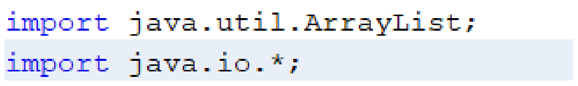
Criterion C: Development

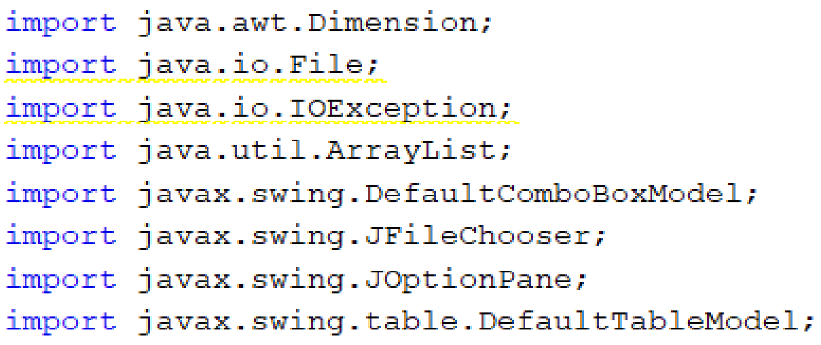
Full Source Code with Comments are found in Appendix 2.

Techniques Used:

Libraries Imported:



```java
import java.util.ArrayList;
import java.io.*;
```

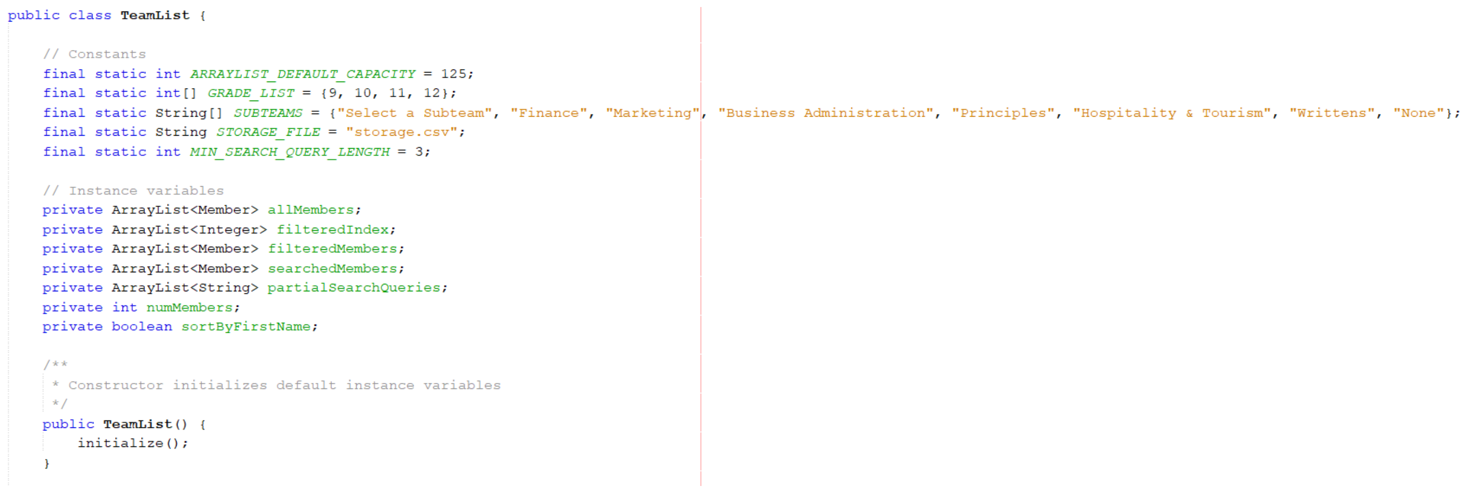


```java
import java.awt.Dimension;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
```

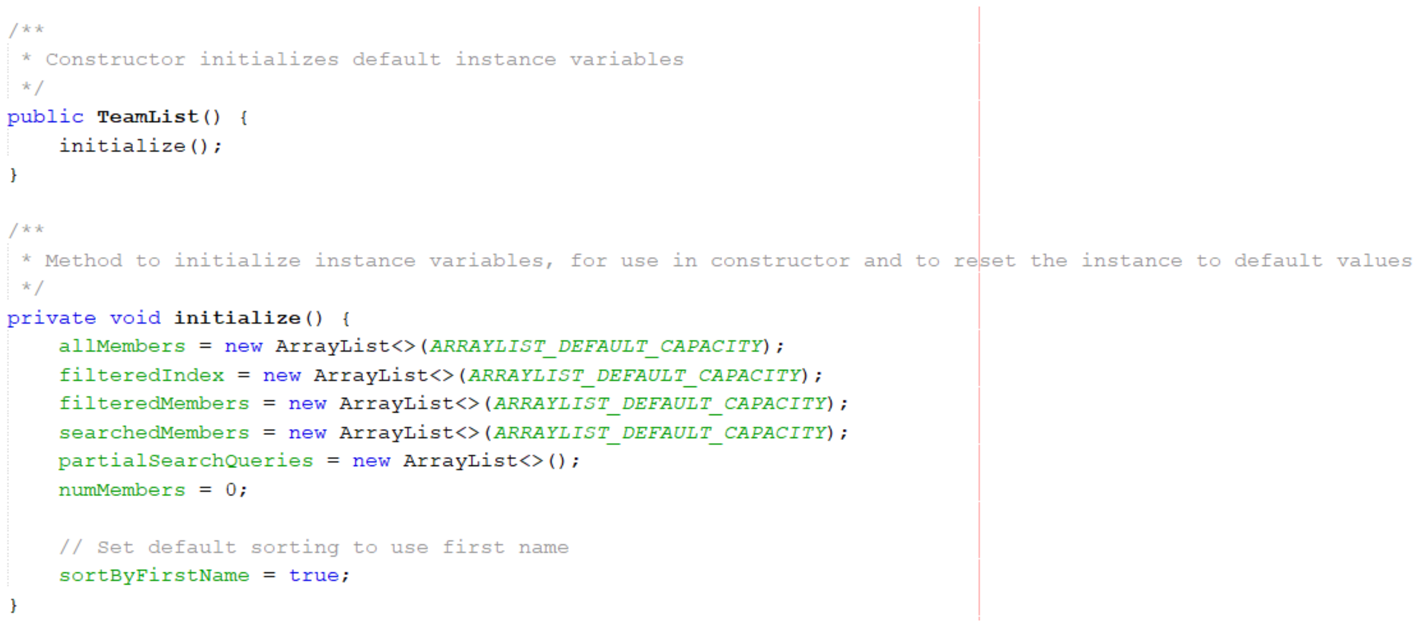
OOP:

Polymorphism, Encapsulation, Inheritance

Constructor of TeamList



```java
public class TeamList {

    // Constants
    final static int ARRAYLIST_DEFAULT_CAPACITY = 125;
    final static int[] GRADE_LIST = {9, 10, 11, 12};
    final static String[] SUBTEAMS = {"Select a Subteam", "Finance", "Marketing", "Business Administration", "Principles", "Hospitality & Tourism", "Writtens", "None"};
    final static String STORAGE_FILE = "storage.csv";
    final static int MIN_SEARCH_QUERY_LENGTH = 3;

    // Instance variables
    private ArrayList<Member> allMembers;
    private ArrayList<Integer> filteredIndex;
    private ArrayList<Member> filteredMembers;
    private ArrayList<Member> searchedMembers;
    private ArrayList<String> partialSearchQueries;
    private int numMembers;
    private boolean sortByFirstName;

    /**
     * Constructor initializes default instance variables
     */
    public TeamList() {
        initialize();
    }
```

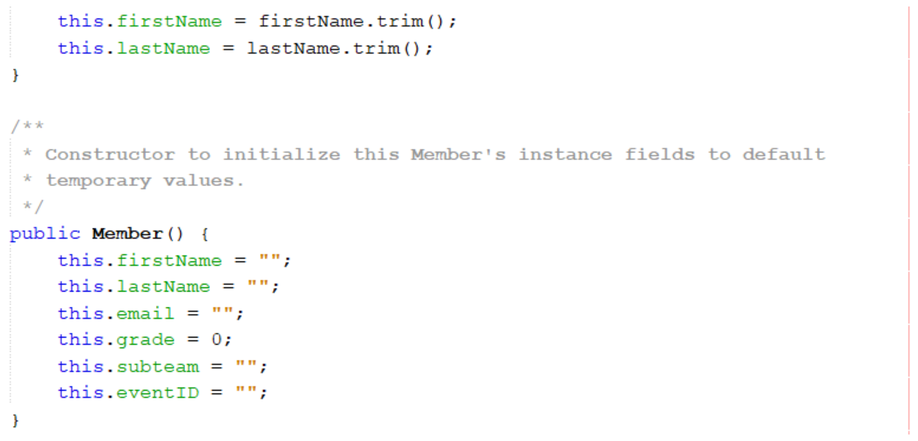


```
/**
 * Constructor initializes default instance variables
 */
public TeamList() {
    initialize();
}

/**
 * Method to initialize instance variables, for use in constructor and to reset the instance to default values
 */
private void initialize() {
    allMembers = new ArrayList<>(ARRAYLIST_DEFAULT_CAPACITY);
    filteredIndex = new ArrayList<>(ARRAYLIST_DEFAULT_CAPACITY);
    filteredMembers = new ArrayList<>(ARRAYLIST_DEFAULT_CAPACITY);
    searchedMembers = new ArrayList<>(ARRAYLIST_DEFAULT_CAPACITY);
    partialSearchQueries = new ArrayList<>();
    numMembers = 0;

    // Set default sorting to use first name
    sortByFirstName = true;
}
```

A helper method is used to initialize instance variables because I re-initialize instance fields in deleteAllMembers. Default ArrayList capacities are set to avoid dynamic resizing, which is inefficient. If size > 125 Members then the ArrayList increases capacity automatically.

Member Constructor

```java
/**
 * The Member class containing instance fields and methods for each Member that
 * will be added in the DECA team organizer.
 *
 * @author Hugh Jiang
 */
public class Member {

    private String email;
    private String eventID; // DECA event ID, i.e. BFS or FTDM
    private String firstName;
    private int grade;
    private String lastName;
    private String subteam; // DECA subteam name

    /**
     * Constructor to initialize this Member's instance fields
     *
     * @param firstName the String containing the first name of this Member
     * @param lastName the String containing the last name of this Member
     * @param email the String containing this Member's email
     * @param grade the integer containing this Member's grade level
     * @param subteam the String containing this Member's subteam
     * @param eventID the String with this Member's DECA event ID (i.e. BFS,
     * BTDM, etc)
     */
    public Member(String firstName, String lastName, String email, int grade, String subteam, String eventID) {
        this.firstName = firstName.trim();
        this.lastName = lastName.trim();
        this.email = email.trim();
        this.grade = grade;
        this.subteam = subteam.trim();
        this.eventID = eventID.trim();
    }

    /**
     * Constructor to initialize this Member's instance fields without
     * parameters for subteam and eventID, which will be set to default
     * temporary values
     *
     * @param firstName the String containing the first name of this Member
     * @param lastName the String containing the last name of this Member
     * @param email the String containing this Member's email
     * @param grade the integer containing this Member's grade level
     */
    public Member(String firstName, String lastName, String email, int grade) {
        this.firstName = firstName.trim();
        this.lastName = lastName.trim();
        this.email = email.trim();
        this.grade = grade;
        this.subteam = "";
        this.eventID = "";
    }

    /**
     * Constructor to initialize this Member's instance fields without
     * parameters for subteam, eventID, grade, email, which will be set to
     * default temporary values
     *
     * @param firstName the String containing the first name of this Member
     * @param lastName the String containing the last name of this Member
     */
    public Member(String firstName, String lastName) {
        this();
```



```
    this.firstName = firstName.trim();
    this.lastName = lastName.trim();
}

/**
 * Constructor to initialize this Member's instance fields to default
 * temporary values.
 */
public Member() {
    this.firstName = "";
    this.lastName = "";
    this.email = "";
    this.grade = 0;
    this.subteam = "";
    this.eventID = "";
}
```

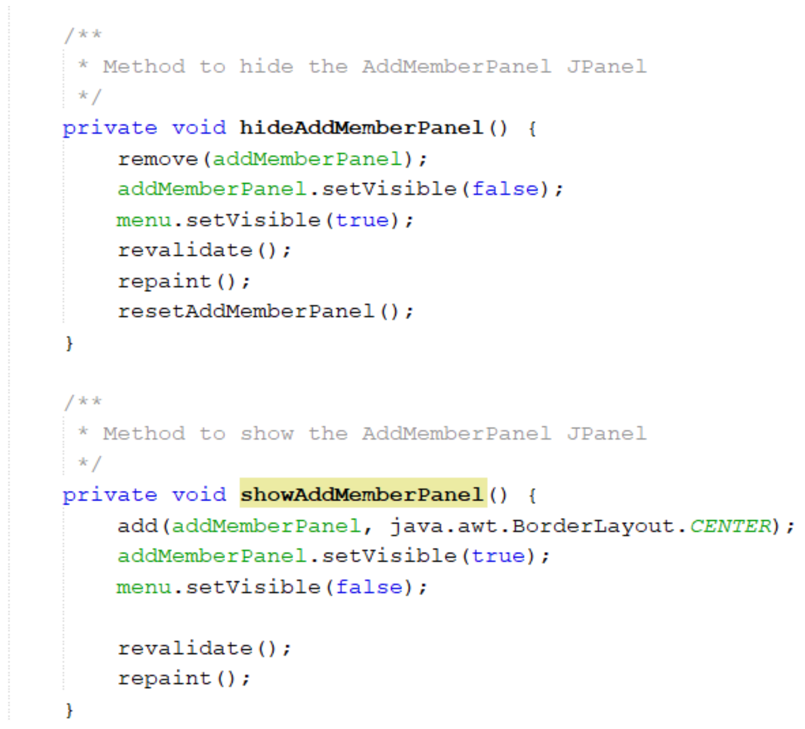
Data Structure Choices

Member Objects are stored in TeamList class ArrayLists. At first, I considered a SQL database, but for less than 100 members it would've been unnecessary. I also thought about ADTs like binary search trees. However, since Member objects have multiple instance variables, there isn't one single way to compare them, and a binary tree would be inefficient for searching partial matches. I also needed to directly access Members on the list to delete/edit them, which made ArrayLists the best choice.

GUI:

GUI was formatted using NetBeans's GUI builder. ActionListener code, dynamic GUI elements, etc were programmed myself. Gui class inherits from JFrame so I can use its pre-existing methods.

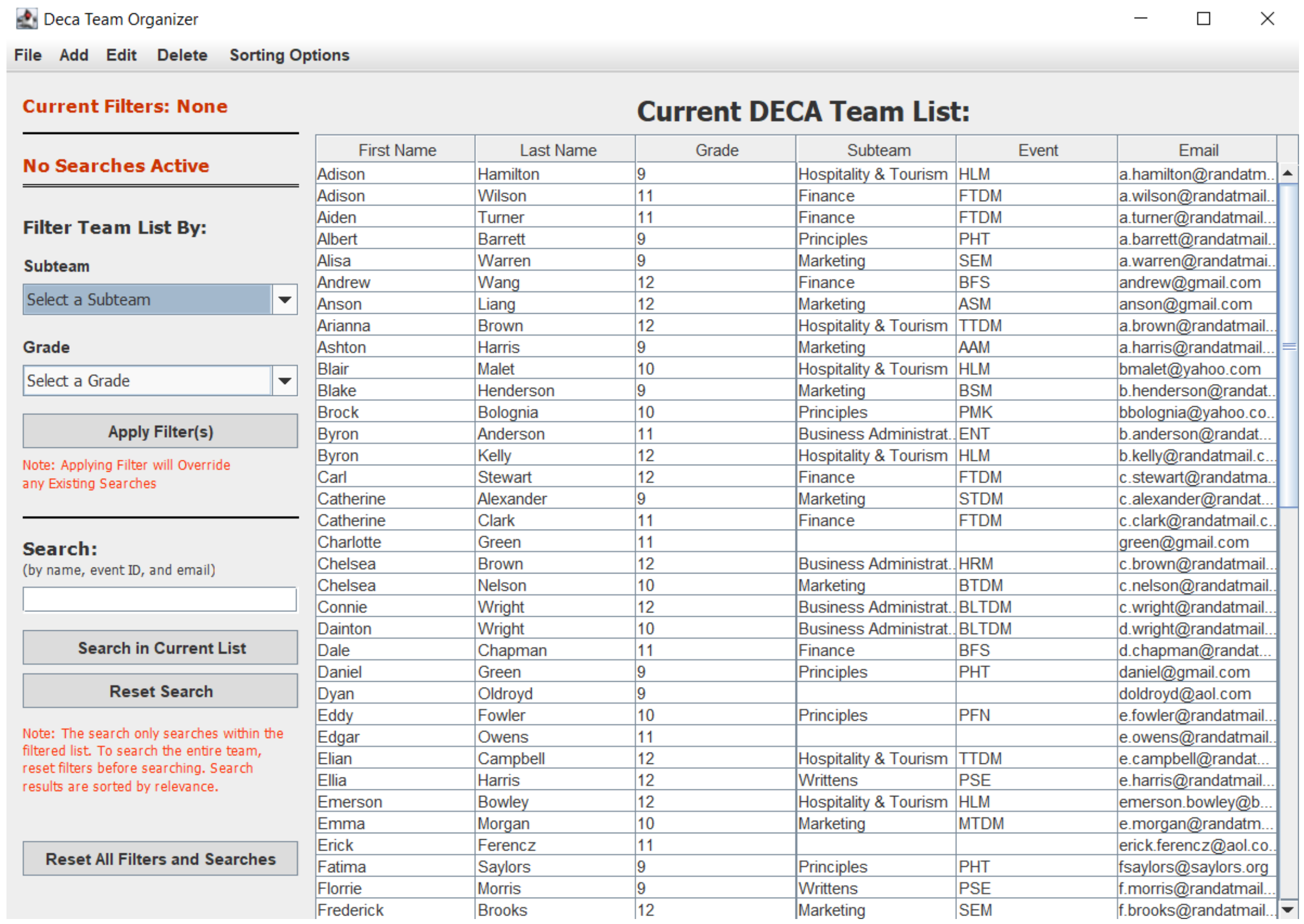
Displaying Different Panels



```java
/**
 * Method to hide the AddMemberPanel JPanel
 */
private void hideAddMemberPanel() {
    remove(addMemberPanel);
    addMemberPanel.setVisible(false);
    menu.setVisible(true);
    revalidate();
    repaint();
    resetAddMemberPanel();
}

/**
 * Method to show the AddMemberPanel JPanel
 */
private void showAddMemberPanel() {
    add(addMemberPanel, java.awt.BorderLayout.CENTER);
    addMemberPanel.setVisible(true);
    menu.setVisible(false);

    revalidate();
    repaint();
}
```

When an ActionListener detects the click of a menu-item or button requiring a panel change, then they call hide and show methods (example methods above)

Displaying Members in the GUI Table



Note: Applying Filter will Override any Existing Searches

Search:
(by name, event ID, and email)

Note: The search only searches within the filtered list. To search the entire team, reset filters before searching. Search results are sorted by relevance.

