLabel = new MyTextLabel("Email", SwingConstants.CENTER);
        studentEmailLabel.setTextLabel(200, 20, 100, 540, Color.BLACK, null, new Font("Arial", Font.BOLD, 18));

        studentEmailLabelAns = new MyTextLabel("", SwingConstants.CENTER);
        studentEmailLabelAns.setTextLabel(300, 20, 50, 560, Color.BLACK, null, new Font("Arial", Font.PLAIN, 16));

        teacherNameLabel = new MyTextLabel("Name", SwingConstants.CENTER);
        teacherNameLabel.setTextLabel(200, 20, 500, 220, Color.BLACK, null, new Font("Arial", Font.BOLD, 18));
```

```java

        teacherNameLabelAns = new MyTextLabel("", SwingConstants.CENTER);
        teacherNameLabelAns.setTextLabel(200, 20, 500, 240, Color.BLACK, null, new Font("Arial", Font.PLAIN, 16));

        teacherIDLabel = new MyTextLabel("ID", SwingConstants.CENTER);
        teacherIDLabel.setTextLabel(200, 20, 500, 300, Color.BLACK, null, new Font("Arial", Font.BOLD, 18));

        teacherIDLabelAns = new MyTextLabel("", SwingConstants.CENTER);
        teacherIDLabelAns.setTextLabel(200, 20, 500, 320, Color.BLACK, null, new Font("Arial", Font.PLAIN, 16));

        teacherEmailLabel = new MyTextLabel("Email", SwingConstants.CENTER);
        teacherEmailLabel.setTextLabel(200, 20, 500, 380, Color.BLACK, null, new Font("Arial", Font.BOLD, 18));

        teacherEmailLabelAns = new MyTextLabel("", SwingConstants.CENTER);
        teacherEmailLabelAns.setTextLabel(300, 20, 450, 400, Color.BLACK, null, new Font("Arial", Font.PLAIN, 16));

        backButton = new MyButton("Back");
        backButton.setButton(160, 50, 0, 713);
        backButton.addActionListener(this);

        add(infoLabel);

        add(studentsListLabel);
        add(studentsMenu);
        add(viewStudentInfoButton);

        add(teachersListLabel);
        add(teachersMenu);
        add(viewTeacherInfoButton);

        add(studentNameLabel);
        add(studentNameLabelAns);
        add(studentIDLabel);
        add(studentIDLabelAns);
        add(studentGradeLabel);
        add(studentGradeLabelAns);
        add(studentIsInIBLabel);
        add(studentIsInIBLabelAns);
        add(studentEmailLabel);
        add(studentEmailLabelAns);

        add(teacherNameLabel);
        add(teacherNameLabelAns);
        add(teacherIDLabel);
        add(teacherIDLabelAns);
        add(teacherEmailLabel);
        add(teacherEmailLabelAns);
```

```java

        hideLabels();

        add(backButton);
    }

    /**
     * Hide all labels responsible for displaying information about a Student and Teacher
     */
    public void hideLabels() {
        studentNameLabel.setVisible(false);
        studentNameLabelAns.setVisible(false);
        studentIDLabel.setVisible(false);
        studentIDLabelAns.setVisible(false);
        studentGradeLabel.setVisible(false);
        studentGradeLabelAns.setVisible(false);
        studentIsInIBLabel.setVisible(false);
        studentIsInIBLabelAns.setVisible(false);
        studentEmailLabel.setVisible(false);
        studentEmailLabelAns.setVisible(false);

        teacherNameLabel.setVisible(false);
        teacherNameLabelAns.setVisible(false);
        teacherIDLabel.setVisible(false);
        teacherIDLabelAns.setVisible(false);
        teacherEmailLabel.setVisible(false);
        teacherEmailLabelAns.setVisible(false);
    }

    /**
     * Show all labels responsible for displaying information about a Student
     */
    public void showStudentLabels() {
        studentNameLabel.setVisible(true);
        studentNameLabelAns.setVisible(true);
        studentIDLabel.setVisible(true);
        studentIDLabelAns.setVisible(true);
        studentGradeLabel.setVisible(true);
        studentGradeLabelAns.setVisible(true);
        studentIsInIBLabel.setVisible(true);
        studentIsInIBLabelAns.setVisible(true);
        studentEmailLabel.setVisible(true);
        studentEmailLabelAns.setVisible(true);
    }