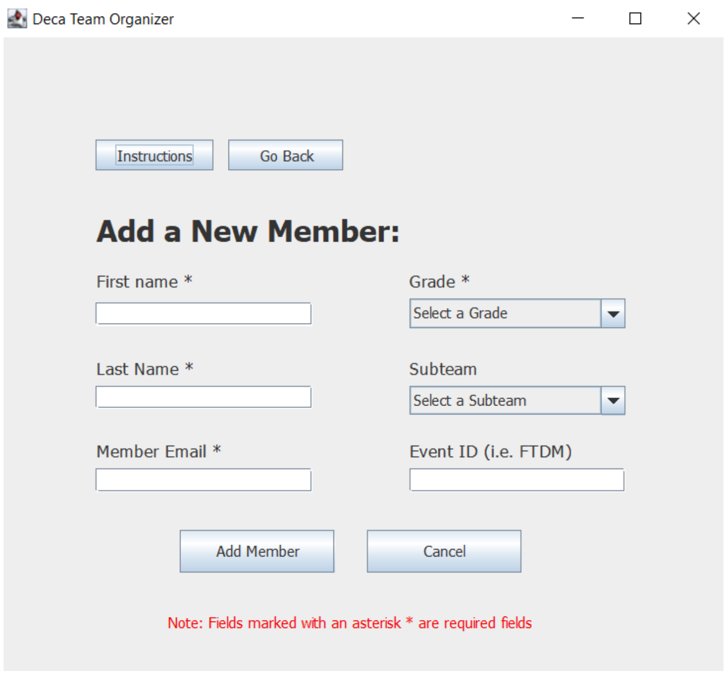
Current DECA Team List:

First Name	Last Name	Grade	Subteam	Event	Email
Adison	Hamilton	9	Hospitality & Tourism	HLM	a.hamilton@randatm...
Adison	Wilson	11	Finance	FTDM	a.wilson@randatmail...
Aiden	Turner	11	Finance	FTDM	a.turner@randatmail...
Albert	Barrett	9	Principles	PHT	a.barrett@randatmail...
Alisa	Warren	9	Marketing	SEM	a.warren@randatmai...
Andrew	Wang	12	Finance	BFS	andrew@gmail.com
Anson	Liang	12	Marketing	ASM	anson@gmail.com
Arianna	Brown	12	Hospitality & Tourism	TTDM	a.brown@randatmail...
Ashton	Harris	9	Marketing	AAM	a.harris@randatmail...
Blair	Malet	10	Hospitality & Tourism	HLM	bmalet@yahoo.com
Blake	Henderson	9	Marketing	BSM	b.henderson@randat...
Brock	Bolognia	10	Principles	PMK	bbolognia@yahoo.co...
Byron	Anderson	11	Business Administrat...	ENT	b.anderson@randat...
Byron	Kelly	12	Hospitality & Tourism	HLM	b.kelly@randatmail.c...
Carl	Stewart	12	Finance	FTDM	c.stewart@randatma...
Catherine	Alexander	9	Marketing	STDM	c.alexander@randat...
Catherine	Clark	11	Finance	FTDM	c.clark@randatmail.c...
Charlotte	Green	11			green@gmail.com
Chelsea	Brown	12	Business Administrat...	HRM	c.brown@randatmail...
Chelsea	Nelson	10	Marketing	BTDM	c.nelson@randatmail...
Connie	Wright	12	Business Administrat...	BLTDM	c.wright@randatmail...
Dainton	Wright	10	Business Administrat...	BLTDM	d.wright@randatmail...
Dale	Chapman	11	Finance	BFS	d.chapman@randat...
Daniel	Green	9	Principles	PHT	daniel@gmail.com
Dyan	Oldroyd	9			doldroyd@aol.com
Eddy	Fowler	10	Principles	PFN	e.fowler@randatmail...
Edgar	Owens	11			e.owens@randatmail...
Elian	Campbell	12	Hospitality & Tourism	TTDM	e.campbell@randat...
Ellia	Harris	12	Writtens	PSE	e.harris@randatmail...
Emerson	Bowley	12	Hospitality & Tourism	HLM	emerson.bowley@b...
Emma	Morgan	10	Marketing	MTDM	e.morgan@randatm...
Erick	Ferencz	11			erick.ferencz@aol.co...
Fatima	Saylors	9	Principles	PHT	fsaylors@saylors.org
Florrie	Morris	9	Writtens	PSE	f.morris@randatmail...
Frederick	Brooks	12	Marketing	SEM	f.brooks@randatmail...

```java
/**
 * Populate main display table with the full team list
 */
public void populateTable() {
    populateTable(memberList.getMembers());

    // Set boolean flag to indicate main table is showed
    mainTableShowed = true;
}
```

```
/**
 * Populate main display table with specified ArrayList of Members: this method overloads populateTable().
 * @param memberArr the ArrayList of Members that is to be added to the table
 */
public void populateTable(ArrayList<Member> memberArr) {
    // Set boolean flag
    mainTableShowed = false;

    DefaultTableModel model = (DefaultTableModel) displayTable.getModel();

    // Remove all existing elements on table
    model.getDataVector().removeAllElements();

    String[] rowData = new String[NUM_TABLE_COLUMNS];
    Member temp;

    for (int i = 0; i < memberArr.size(); i++) {
        temp = memberArr.get(i);
        rowData[0] = temp.getFirstName();
        rowData[1] = temp.getLastname();
        rowData[2] = String.valueOf(temp.getGrade());
        rowData[3] = "" + temp.getSubteam();
        rowData[4] = "" + temp.getEventID();
        rowData[5] = temp.getEmail();

        model.addRow(rowData);
    }
    // Reset any previous selections
    displayTable.clearSelection();

    // Reset gui to show the updated table
    revalidate();
    repaint();
}
```

Adding Members



Add a New Member:

First name *

Last Name *

Member Email *

Grade *

Select a Grade

Subteam

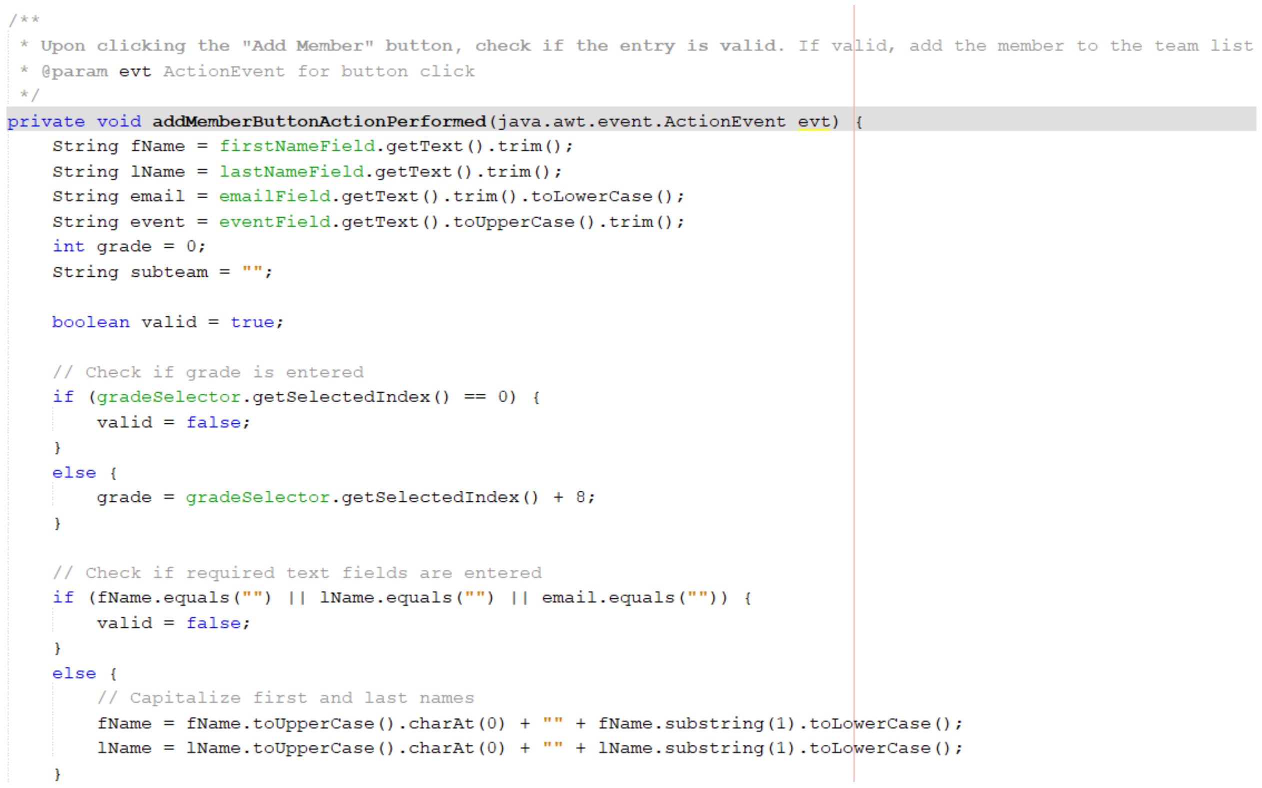
Select a Subteam

Event ID (i.e. FTDM)

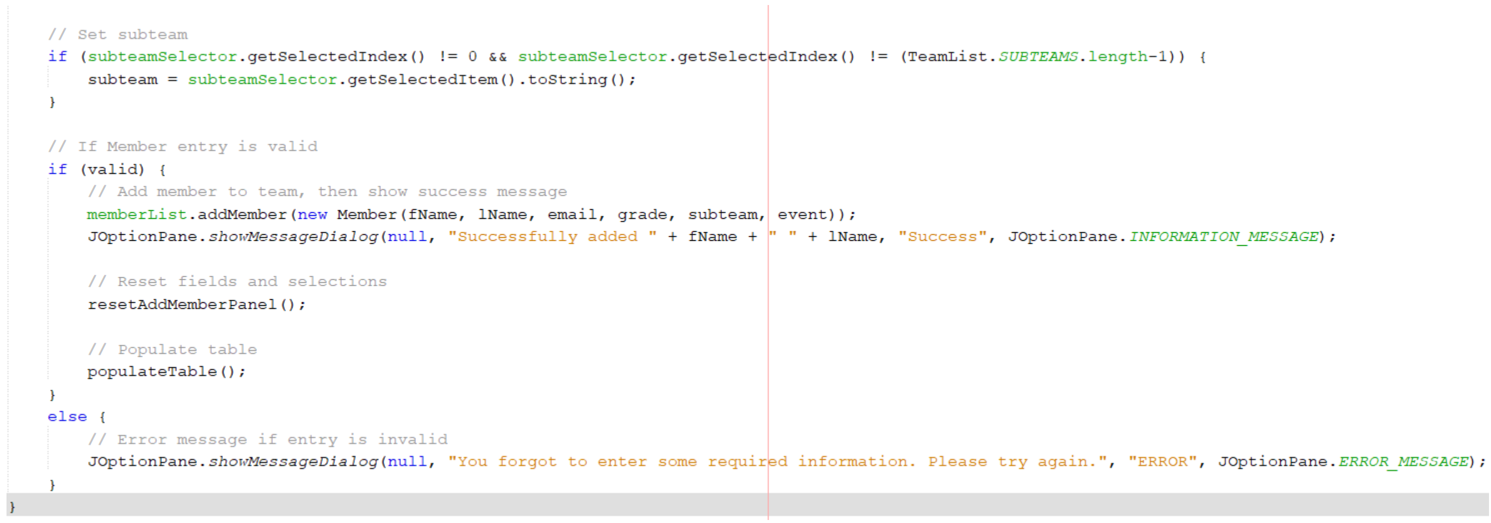
Note: Fields marked with an asterisk * are required fields

ActionListener: “AddMember” Button

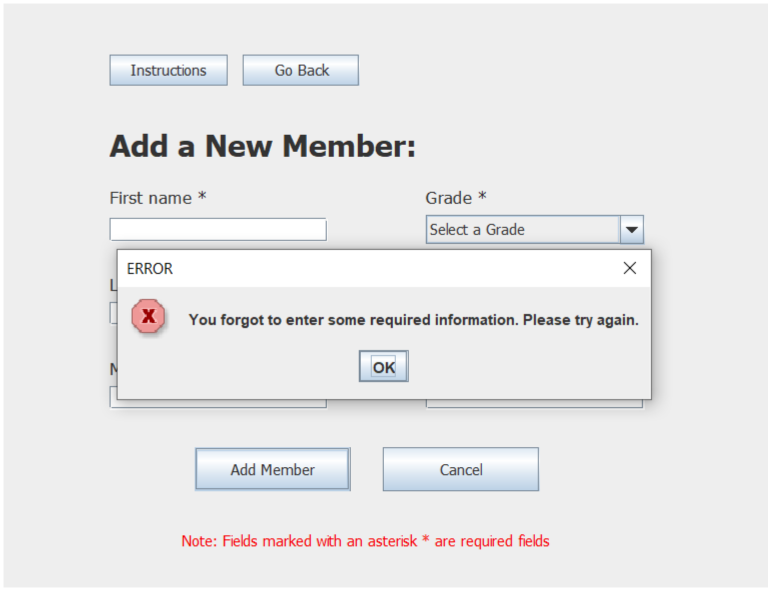
Upon clicking “Add Member”, verify + cleanse input (or show error message) and add a new Member to TeamList object. Then reset panel to allow adding more members.



```java
/**
 * Upon clicking the "Add Member" button, check if the entry is valid. If valid, add the member to the team list
 * @param evt ActionEvent for button click
 */
private void addMemberButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String fName = firstNameField.getText().trim();
    String lName = lastNameField.getText().trim();
    String email = emailField.getText().trim().toLowerCase();
    String event = eventField.getText().toUpperCase().trim();
    int grade = 0;
    String subteam = "";

    boolean valid = true;

    // Check if grade is entered
    if (gradeSelector.getSelectedIndex() == 0) {
        valid = false;
    }
    else {
        grade = gradeSelector.getSelectedIndex() + 8;
    }

    // Check if required text fields are entered
    if (fName.equals("") || lName.equals("") || email.equals("")) {
        valid = false;
    }
    else {
        // Capitalize first and last names
        fName = fName.toUpperCase().charAt(0) + "" + fName.substring(1).toLowerCase();
        lName = lName.toUpperCase().charAt(0) + "" + lName.substring(1).toLowerCase();
    }
```



```java
    // Set subteam
    if (subteamSelector.getSelectedIndex() != 0 && subteamSelector.getSelectedIndex() != (TeamList.SUBTEAMS.length-1)) {
        subteam = subteamSelector.getSelectedItem().toString();
    }

    // If Member entry is valid
    if (valid) {
        // Add member to team, then show success message
        memberList.addMember(new Member(fName, lName, email, grade, subteam, event));
        JOptionPane.showMessageDialog(null, "Successfully added " + fName + " " + lName, "Success", JOptionPane.INFORMATION_MESSAGE);

        // Reset fields and selections
        resetAddMemberPanel();

        // Populate table
        populateTable();
    }
    else {
        // Error message if entry is invalid
        JOptionPane.showMessageDialog(null, "You forgot to enter some required information. Please try again.", "ERROR", JOptionPane.ERROR_MESSAGE);
    }
}
```



Add a New Member:

You forgot to enter some required information. Please try again.