    /**
     * Show all labels responsible for displaying information about a Teacher
```

```
188     */
189    public void showTeacherLabels() {
190        teacherNameLabel.setVisible(true);
191        teacherNameLabelAns.setVisible(true);
192        teacherIDLabel.setVisible(true);
193        teacherIDLabelAns.setVisible(true);
194        teacherEmailLabel.setVisible(true);
195        teacherEmailLabelAns.setVisible(true);
196    }
197
198    /**
199     * Set the list of Students and Teachers and display them on the page. These lists are automatically sorted alphabetically
200     * @param sList The list of Students currently in the database
201     * @param tList The list of Teachers currently in the database
202     */
203    public void setLists(LinkedListStudent sList, LinkedListTeacher tList) {
204        remove(studentsMenu);
205        remove(teachersMenu);
206
207        studentsMenu = new Choice();
208        studentsMenu.setBounds(100, 135, 200, 40);
209
210        teachersMenu = new Choice();
211        teachersMenu.setBounds(500, 135, 200, 40);
212
213        studentsList = sList;
214        teachersList = tList;
215
216        // Sort the list of students alphabetically and display this list
217        studentsList.sortAlphabetically(true);
218        for (int i = 0; i < studentsList.getSize(); i ++) {
219            studentsMenu.add((i + 1) + ". " + studentsList.get(0, studentsList.getHead(), i).getName());
220        }
221
222        // Sort the list of teachers alphabetically and display this list
223        teachersList.sortAlphabetically(true);
224        for (int i = 0; i < teachersList.getSize(); i ++) {
225            teachersMenu.add((i + 1) + ". " + teachersList.get(0, teachersList.getHead(), i).getName());
226        }
227
228        add(studentsMenu);
229        add(teachersMenu);
230    }
231
232    /**
233     * Receives a string from the list of Students and returns the Student associated with that string
234     * @param str The selected string from the Student's pop-up menu of choices
```

```java
 * @return The Student associated with the selected string in the Student's pop-up menu of choices
 */
private Student findStudent(String str) {
    int idx = Character.getNumericValue(str.charAt(0)) - 1;
    Student stu = studentsList.get(0, studentsList.getHead(), idx);
    return stu;
}

/**
 * Receives a string from the list of teachers and return the teacher associated with that string
 * @param str The selected string from the Teacher's pop-up menu of choices
 * @return The Teacher associated with the selected string in the Teacher's pop-up menu of choices
 */
private Teacher findTeacher(String str) {
    int idx = Character.getNumericValue(str.charAt(0)) - 1;
    Teacher tch = teachersList.get(0, teachersList.getHead(), idx);
    return tch;
}

public void actionPerformed(ActionEvent e) {
    MyButton button = (MyButton) e.getSource();

    if (button.equals(backButton)) {   // User wants to go back to the main menu page
        studentsList.sortAlphabetically(false);
        teachersList.sortAlphabetically(false);

        hideLabels();

        setVisible(false);
        DriverClass.displayMainMenu();
    } else if (button.equals(viewStudentInfoButton)) {   // User wants to view the selected student's info
        if (studentsMenu.getSelectedItem() != null) {
            Student stu = findStudent(studentsMenu.getSelectedItem());

            studentNameLabelAns.setText(stu.getName());
            studentIDLabelAns.setText(String.valueOf(stu.getID()));
            studentGradeLabelAns.setText(String.valueOf(stu.getGrade()));
            studentIsInIBLabelAns.setText(stu.isInIB() ? "Yes" : "No");
            studentEmailLabelAns.setText(stu.getEmail());

            showStudentLabels();
        } else JOptionPane.showMessageDialog(this, "There is no student selected.", "Warning", JOptionPane.WARNING_MESSAGE);
    } else if (button.equals(viewTeacherInfoButton)) {   // User wants to view the selected teacher's info
        if (teachersMenu.getSelectedItem() != null) {
            Teacher tch = findTeacher(teachersMenu.getSelectedItem());

            teacherNameLabelAns.setText(tch.getName());
```

```
                teacherIDLabelAns.setText(String.valueOf(tch.getID()));
                teacherEmailLabelAns.setText(tch.getEmail());

                showTeacherLabels();
            } else JOptionPane.showMessageDialog(this, "There is no teacher selected.", "Warning", JOptionPane.WARNING_MESSAGE);
        }
    }
}
```