Note: Fields marked with an asterisk * are required fields

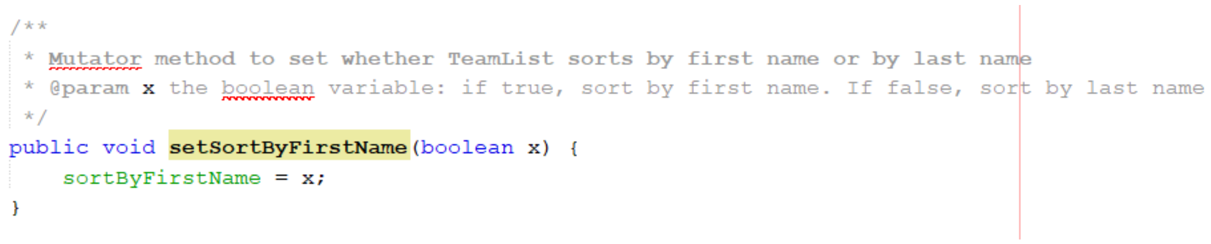
<u>TeamList.addMember:</u>

```
/**
 * Mutator method to add a member
 * @param m the Member to be added to the team
 */
public void addMember(Member m) {

    // Add member to the allMembers ArrayList
    allMembers.add(m);

    // Increment Counter
    numMembers++;

    // Sort (by insertion)
    sort();

    /*
    Note: add member to other subteams only when requested in their respective mutator methods:
    this way, we do not have to continually re-sort multiple arrays
    */

    // Set list of filtered members to default (no filter)
    filteredMembers = allMembers;
}
```

Sorting Algorithm (Modified Recursive Insertion Sort)

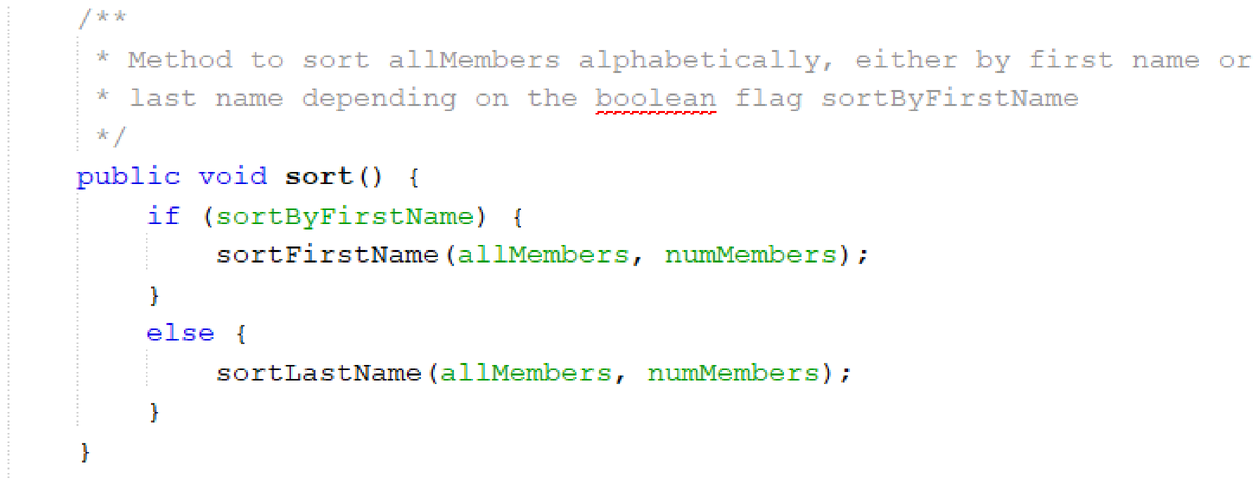
Sorting uses insertion sort because my program adds individual members into TeamList's allMembers ArrayList. Insertion sort is most efficient for sorting new elements added into an already sorted list.

Since Member objects have multiple methods of comparison, user can choose in GUI to sort alphabetically by either first/last name. The selected option is indicated by TeamList’s boolean sortByFirstName (by default, sortByFirstName=true).



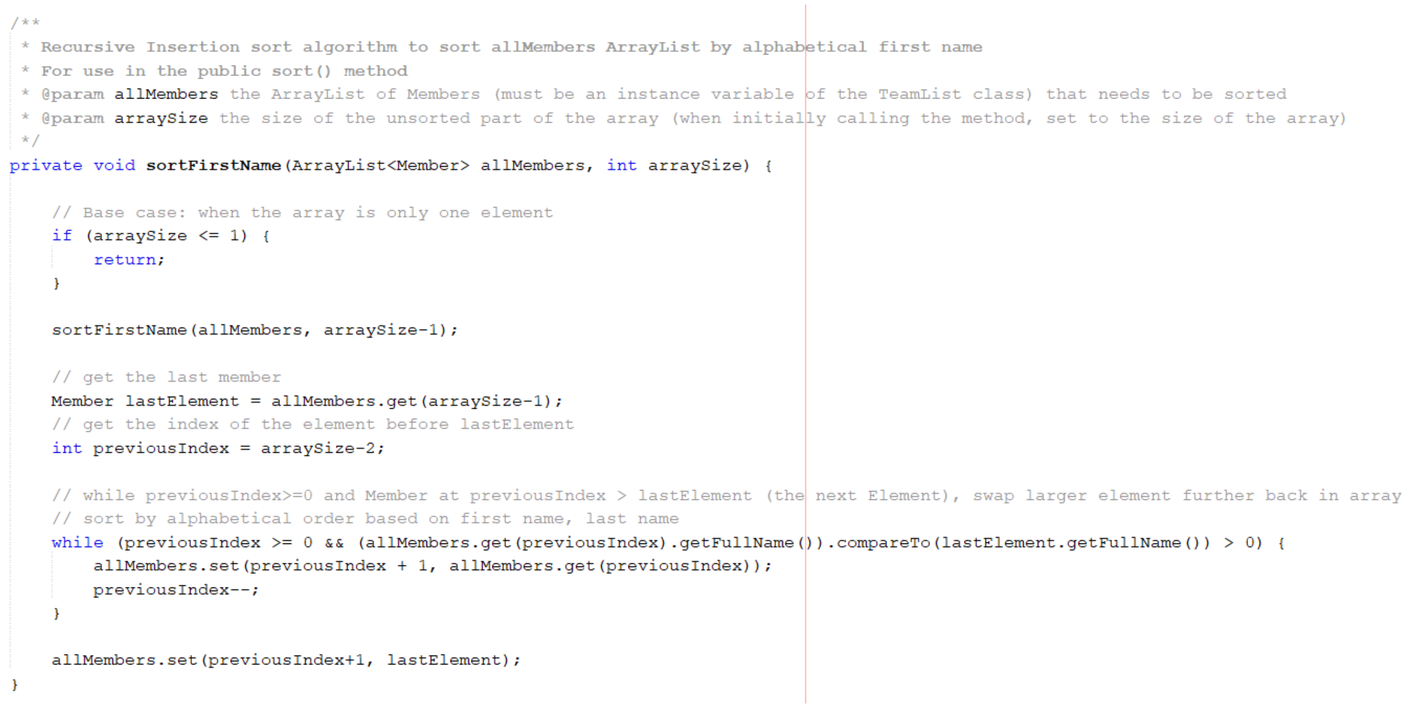
```java
/**
 * Mutator method to set whether TeamList sorts by first name or by last name
 * @param x the boolean variable: if true, sort by first name. If false, sort by last name
 */
public void setSortByFirstName(boolean x) {
    sortByFirstName = x;
}
```

<u>TeamList.sort():</u>

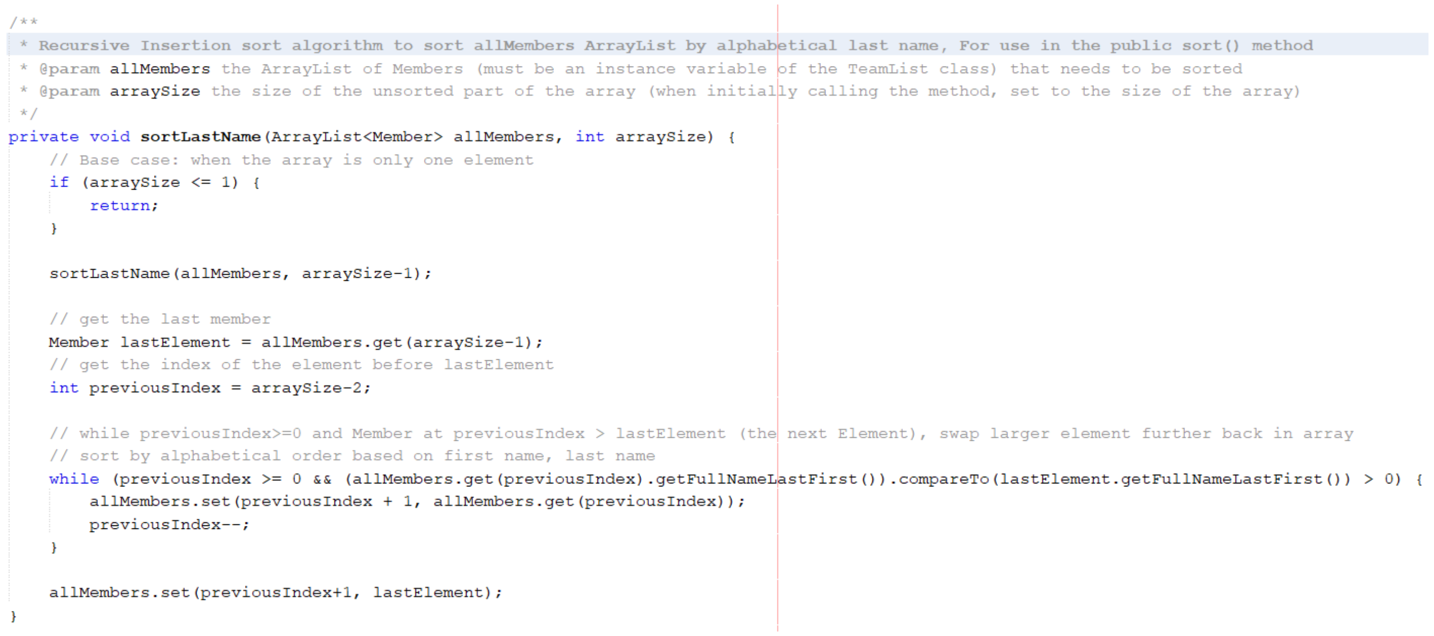


```java
/**
 * Method to sort allMembers alphabetically, either by first name or
 * last name depending on the boolean flag sortByFirstName
 */
public void sort() {
    if (sortByFirstName) {
        sortFirstName(allMembers, numMembers);
    }
    else {
        sortLastName(allMembers, numMembers);
    }
}
```

Helper Methods perform recursive insertion sort depending on which sorting option was selected by the user:



```java
/**
 * Recursive Insertion sort algorithm to sort allMembers ArrayList by alphabetical first name
 * For use in the public sort() method
 * @param allMembers the ArrayList of Members (must be an instance variable of the TeamList class) that needs to be sorted
 * @param arraySize the size of the unsorted part of the array (when initially calling the method, set to the size of the array)
 */
private void sortFirstName(ArrayList<Member> allMembers, int arraySize) {

    // Base case: when the array is only one element
    if (arraySize <= 1) {
        return;
    }

    sortFirstName(allMembers, arraySize-1);

    // get the last member
    Member lastElement = allMembers.get(arraySize-1);
    // get the index of the element before lastElement
    int previousIndex = arraySize-2;

    // while previousIndex>=0 and Member at previousIndex > lastElement (the next Element), swap larger element further back in array
    // sort by alphabetical order based on first name, last name
    while (previousIndex >= 0 && (allMembers.get(previousIndex).getFullName()).compareTo(lastElement.getFullName()) > 0) {
        allMembers.set(previousIndex + 1, allMembers.get(previousIndex));
        previousIndex--;
    }

    allMembers.set(previousIndex+1, lastElement);
}
```

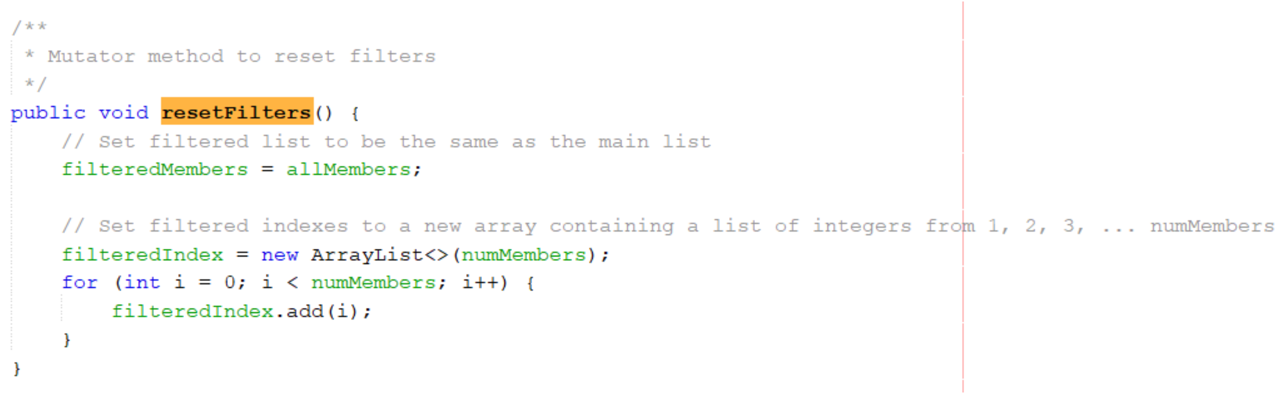


```java
/**
 * Recursive Insertion sort algorithm to sort allMembers ArrayList by alphabetical last name, For use in the public sort() method
 * @param allMembers the ArrayList of Members (must be an instance variable of the TeamList class) that needs to be sorted
 * @param arraySize the size of the unsorted part of the array (when initially calling the method, set to the size of the array)
 */
private void sortLastName(ArrayList<Member> allMembers, int arraySize) {
    // Base case: when the array is only one element
    if (arraySize <= 1) {
        return;
    }

    sortLastName(allMembers, arraySize-1);

    // get the last member
    Member lastElement = allMembers.get(arraySize-1);
    // get the index of the element before lastElement
    int previousIndex = arraySize-2;

    // while previousIndex>=0 and Member at previousIndex > lastElement (the next Element), swap larger element further back in array
    // sort by alphabetical order based on first name, last name
    while (previousIndex >= 0 && (allMembers.get(previousIndex).getFullNameLastFirst()).compareTo(lastElement.getFullNameLastFirst()) > 0) {
        allMembers.set(previousIndex + 1, allMembers.get(previousIndex));
        previousIndex--;
    }

    allMembers.set(previousIndex+1, lastElement);
}
```

Note: Since filtered member lists are initially populated linearly from a copy of allMembers, filtered lists will be sorted correctly too.

Layered Filtering of Members

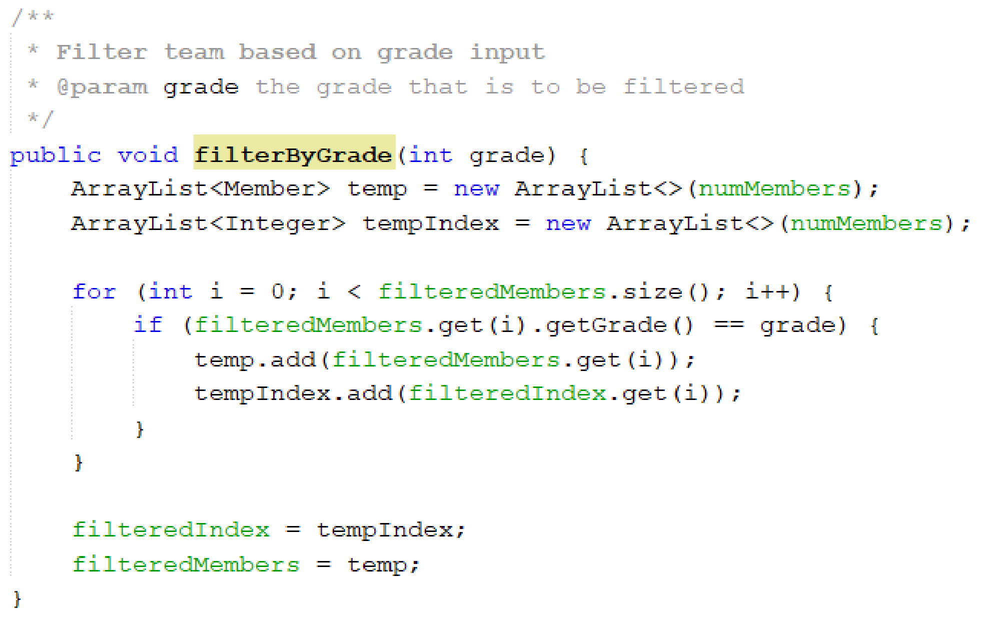
When the user selects filter(s) in GUI, I first reset pre-existing filters by calling TeamList.resetFilters().



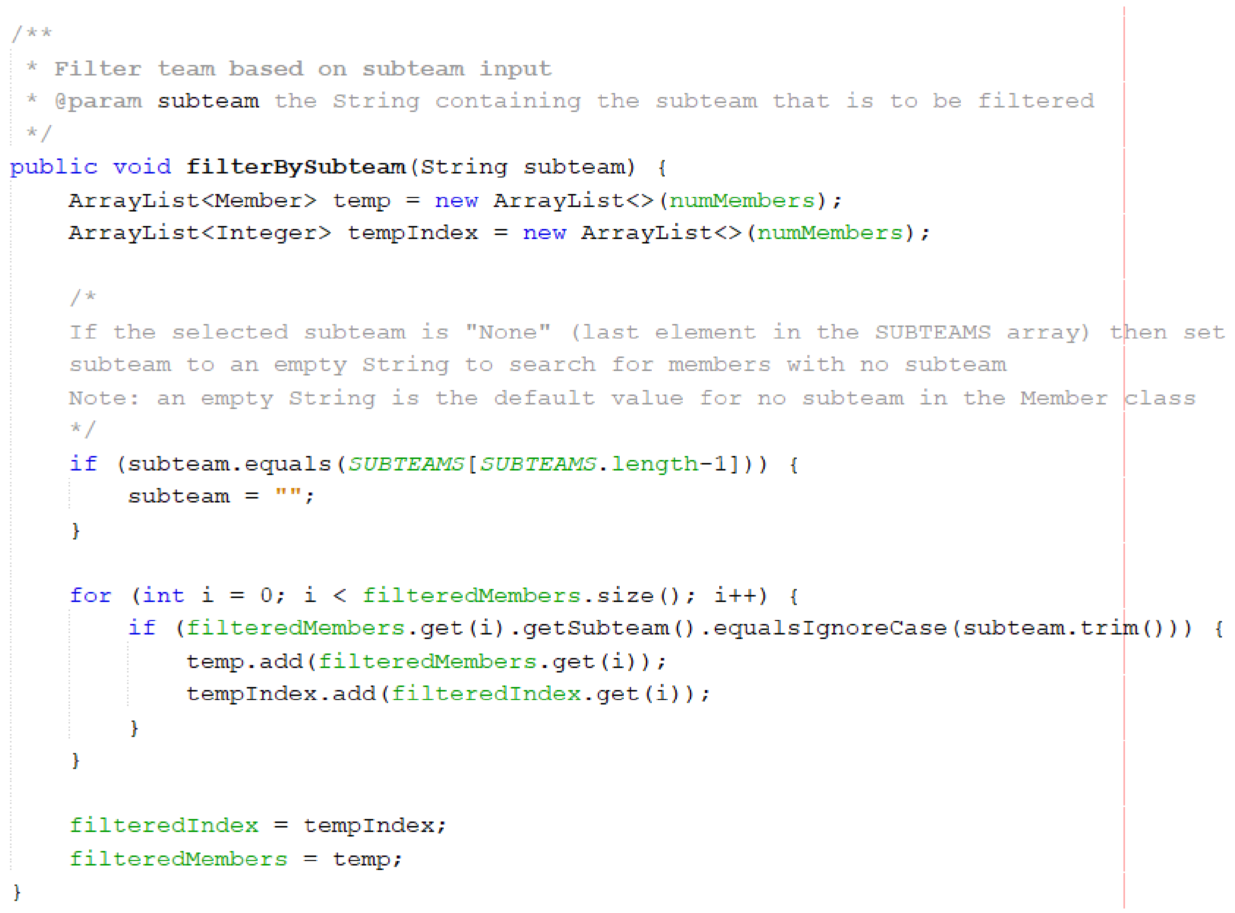
```java
/**
 * Mutator method to reset filters
 */
public void resetFilters() {
    // Set filtered list to be the same as the main list
    filteredMembers = allMembers;

    // Set filtered indexes to a new array containing a list of integers from 1, 2, 3, ... numMembers
    filteredIndex = new ArrayList<>(numMembers);
    for (int i = 0; i < numMembers; i++) {
        filteredIndex.add(i);
    }
}
```

Note: ArrayList<Integer> filteredIndex was originally meant to allow direct access to Members in filtered lists using the index of the Member in the allMembers list (to allow faster deletion/editing from filtered lists), but this was not fully implemented yet throughout the program.

The following method(s) are called depending on user's filter selections. These methods linear search filteredMembers for the correct Members, saves them to a temporary ArrayList, and then sets filteredMembers = temp.



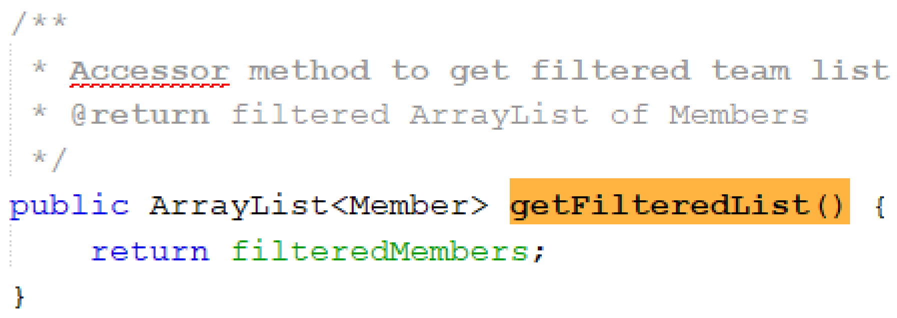
```java
/**
 * Filter team based on grade input
 * @param grade the grade that is to be filtered
 */
public void filterByGrade(int grade) {
    ArrayList<Member> temp = new ArrayList<>(numMembers);
    ArrayList<Integer> tempIndex = new ArrayList<>(numMembers);

    for (int i = 0; i < filteredMembers.size(); i++) {
        if (filteredMembers.get(i).getGrade() == grade) {
            temp.add(filteredMembers.get(i));
            tempIndex.add(filteredIndex.get(i));
        }
    }

    filteredIndex = tempIndex;
    filteredMembers = temp;
}
```



```
/**
 * Filter team based on subteam input
 * @param subteam the String containing the subteam that is to be filtered
 */
public void filterBySubteam(String subteam) {
    ArrayList<Member> temp = new ArrayList<>(numMembers);
    ArrayList<Integer> tempIndex = new ArrayList<>(numMembers);

    /*
    If the selected subteam is "None" (last element in the SUBTEAMS array) then set
    subteam to an empty String to search for members with no subteam
    Note: an empty String is the default value for no subteam in the Member class
    */
    if (subteam.equals(SUBTEAMS[SUBTEAMS.length-1])) {
        subteam = "";
    }

    for (int i = 0; i < filteredMembers.size(); i++) {
        if (filteredMembers.get(i).getSubteam().equalsIgnoreCase(subteam.trim())) {
            temp.add(filteredMembers.get(i));
            tempIndex.add(filteredIndex.get(i));
        }
    }

    filteredIndex = tempIndex;
    filteredMembers = temp;
}
```

Using separate methods to filter by different conditions improves the program’s extensibility, because new filters can be added at any time by creating new methods like filterByCondition(param). No other code in TeamList would need to be changed.

Getter method returns filteredList for GUI display:



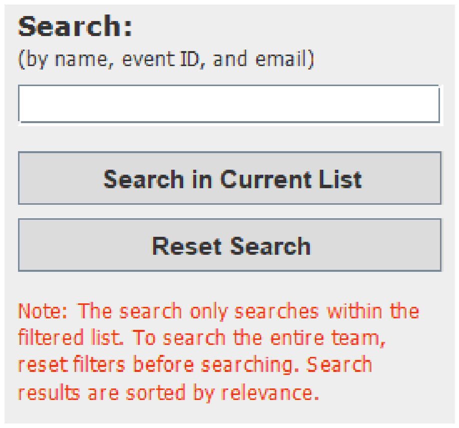
```
/**
 * Accessor method to get filtered team list
 * @return filtered ArrayList of Members
 */
public ArrayList<Member> getFilteredList() {
    return filteredMembers;
}
```

Gui Implementation:

```java
// Filter and display members when the button filterButton is clicked
private void filterButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // Upon clicking of the filter button
    String subteam ="";
    int grade = 0;
    String statusText = "<html>Filtered By:";
    boolean filtered = false;

    // Reset previous (if any) filters stored in the memberList object
    memberList.resetFilters();

    // Reset search fields
    searchStatus.setText(DEFAULT_SEARCH_STATUS_TEXT);
    searchField.setText("");

    // Get selected subteam from GUi
    if (filterSubteamDropdown.getSelectedIndex() != 0) {
        // Filter processing
        subteam = filterSubteamDropdown.getSelectedItem().toString();
        memberList.filterBySubteam(subteam);
        // Update status text
        statusText = statusText + "<br>Subteam: " + subteam;
        // Reset filter dropdown to default
        filterSubteamDropdown.setSelectedIndex(0);

        // Boolean flag
        filtered = true;
    }

    // Get selected grade from GUI
    if (gradeDropdown.getSelectedIndex() != 0) {
        // Since grades are set as {none, 9, 10, 11, 12}, the grade is the index + 8
        grade = gradeDropdown.getSelectedIndex() + 8;
        memberList.filterByGrade(grade);
        statusText = statusText + "<br>Grade: " + grade;

        gradeDropdown.setSelectedIndex(0);

        // boolean flag
        filtered = true;
    }

    // Check if filter was successful
    if (filtered) {
        // Close html tag in label and then set the status text
        statusText = statusText + "</html>";
        filterStatus.setText(statusText);

        // Set boolean flag to indicate the filtered list is being shown
        mainTableShowed = false;

        // Populate the table with the filtered list
        populateTable(memberList.getFilteredList());
    }
    // Else, if no filters were selected, handle error
    else {
        JOptionPane.showMessageDialog(null, "No filters were selected", "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

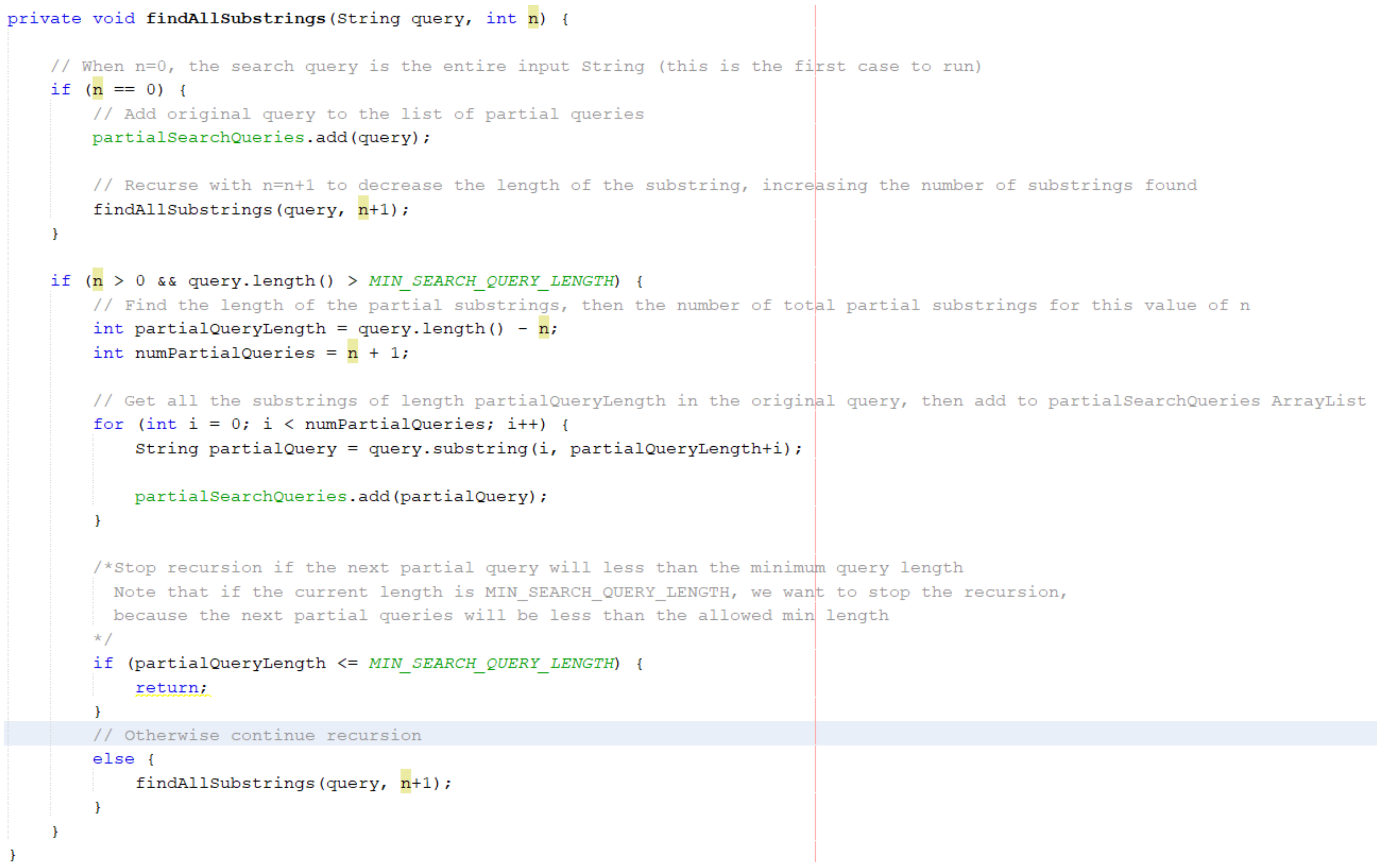
Member Searching Algorithm

My searching algorithm searches by name, email & event ID simultaneously. The results include partial query matches and are sorted by relevance. This is done by finding all the substrings in the query, and then searching for members that contain the partial queries.



Note: The search only searches within the filtered list. To search the entire team, reset filters before searching. Search results are sorted by relevance.

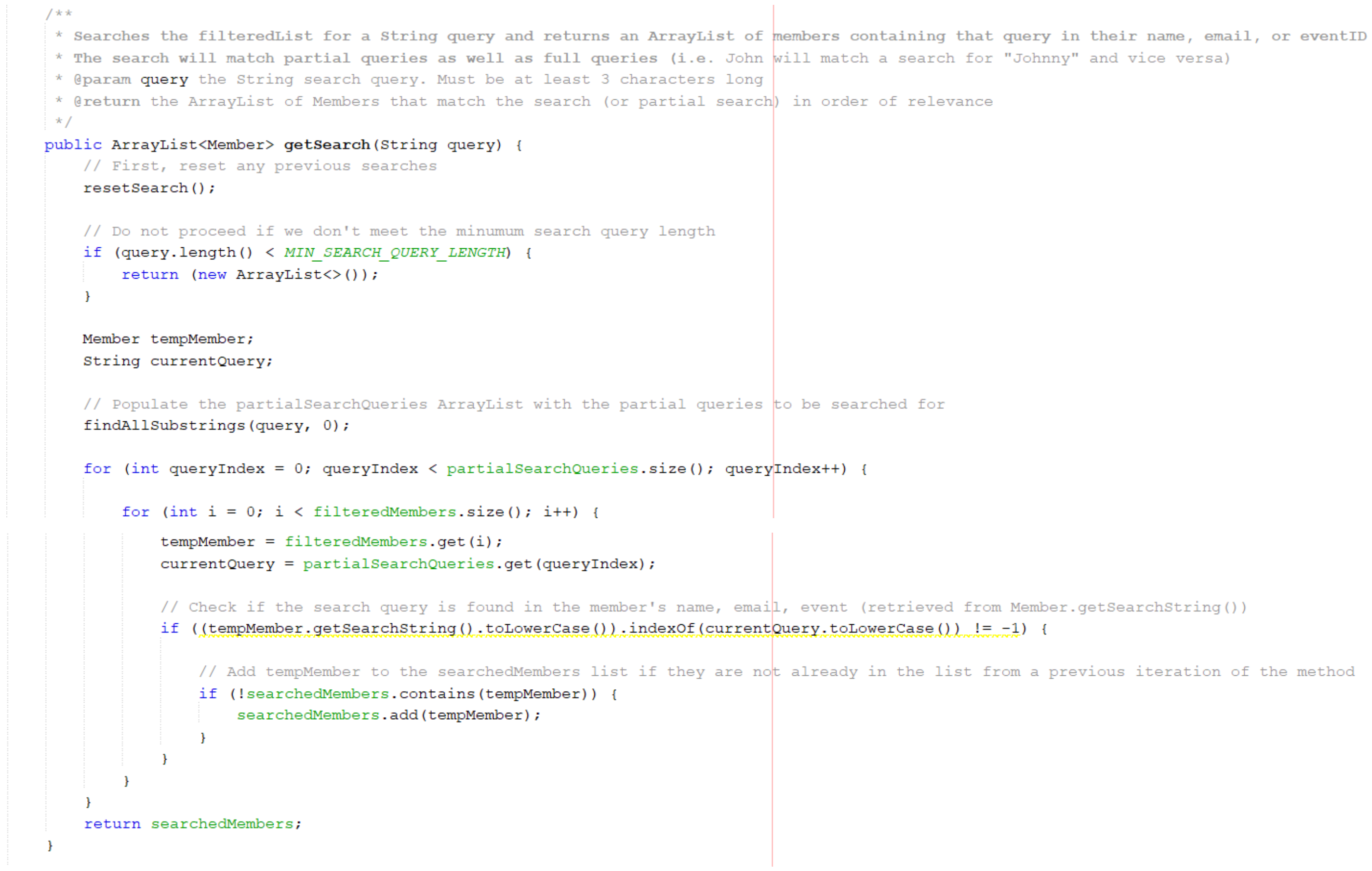
First, I use a recursive method to find all the possible substrings within the search query (greater than 3 characters long). The substrings are stored in a global ArrayList in descending length.