```java
package mainMenu;

import java.awt.event.*;
import java.io.*;
import javax.swing.*;
import javax.swing.border.*;
import driverProgram.*;
import objects.*;

public class HistoryPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private JTextArea display;   // This text area will display the history of actions

    private MyButton backButton, clearHistoryButton;

    /**
     * Initializes the values for a "HistoryPage" object
     */
    public HistoryPage() {
        super("History Page");

        JPanel canvas = new JPanel();
        canvas.setBorder(new TitledBorder (new EtchedBorder(), "                    History                    "));

        display = new JTextArea(16, 58);
        display.setEditable(false);

        JScrollPane scroll = new JScrollPane(display);
        scroll.setVerticalScrollBarPolicy (ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

        backButton = new MyButton("Back");
        backButton.addActionListener(this);

        clearHistoryButton = new MyButton("Clear History");
        clearHistoryButton.addActionListener(this);

        canvas.add(backButton);
        canvas.add(scroll);
        canvas.add(clearHistoryButton);
        add(canvas);
        pack();

        setVisible(false);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
```

```java

    /**
     * Adding an event to the history page
     * @param str The string representing the event which will be added to the history page
     * @param addToRandomAccessFile "True" if this event should be added to the "HistoryInfo.txt" RandomAccessFile, "false" otherwise
     */
    public void addText(String str, boolean addToRandomAccessFile) {
        display.append(str + "\n");
        pack();

        if (addToRandomAccessFile) {
            // Adding the details of the current event to the "HistoryInfo.txt" RandomAccessFile.
            try {
                RandomAccessFile historyInfo = new RandomAccessFile("HistoryInfo.txt", "rw");
                historyInfo.seek(historyInfo.length());
                historyInfo.writeBytes(str + "\n");
                historyInfo.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * Clearing the history of all events
     * @param addToRandomAccessFile "True" if the "HistoryInfo.txt" RandomAccessFile should clear all events from its history, "false" otherwise
     */
    public void clearHistory(boolean addToRandomAccessFile) {
        display.setText("");

        if (addToRandomAccessFile) {
            // Delete all of the content in the "HistoryInfo.txt" RandomAccessFile.
            try {
                RandomAccessFile historyInfo = new RandomAccessFile("HistoryInfo.txt", "rw");
                historyInfo.setLength(0);
                historyInfo.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

```

```java
    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();

        if (button.equals(backButton)) {   // User wants to go back to the main menu page
            setVisible(false);
            DriverClass.displayMainMenu();
        } else if (button.equals(clearHistoryButton)) {   // User wants to clear the details of all events that have occurred
            clearHistory(true);
        }
    }
}
```

```java
package mainMenu;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;
import driverProgram.*;
import objects.*;

// This page is just for displaying all image citations (in MLA format) that have been used as background images for this program
public class ImageCitationsPage extends JFrame implements ActionListener {
    private static final long serialVersionUID = 1L;

    private MyButton backButton;

    /**
     * Initializes the values for a "ImageCitationsPage" object
     */
    public ImageCitationsPage() {
        super("Image Citations Page");

        Canvas canvas = new Canvas();

        add(canvas);

        setSize(1000, 600);
        setVisible(false);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        backButton = new MyButton("Back");
        backButton.setButton(100, 40, 0, 560);
        backButton.addActionListener(this);

        add(backButton, BorderLayout.WEST);
    }

    /**
     *
     * @author Austin
     * Canvas for the "Image Citations" page
     */
    static class Canvas extends JPanel {
        private static final long serialVersionUID = 1L;
```

```java

        private BufferedImage background;

        /**
         * Initializes the values for a "Canvas" object
         */
        public Canvas() {
            try {
                background = ImageIO.read(this.getClass().getResource("/imageCitationsBackground.png"));
            } catch (IOException e) {
                e.printStackTrace();
            }
            setLayout(null);
        }

        /**
         * Setting up the background image for this canvas
         */
        public void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.drawImage(background, 0, 0, getWidth(), getHeight(), null);
        }
    }

    public void actionPerformed(ActionEvent e) {
        MyButton button = (MyButton) e.getSource();

        if (button.equals(backButton)) {   // User wants to go back to the main menu page
            setVisible(false);
            DriverClass.displayMainMenu();
        }
    }
}
```