```java
private void findAllSubstrings(String query, int n) {

    // When n=0, the search query is the entire input String (this is the first case to run)
    if (n == 0) {
        // Add original query to the list of partial queries
        partialSearchQueries.add(query);

        // Recurse with n=n+1 to decrease the length of the substring, increasing the number of substrings found
        findAllSubstrings(query, n+1);
    }

    if (n > 0 && query.length() > MIN_SEARCH_QUERY_LENGTH) {
        // Find the length of the partial substrings, then the number of total partial substrings for this value of n
        int partialQueryLength = query.length() - n;
        int numPartialQueries = n + 1;

        // Get all the substrings of length partialQueryLength in the original query, then add to partialSearchQueries ArrayList
        for (int i = 0; i < numPartialQueries; i++) {
            String partialQuery = query.substring(i, partialQueryLength+i);

            partialSearchQueries.add(partialQuery);
        }

        /*Stop recursion if the next partial query will less than the minimum query length
          Note that if the current length is MIN_SEARCH_QUERY_LENGTH, we want to stop the recursion,
          because the next partial queries will be less than the allowed min length
        */
        if (partialQueryLength <= MIN_SEARCH_QUERY_LENGTH) {
            return;
        }
        // Otherwise continue recursion
        else {
            findAllSubstrings(query, n+1);
        }
    }
}
```

When findAllSubstrings is called in the getSearch() method, n is set to 0 to set substring length = query length. Each time n is incremented recursively, the length of substrings found will decrease by 1, and the number of substrings added in that recursive round will be n + 1. A for loop adds all the substrings of the specified length into the partialSearchQueries ArrayList.

The return method getSearch() below calls the above method and uses nested loops to perform a linear search of filteredMembers to search for the user's query. The ArrayList of resulting Members is returned to be displayed in Gui.



```java
/**
 * Searches the filteredList for a String query and returns an ArrayList of members containing that query in their name, email, or eventID
 * The search will match partial queries as well as full queries (i.e. John will match a search for "Johnny" and vice versa)
 * @param query the String search query. Must be at least 3 characters long
 * @return the ArrayList of Members that match the search (or partial search) in order of relevance
 */
public ArrayList<Member> getSearch(String query) {
    // First, reset any previous searches
    resetSearch();

    // Do not proceed if we don't meet the minumum search query length
    if (query.length() < MIN_SEARCH_QUERY_LENGTH) {
        return (new ArrayList<>());
    }

    Member tempMember;
    String currentQuery;

    // Populate the partialSearchQueries ArrayList with the partial queries to be searched for
    findAllSubstrings(query, 0);

    for (int queryIndex = 0; queryIndex < partialSearchQueries.size(); queryIndex++) {

        for (int i = 0; i < filteredMembers.size(); i++) {
            tempMember = filteredMembers.get(i);
            currentQuery = partialSearchQueries.get(queryIndex);

            // Check if the search query is found in the member's name, email, event (retrieved from Member.getSearchString())
            if ((tempMember.getSearchString().toLowerCase()).indexOf(currentQuery.toLowerCase()) != -1) {

                // Add tempMember to the searchedMembers list if they are not already in the list from a previous iteration of the method
                if (!searchedMembers.contains(tempMember)) {
                    searchedMembers.add(tempMember);
                }
            }
        }
    }
    return searchedMembers;
}
```

Since partialSearchQueries has substrings added in descending order of length, search results will be sorted by relevance since queries are compared starting from partialSearchQueries.get(0);

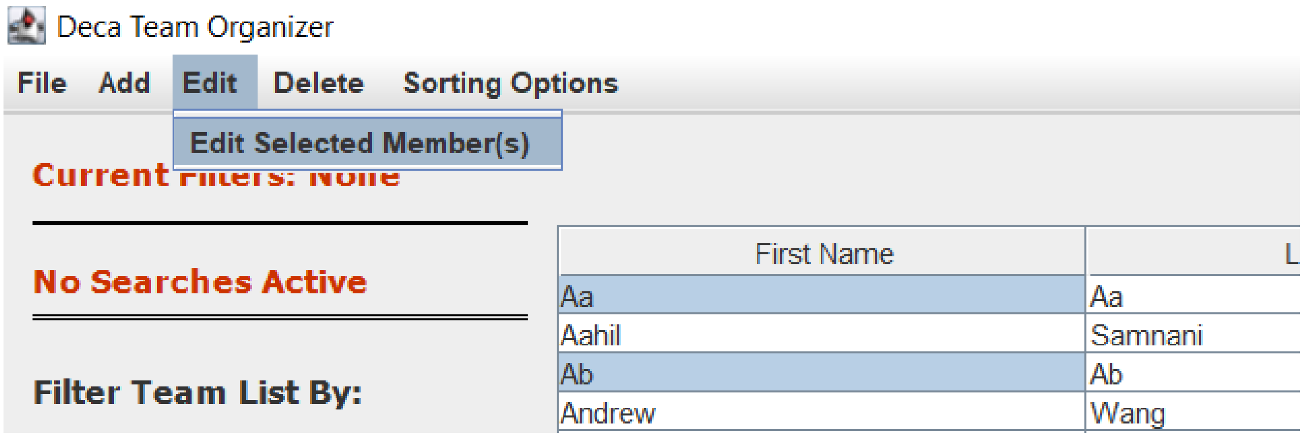
Other helper methods:

```java
/**
 * Accessor method to return the member's full name, email, and event ID in
 * a String. Used for searching purposes in TeamList
 *
 * @return the String containing the Member's full name, email, and event ID
 */
public String getSearchString() {
    return firstName + " " + lastName + " " + email + " " + eventID;
}
```

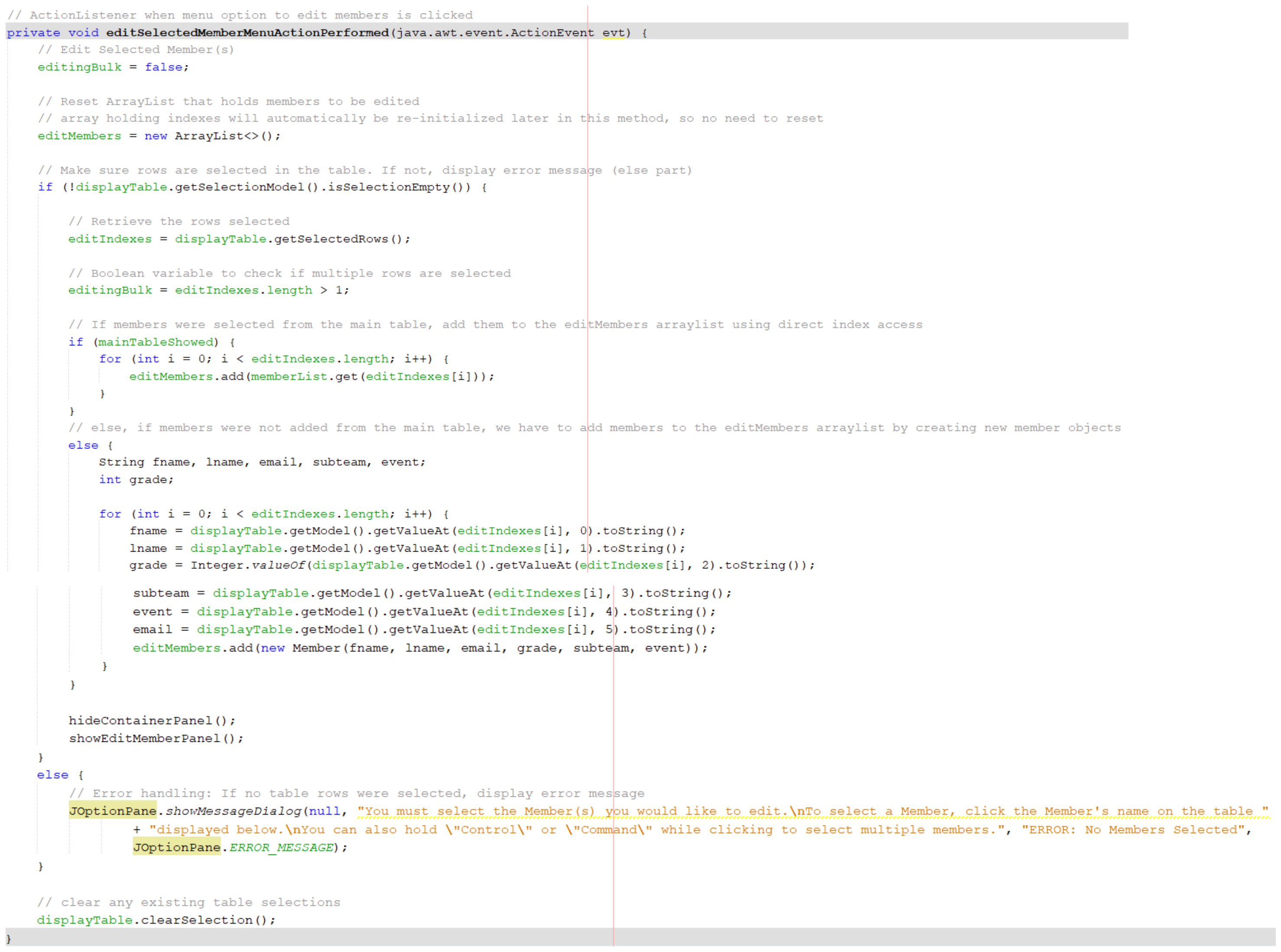
```java
/**
 * Mutator method to reset searches. This is a private method because searches are
 * automatically reset after each search so layering search upon search is not possible,
 * thus it will only be privately called within TeamList and not in Gui
 */
private void resetSearch() {
    searchedMembers = new ArrayList<>();
    partialSearchQueries = new ArrayList<>();
}
```

Editing Members:

First, I add the selected members to an ArrayList containing the members needing to be edited.

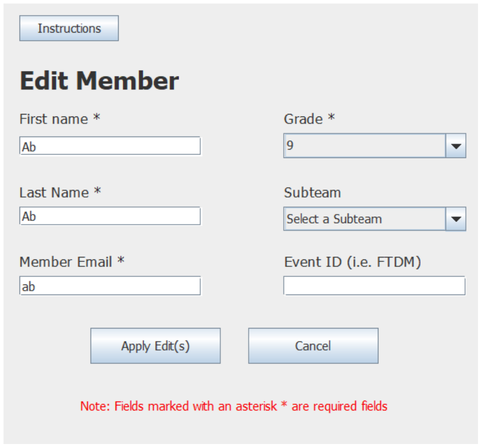


First Name	L
Aa	Aa
Aahil	Samnani
Ab	Ab
Andrew	Wang



```java
// ActionListener when menu option to edit members is clicked
private void editSelectedMemberMenuActionPerformed(java.awt.event.ActionEvent evt) {
    // Edit Selected Member(s)
    editingBulk = false;

    // Reset ArrayList that holds members to be edited
    // array holding indexes will automatically be re-initialized later in this method, so no need to reset
    editMembers = new ArrayList<>();

    // Make sure rows are selected in the table. If not, display error message (else part)
    if (!displayTable.getSelectionModel().isSelectionEmpty()) {

        // Retrieve the rows selected
        editIndexes = displayTable.getSelectedRows();

        // Boolean variable to check if multiple rows are selected
        editingBulk = editIndexes.length > 1;

        // If members were selected from the main table, add them to the editMembers arraylist using direct index access
        if (mainTableShowed) {
            for (int i = 0; i < editIndexes.length; i++) {
                editMembers.add(memberList.get(editIndexes[i]));
            }
        }
        // else, if members were not added from the main table, we have to add members to the editMembers arraylist by creating new member objects
        else {
            String fname, lname, email, subteam, event;
            int grade;

            for (int i = 0; i < editIndexes.length; i++) {
                fname = displayTable.getModel().getValueAt(editIndexes[i], 0).toString();
                lname = displayTable.getModel().getValueAt(editIndexes[i], 1).toString();
                grade = Integer.valueOf(displayTable.getModel().getValueAt(editIndexes[i], 2).toString());

                subteam = displayTable.getModel().getValueAt(editIndexes[i], 3).toString();
                event = displayTable.getModel().getValueAt(editIndexes[i], 4).toString();
                email = displayTable.getModel().getValueAt(editIndexes[i], 5).toString();
                editMembers.add(new Member(fname, lname, email, grade, subteam, event));
            }
        }

        hideContainerPanel();
        showEditMemberPanel();
    }
    else {
        // Error handling: If no table rows were selected, display error message
        JOptionPane.showMessageDialog(null, "You must select the Member(s) you would like to edit.\nTo select a Member, click the Member's name on the table "
                + "displayed below.\nYou can also hold \"Control\" or \"Command\" while clicking to select multiple members.", "ERROR: No Members Selected",
                JOptionPane.ERROR_MESSAGE);
    }

    // clear any existing table selections
    displayTable.clearSelection();
}
```

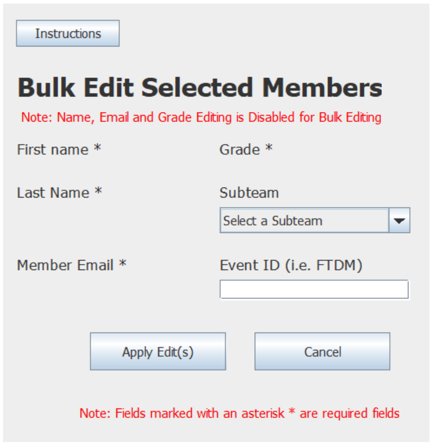
Then, I show the edit members panel depending on whether or not editing by bulk, and apply edits using the following method:



Edit Member

First name * Ab	Grade * 9
Last Name * Ab	Subteam Select a Subteam
Member Email * ab	Event ID (i.e. FTDM)

Note: Fields marked with an asterisk * are required fields

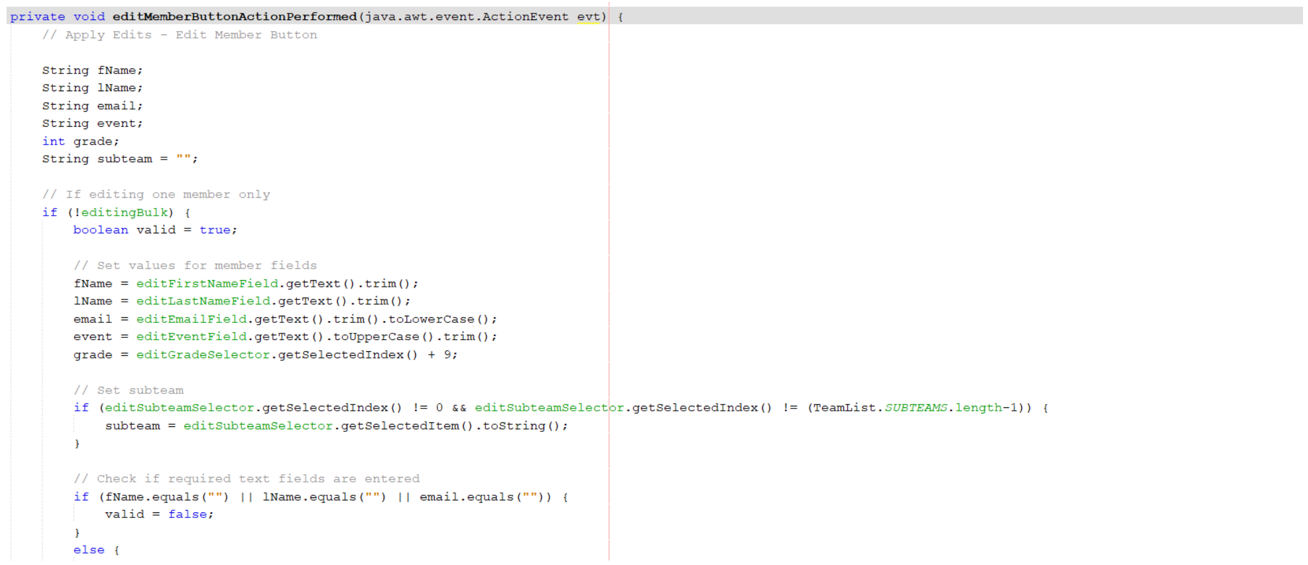


Bulk Edit Selected Members

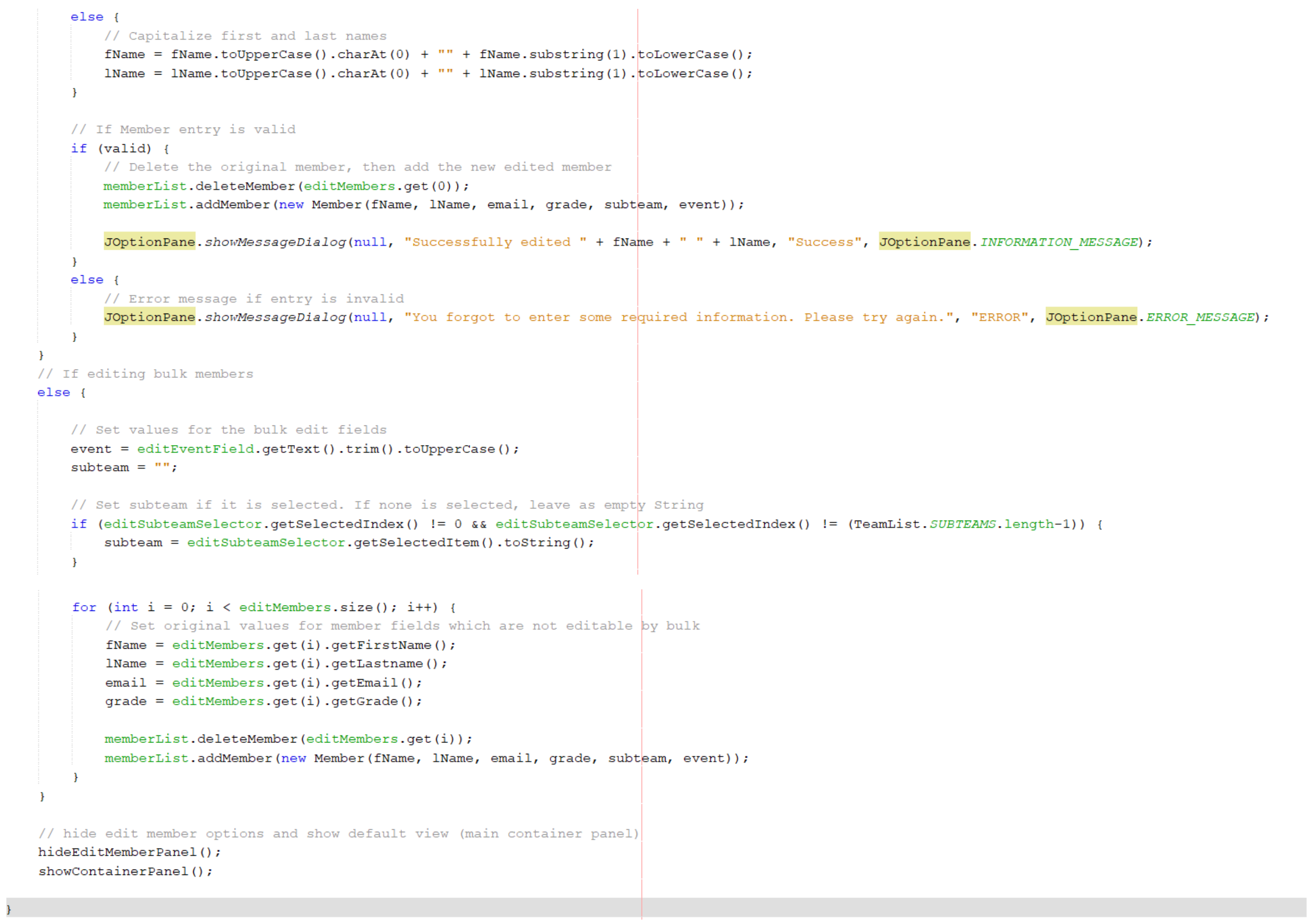
Note: Name, Email and Grade Editing is Disabled for Bulk Editing

First name *	Grade *
Last Name *	Subteam Select a Subteam
Member Email *	Event ID (i.e. FTDM)

Note: Fields marked with an asterisk * are required fields



```java
private void editMemberButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // Apply Edits - Edit Member Button

    String fName;
    String lName;
    String email;
    String event;
    int grade;
    String subteam = "";

    // If editing one member only
    if (!editingBulk) {
        boolean valid = true;

        // Set values for member fields
        fName = editFirstNameField.getText().trim();
        lName = editLastNameField.getText().trim();
        email = editEmailField.getText().trim().toLowerCase();
        event = editEventField.getText().toUpperCase().trim();
        grade = editGradeSelector.getSelectedIndex() + 9;

        // Set subteam
        if (editSubteamSelector.getSelectedIndex() != 0 && editSubteamSelector.getSelectedIndex() != (TeamList.SUBTEAMS.length-1)) {
            subteam = editSubteamSelector.getSelectedItem().toString();
        }

        // Check if required text fields are entered
        if (fName.equals("") || lName.equals("") || email.equals("")) {
            valid = false;
        }
        else {
```



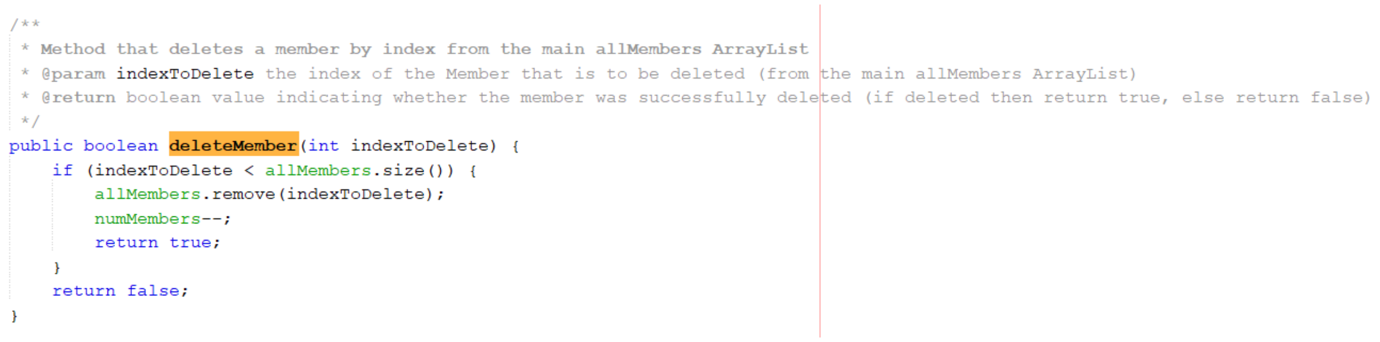
```java
        else {
            // Capitalize first and last names
            fName = fName.toUpperCase().charAt(0) + "" + fName.substring(1).toLowerCase();
            lName = lName.toUpperCase().charAt(0) + "" + lName.substring(1).toLowerCase();
        }

        // If Member entry is valid
        if (valid) {
            // Delete the original member, then add the new edited member
            memberList.deleteMember(editMembers.get(0));
            memberList.addMember(new Member(fName, lName, email, grade, subteam, event));

            JOptionPane.showMessageDialog(null, "Successfully edited " + fName + " " + lName, "Success", JOptionPane.INFORMATION_MESSAGE);
        }
        else {
            // Error message if entry is invalid
            JOptionPane.showMessageDialog(null, "You forgot to enter some required information. Please try again.", "ERROR", JOptionPane.ERROR_MESSAGE);
        }
    }
    // If editing bulk members
    else {

        // Set values for the bulk edit fields
        event = editEventField.getText().trim().toUpperCase();
        subteam = "";

        // Set subteam if it is selected. If none is selected, leave as empty String
        if (editSubteamSelector.getSelectedIndex() != 0 && editSubteamSelector.getSelectedIndex() != (TeamList.SUBTEAMS.length-1)) {
            subteam = editSubteamSelector.getSelectedItem().toString();
        }

        for (int i = 0; i < editMembers.size(); i++) {
            // Set original values for member fields which are not editable by bulk
            fName = editMembers.get(i).getFirstName();
            lName = editMembers.get(i).getLastname();
            email = editMembers.get(i).getEmail();
            grade = editMembers.get(i).getGrade();

            memberList.deleteMember(editMembers.get(i));
            memberList.addMember(new Member(fName, lName, email, grade, subteam, event));
        }
    }

    // hide edit member options and show default view (main container panel)
    hideEditMemberPanel();
    showContainerPanel();

}
```

Deletion Algorithm

Deleting Members from Default Display Table:

Members can be directly accessed by index in allMembers using the Gui table's row index.

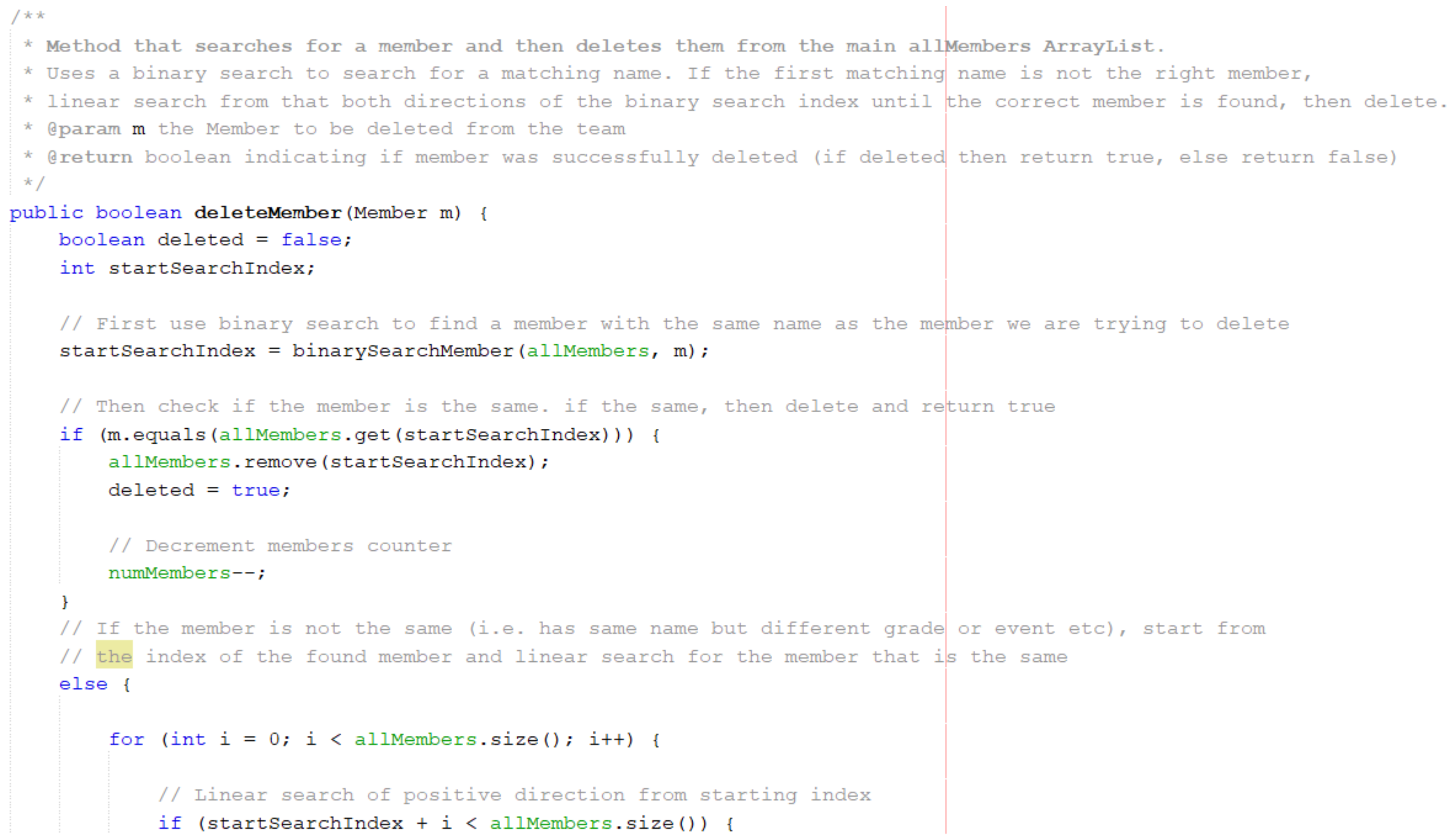
TeamList.deleteMember(int indexToDelete):



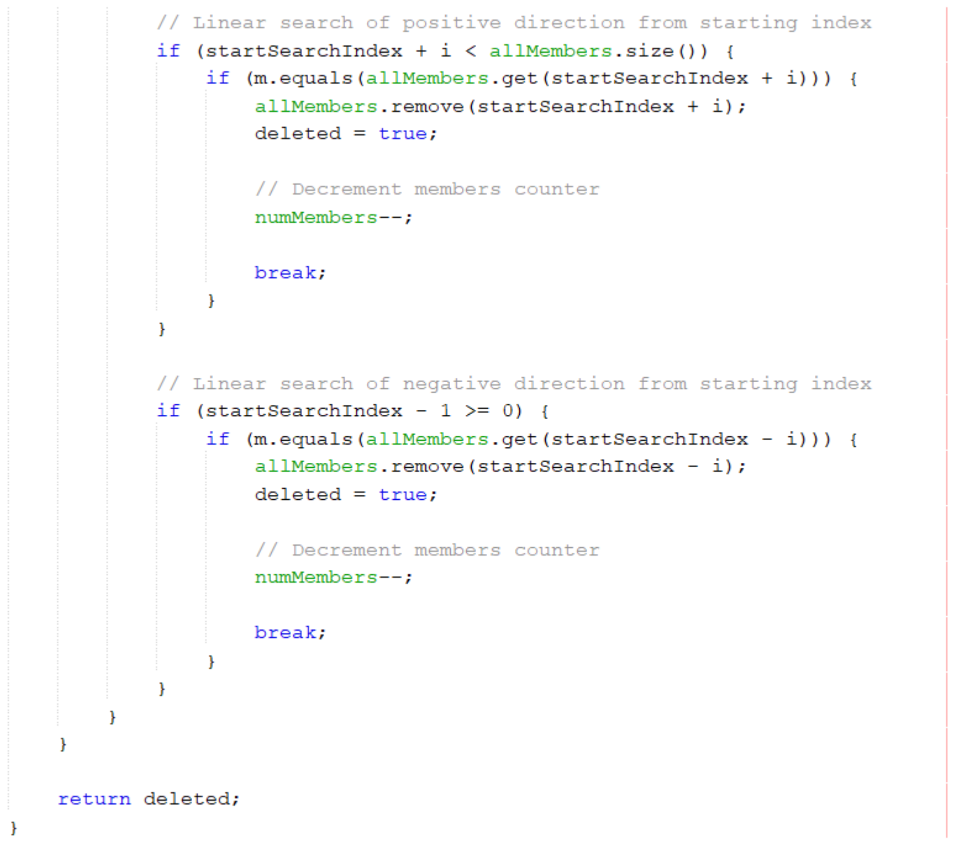
```java
/**
 * Method that deletes a member by index from the main allMembers ArrayList
 * @param indexToDelete the index of the Member that is to be deleted (from the main allMembers ArrayList)
 * @return boolean value indicating whether the member was successfully deleted (if deleted then return true, else return false)
 */
public boolean deleteMember(int indexToDelete) {
    if (indexToDelete < allMembers.size()) {
        allMembers.remove(indexToDelete);
        numMembers--;
        return true;
    }
    return false;
}
```

Deleting Members from Filtered/Searched List
If deleting from a filtered/searched list, then table row indexes will not correspond to the Member's index in allMembers. I use method overloading in TeamList to allow deletion without an index.

TeamList.deleteMember(Member m):



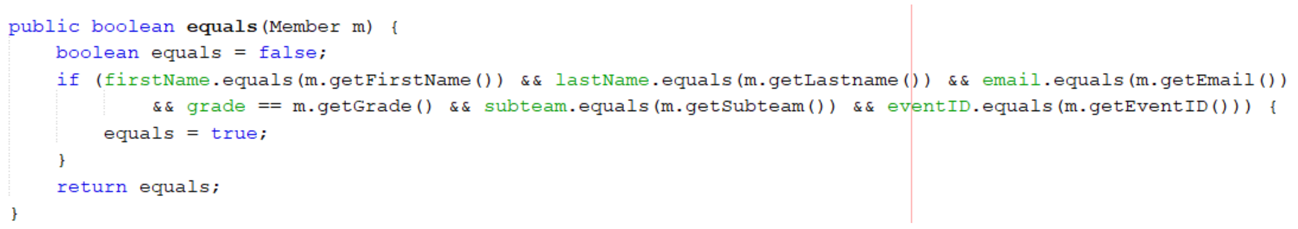
```java
/**
 * Method that searches for a member and then deletes them from the main allMembers ArrayList.
 * Uses a binary search to search for a matching name. If the first matching name is not the right member,
 * linear search from that both directions of the binary search index until the correct member is found, then delete.
 * @param m the Member to be deleted from the team
 * @return boolean indicating if member was successfully deleted (if deleted then return true, else return false)
 */
public boolean deleteMember(Member m) {
    boolean deleted = false;
    int startSearchIndex;

    // First use binary search to find a member with the same name as the member we are trying to delete
    startSearchIndex = binarySearchMember(allMembers, m);

    // Then check if the member is the same. if the same, then delete and return true
    if (m.equals(allMembers.get(startSearchIndex))) {
        allMembers.remove(startSearchIndex);
        deleted = true;

        // Decrement members counter
        numMembers--;
    }
    // If the member is not the same (i.e. has same name but different grade or event etc), start from
    // the index of the found member and linear search for the member that is the same
    else {

        for (int i = 0; i < allMembers.size(); i++) {

            // Linear search of positive direction from starting index
            if (startSearchIndex + i < allMembers.size()) {
```



```
            // Linear search of positive direction from starting index
            if (startSearchIndex + i < allMembers.size()) {
                if (m.equals(allMembers.get(startSearchIndex + i))) {
                    allMembers.remove(startSearchIndex + i);
                    deleted = true;

                    // Decrement members counter
                    numMembers--;

                    break;
                }
            }

            // Linear search of negative direction from starting index
            if (startSearchIndex - 1 >= 0) {
                if (m.equals(allMembers.get(startSearchIndex - i))) {
                    allMembers.remove(startSearchIndex - i);
                    deleted = true;

                    // Decrement members counter
                    numMembers--;

                    break;
                }
            }
        }
    }

    return deleted;
}
```

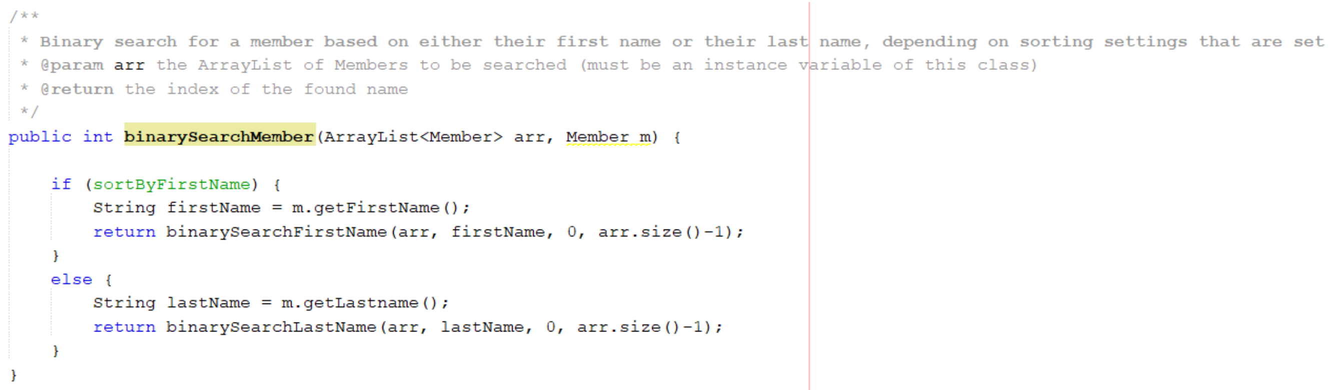
Since allMembers is always sorted, the method performs binary search for the requested Member's name, then compares the rest of the instance fields to ensure it's the correct Member (and not duplicate name). If it's not the right Member, then linear search is performed starting from the index of the retrieved Member, going in both directions of the ArrayList simultaneously. Since Members are sorted by name, duplicate names will be next to each other, and starting linear search after a binary search improves deletion efficiency to nearly $O(\log_2 N)$ versus O(N) for a strict linear search.

Member .equals method:



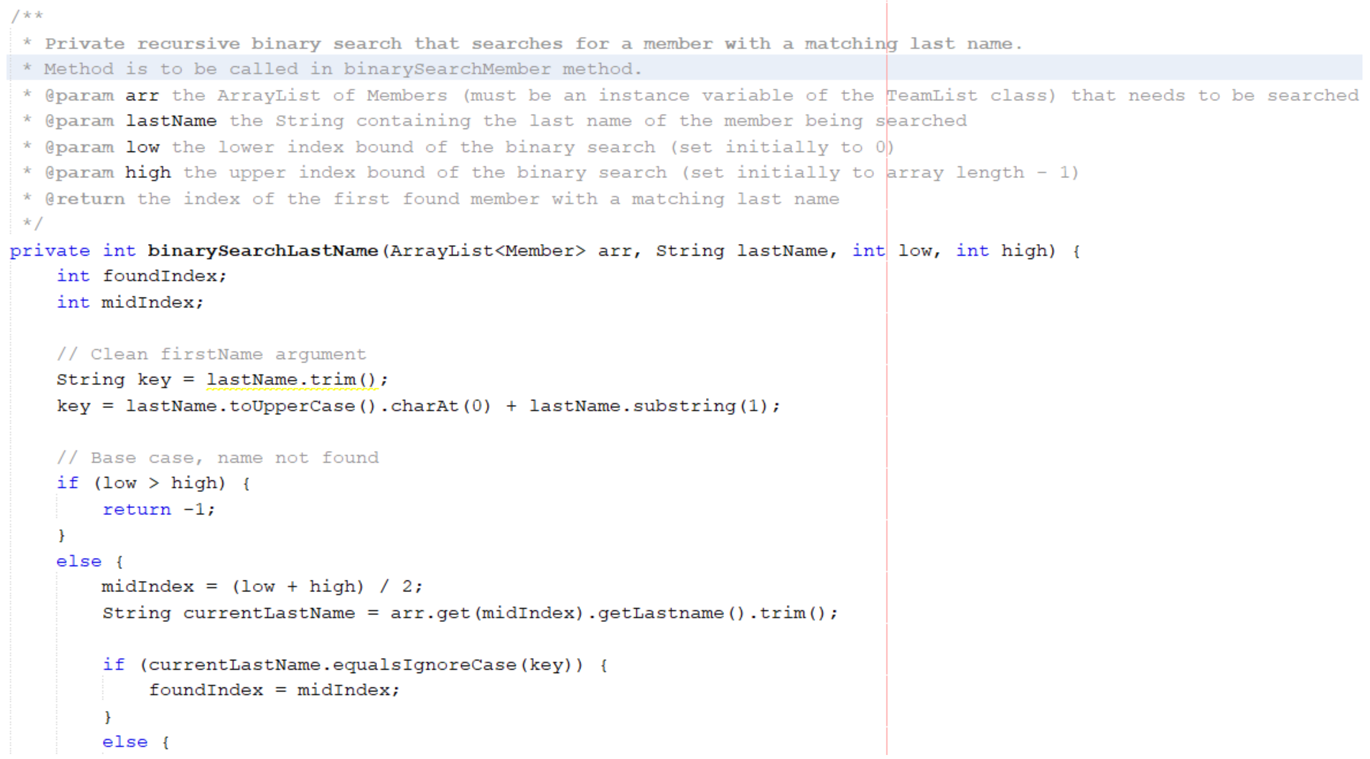
```
public boolean equals(Member m) {
    boolean equals = false;
    if (firstName.equals(m.getFirstName()) && lastName.equals(m.getLastname()) && email.equals(m.getEmail())
            && grade == m.getGrade() && subteam.equals(m.getSubteam()) && eventID.equals(m.getEventID())) {
        equals = true;
    }
    return equals;
}
```

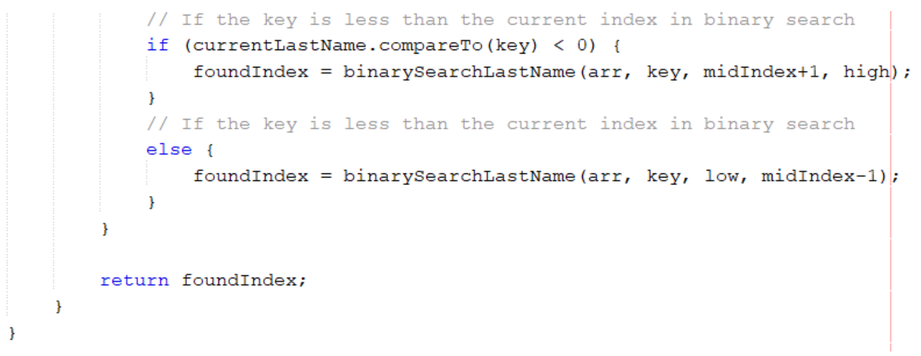
Binary Search:



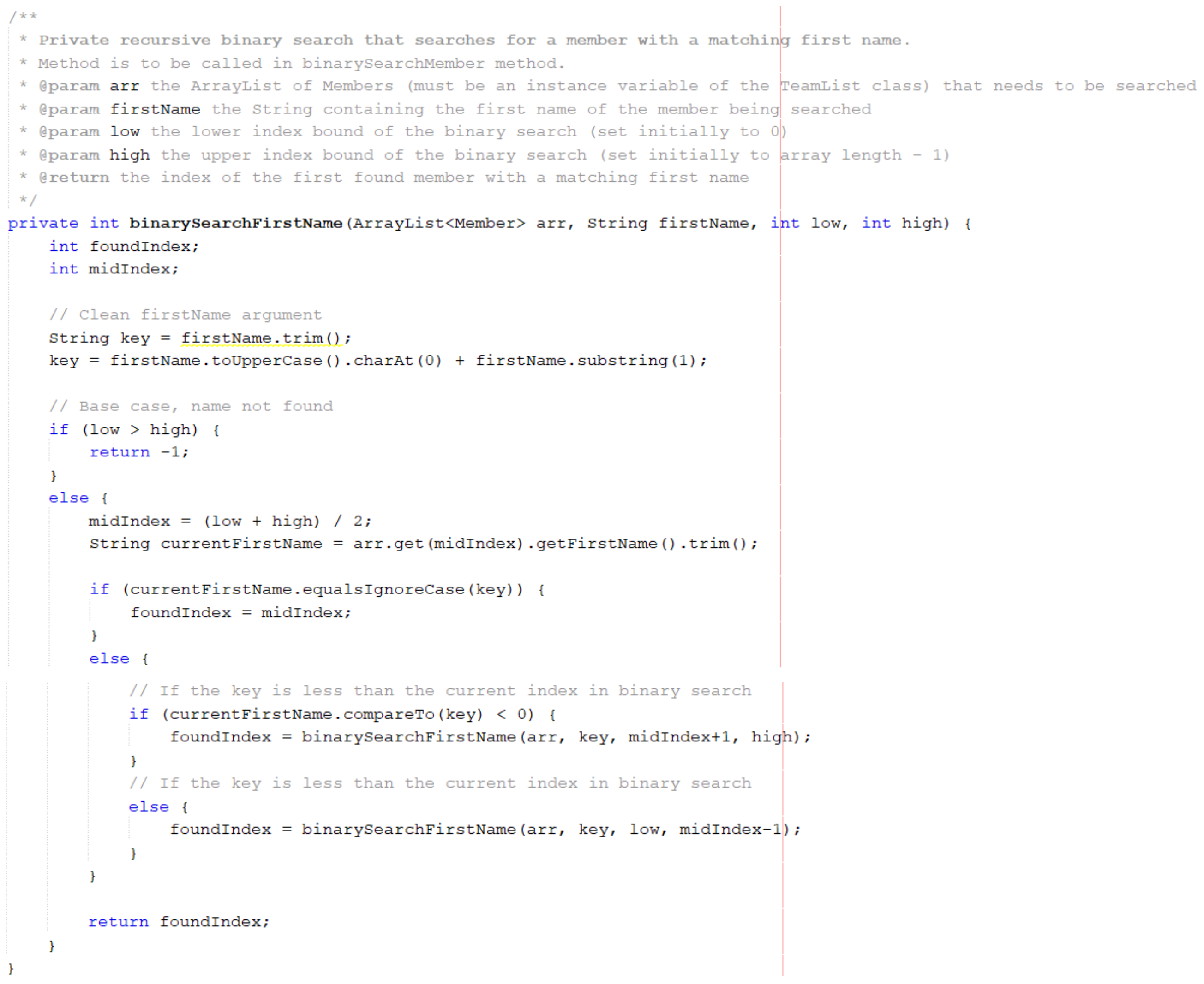
```java
/**
 * Binary search for a member based on either their first name or their last name, depending on sorting settings that are set
 * @param arr the ArrayList of Members to be searched (must be an instance variable of this class)
 * @return the index of the found name
 */
public int binarySearchMember(ArrayList<Member> arr, Member m) {

    if (sortByFirstName) {
        String firstName = m.getFirstName();
        return binarySearchFirstName(arr, firstName, 0, arr.size()-1);
    }
    else {
        String lastName = m.getLastname();
        return binarySearchLastName(arr, lastName, 0, arr.size()-1);
    }
}
```

Helper Methods:





```java
/**
 * Private recursive binary search that searches for a member with a matching last name.
 * Method is to be called in binarySearchMember method.
 * @param arr the ArrayList of Members (must be an instance variable of the TeamList class) that needs to be searched
 * @param lastName the String containing the last name of the member being searched
 * @param low the lower index bound of the binary search (set initially to 0)
 * @param high the upper index bound of the binary search (set initially to array length - 1)
 * @return the index of the first found member with a matching last name
 */
private int binarySearchLastName(ArrayList<Member> arr, String lastName, int low, int high) {
    int foundIndex;
    int midIndex;

    // Clean firstName argument
    String key = lastName.trim();
    key = lastName.toUpperCase().charAt(0) + lastName.substring(1);

    // Base case, name not found
    if (low > high) {
        return -1;
    }
    else {
        midIndex = (low + high) / 2;
        String currentLastName = arr.get(midIndex).getLastname().trim();

        if (currentLastName.equalsIgnoreCase(key)) {
            foundIndex = midIndex;
        }
        else {
            // If the key is less than the current index in binary search
            if (currentLastName.compareTo(key) < 0) {
                foundIndex = binarySearchLastName(arr, key, midIndex+1, high);
            }
            // If the key is less than the current index in binary search
            else {
                foundIndex = binarySearchLastName(arr, key, low, midIndex-1);
            }
        }

        return foundIndex;
    }
}
```

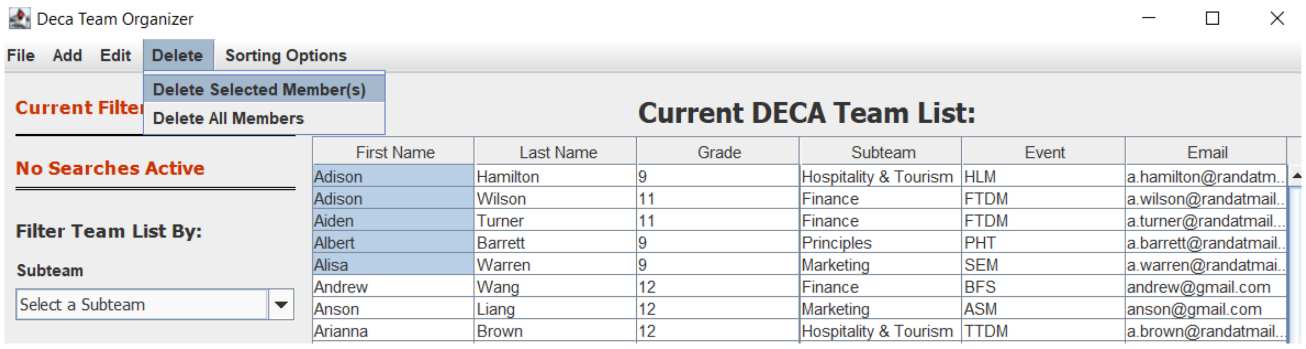


```
/**
 * Private recursive binary search that searches for a member with a matching first name.
 * Method is to be called in binarySearchMember method.
 * @param arr the ArrayList of Members (must be an instance variable of the TeamList class) that needs to be searched
 * @param firstName the String containing the first name of the member being searched
 * @param low the lower index bound of the binary search (set initially to 0)
 * @param high the upper index bound of the binary search (set initially to array length - 1)
 * @return the index of the first found member with a matching first name
 */
private int binarySearchFirstName(ArrayList<Member> arr, String firstName, int low, int high) {
    int foundIndex;
    int midIndex;

    // Clean firstName argument
    String key = firstName.trim();
    key = firstName.toUpperCase().charAt(0) + firstName.substring(1);

    // Base case, name not found
    if (low > high) {
        return -1;
    }
    else {
        midIndex = (low + high) / 2;
        String currentFirstName = arr.get(midIndex).getFirstName().trim();

        if (currentFirstName.equalsIgnoreCase(key)) {
            foundIndex = midIndex;
        }
        else {
            // If the key is less than the current index in binary search
            if (currentFirstName.compareTo(key) < 0) {
                foundIndex = binarySearchFirstName(arr, key, midIndex+1, high);
            }
            // If the key is less than the current index in binary search
            else {
                foundIndex = binarySearchFirstName(arr, key, low, midIndex-1);
            }
        }

        return foundIndex;
    }
}
```

Bulk Deletion

Bulk Deletion is handled in Gui by using a for loop to call the aforementioned methods for multiple Members. In hindsight, for maintainability I should've implemented a bulk deletion method in TeamList.

Gui Implementation:



Current DECA Team List:

First Name	Last Name	Grade	Subteam	Event	Email
Adison	Hamilton	9	Hospitality & Tourism	HLM	a.hamilton@randatm...
Adison	Wilson	11	Finance	FTDM	a.wilson@randatmail...
Aiden	Turner	11	Finance	FTDM	a.turner@randatmail...
Albert	Barrett	9	Principles	PHT	a.barrett@randatmail...
Alisa	Warren	9	Marketing	SEM	a.warren@randatmai...
Andrew	Wang	12	Finance	BFS	andrew@gmail.com
Anson	Liang	12	Marketing	ASM	anson@gmail.com
Arianna	Brown	12	Hospitality & Tourism	TTDM	a.brown@randatmail...

```java
/**
 * Method for deleting selected members from the table upon clicking of the "delete selected members" menu item
 * @param evt
 */
private void deleteSelectedActionPerformed(java.awt.event.ActionEvent evt) {
    String deleted = "";
    String failed = "";

    // First check if there is a row selection on the table
    if (!displayTable.getSelectionModel().isSelectionEmpty()) {

        // If row(s) are selected in the table, ask for confirmation that the user would like to delete the members
        if (JOptionPane.showConfirmDialog(null, "This action is irreversible. Would you like to delete the selected member(s) from the team?",
                "WARNING: Deleting Selected Member(s)", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE) == JOptionPane.YES_OPTION) {

            // If user would like to delete members, get the indexes of the selected table rows to delete
            int[] rowsSelected = displayTable.getSelectedRows();

            // If the current table is displaying the full team list, delete the members by index in the main TeamList ArrayList
            if (mainTableShowed) {

                String tempFirstLastName;

                // Delete the selected members
                for (int i = 0; i < rowsSelected.length; i++) {
                    // Store the full name of the member to be deleted
                    tempFirstLastName = displayTable.getModel().getValueAt(rowsSelected[i], 0).toString() + " "
                            + displayTable.getModel().getValueAt(rowsSelected[i], 1).toString();

                    // deleteMember(int indexToDelete) deletes specified member and returns true if member was deleted successfully
                    // note that the index of the deleted member must be decremented by i because for every member deleted, the sorted array shifts left by 1
                    if(memberList.deleteMember(rowsSelected[i]-i)) {
                        // Record members that were deleted successfully
                        deleted = deleted + " | " + tempFirstLastName + " |";

                    }
                    else {
                        // Record members that couldn't be deleted (this shouldn't ever happen)
                        failed = failed + " | " + tempFirstLastName + " |";
                    }
                }
            }
            // If the current table is not displaying the full team list (and is instead displaying a filtered list),
            // then the members will have to be deleted by searching for the member and then deleting using TeamList class's deleteMember(Member m) method
            else {

                String fname, lname, email, subteam, event;
                int grade;

                for (int i = 0; i < rowsSelected.length; i++) {

                    fname = displayTable.getModel().getValueAt(rowsSelected[i], 0).toString();
                    lname = displayTable.getModel().getValueAt(rowsSelected[i], 1).toString();
                    grade = Integer.valueOf(displayTable.getModel().getValueAt(rowsSelected[i], 2).toString());
                    subteam = displayTable.getModel().getValueAt(rowsSelected[i], 3).toString();
                    event = displayTable.getModel().getValueAt(rowsSelected[i], 4).toString();
                    email = displayTable.getModel().getValueAt(rowsSelected[i], 5).toString();

                    // Update list of deleted members / members that failed to delete
                    if(memberList.deleteMember(new Member (fname, lname, email, grade, subteam, event))) {
                        deleted = deleted + " | " + fname + " " + lname + " |";
                    }
                    else {
                        failed = failed + " |" + fname + " " + lname + " |";
                    }
                }
            }
```

```
                // Show message of who was deleted
                if (failed.equals("")) {
                    // Normal info message that shows the deleted members
                    JOptionPane.showMessageDialog(null, "Successfully deleted these members: " + deleted, "Deleted Information", JOptionPane.INFORMATION_MESSAGE);
                }
                else {
                    // Error handling: if a member isn't found, show error
                    JOptionPane.showMessageDialog(null, "Successfully deleted these members: " + deleted + "\n However, failed to delete these members: ",
                            "Critical Error", JOptionPane.INFORMATION_MESSAGE);
                }

                // Reset container panel
                showContainerPanel();
            }
        }
        // Error handling: If no table rows were selected, display error message
        else {
            JOptionPane.showMessageDialog(null, "You must select the Member(s) you would like to delete from the table.\nTo select a Member, click the "
                    + "Member's name on the table displayed below.\nYou can also hold \"Control\" or \"Command\" while clicking to select multiple members.",
                    "ERROR: No Members Selected", JOptionPane.ERROR_MESSAGE);
        }

        // clear any existing table selections
        displayTable.clearSelection();
}
```

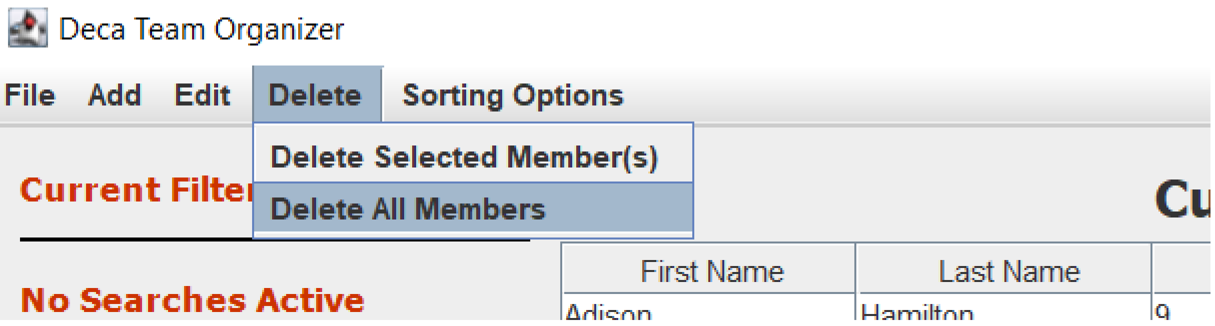
Deleting All Members

TeamList.deleteAll()

```
    /**
     * Delete all members in the team by re-initializing all instance variables
     */
    public void deleteAll() {
        initialize();
    }
```

Re-initializing TeamList instance variables is more efficient than iterating through the members to delete individually.

Gui Implementation:

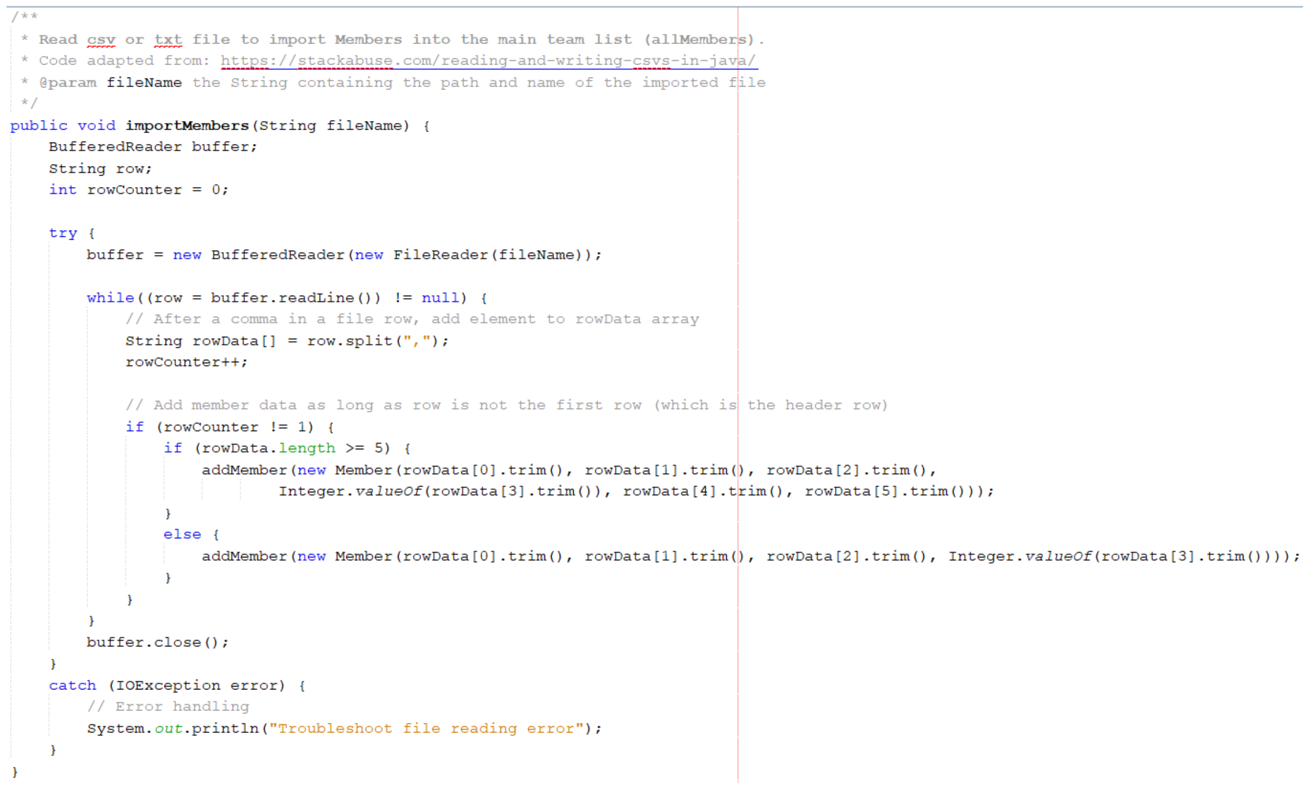


```
private void deleteAllMenuActionPerformed(java.awt.event.ActionEvent evt) {
    // Delete all members
    if (JOptionPane.showConfirmDialog(null, "This action is irreversible. Are you sure you would like to delete all members from the team?",
            "WARNING: Deleting All Members", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE) == JOptionPane.YES_OPTION) {
        memberList.deleteAll();
        memberList.exportMembers();
        memberList.importMembers(TeamList.STORAGE_FILE);
        populateTable();

    }
}
```

File Reading and Writing

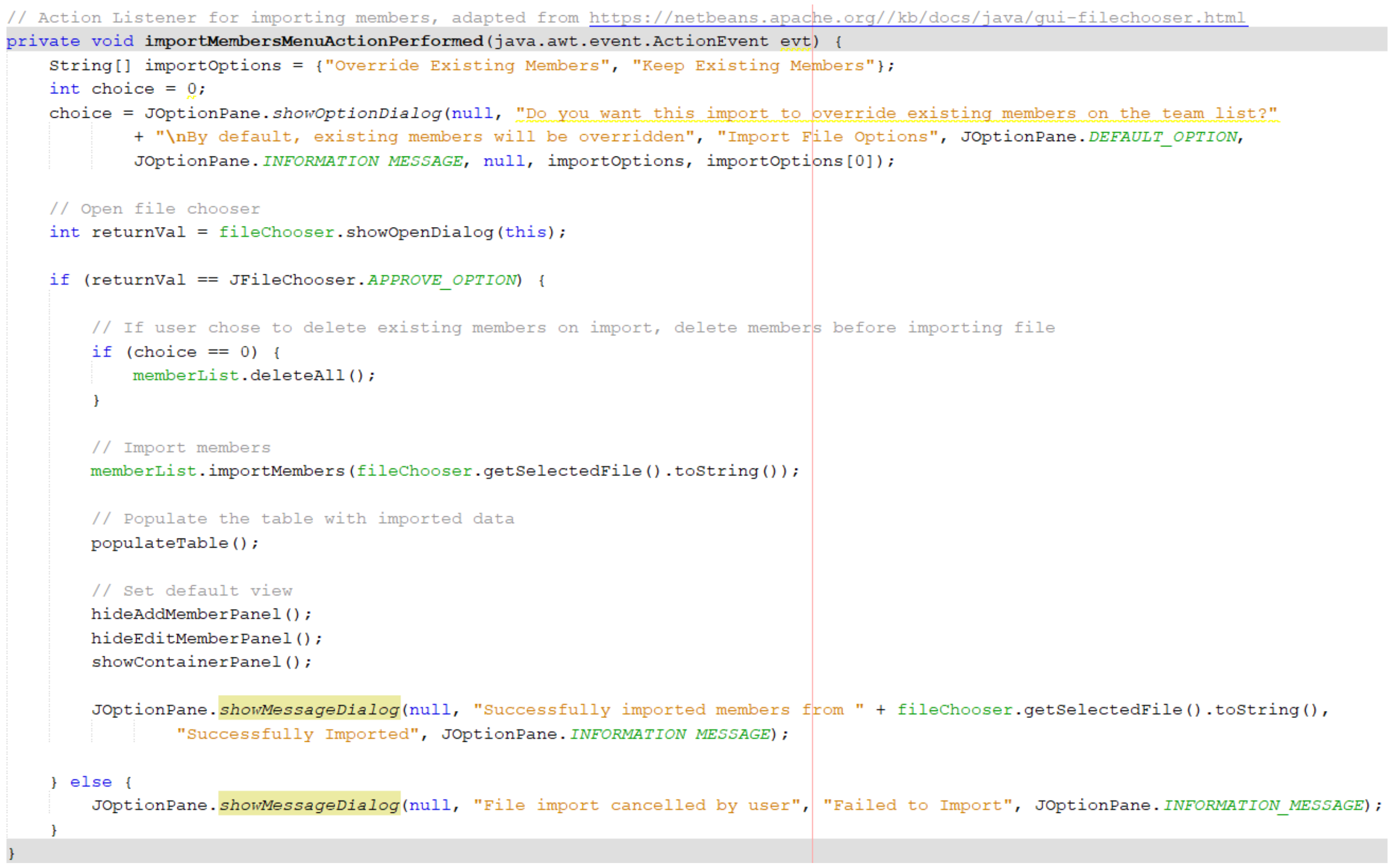
Importing Members

Importing Members in TeamList:



```
/**
 * Read csv or txt file to import Members into the main team list (allMembers).
 * Code adapted from: https://stackabuse.com/reading-and-writing-csvs-in-java/
 * @param fileName the String containing the path and name of the imported file
 */
public void importMembers(String fileName) {
    BufferedReader buffer;
    String row;
    int rowCounter = 0;

    try {
        buffer = new BufferedReader(new FileReader(fileName));

        while((row = buffer.readLine()) != null) {
            // After a comma in a file row, add element to rowData array
            String rowData[] = row.split(",");
            rowCounter++;

            // Add member data as long as row is not the first row (which is the header row)
            if (rowCounter != 1) {
                if (rowData.length >= 5) {
                    addMember(new Member(rowData[0].trim(), rowData[1].trim(), rowData[2].trim(),
                            Integer.valueOf(rowData[3].trim()), rowData[4].trim(), rowData[5].trim()));
                }
                else {
                    addMember(new Member(rowData[0].trim(), rowData[1].trim(), rowData[2].trim(), Integer.valueOf(rowData[3].trim())));
                }
            }
        }
        buffer.close();
    }
    catch (IOException error) {
        // Error handling
        System.out.println("Troubleshoot file reading error");
    }
}
```

Gui Import Members:



```java
// Action Listener for importing members, adapted from https://netbeans.apache.org//kb/docs/java/gui-filechooser.html
private void importMembersMenuActionPerformed(java.awt.event.ActionEvent evt) {
    String[] importOptions = {"Override Existing Members", "Keep Existing Members"};
    int choice = 0;
    choice = JOptionPane.showOptionDialog(null, "Do you want this import to override existing members on the team list?"
            + "\nBy default, existing members will be overridden", "Import File Options", JOptionPane.DEFAULT_OPTION,
            JOptionPane.INFORMATION_MESSAGE, null, importOptions, importOptions[0]);

    // Open file chooser
    int returnVal = fileChooser.showOpenDialog(this);

    if (returnVal == JFileChooser.APPROVE_OPTION) {

        // If user chose to delete existing members on import, delete members before importing file
        if (choice == 0) {
            memberList.deleteAll();
        }

        // Import members
        memberList.importMembers(fileChooser.getSelectedFile().toString());

        // Populate the table with imported data
        populateTable();

        // Set default view
        hideAddMemberPanel();
        hideEditMemberPanel();
        showContainerPanel();

        JOptionPane.showMessageDialog(null, "Successfully imported members from " + fileChooser.getSelectedFile().toString(),
                "Successfully Imported", JOptionPane.INFORMATION_MESSAGE);

    } else {
        JOptionPane.showMessageDialog(null, "File import cancelled by user", "Failed to Import", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

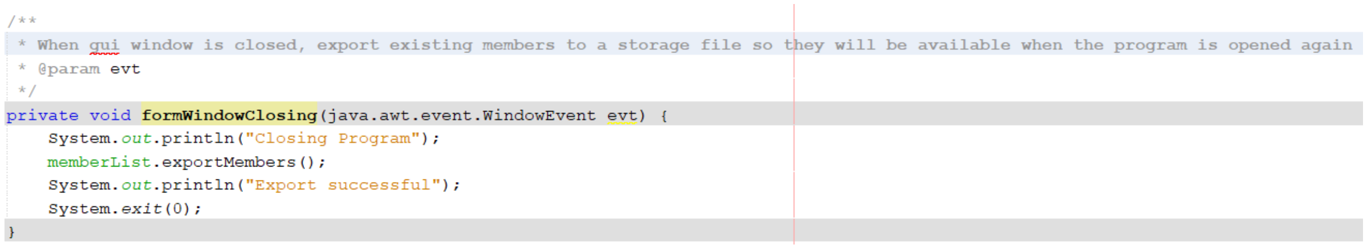
TeamList.exportMembers:

```java
/**
 * Write csv file to export members to the storage file.
 * Code adapted from: https://stackabuse.com/reading-and-writing-csvs-in-java/
 */
public void exportMembers() {

    try {
        FileWriter writer = new FileWriter(STORAGE_FILE);

        // First delete existing file before writing new file
        File file = new File(STORAGE_FILE);
        file.delete();

        // Write file header
        writer.append("First Name");
        writer.append(",");
        writer.append("Last Name");
        writer.append(",");
        writer.append("Email");
        writer.append(",");
        writer.append("Grade");
        writer.append(",");
        writer.append("Subteam");
        writer.append(",");
        writer.append("Event ID");

        // Write rest of file
        for (int i = 0; i < allMembers.size(); i++) {
            writer.append("\n");
            writer.append(allMembers.get(i).getFirstName() + ",");
            writer.append(allMembers.get(i).getLastname() + ",");
            writer.append(allMembers.get(i).getEmail() + ",");
            writer.append(allMembers.get(i).getGrade() + ",");
            writer.append(allMembers.get(i).getSubteam() + " ,");
            writer.append(allMembers.get(i).getEventID() + " ");
        }

        writer.flush();
        writer.close();
    }

    catch (IOException error) {
        // Catch error so program doesn't crash
        System.out.println("Error in exporting file");
    }
}
```

Automatic Program Saving

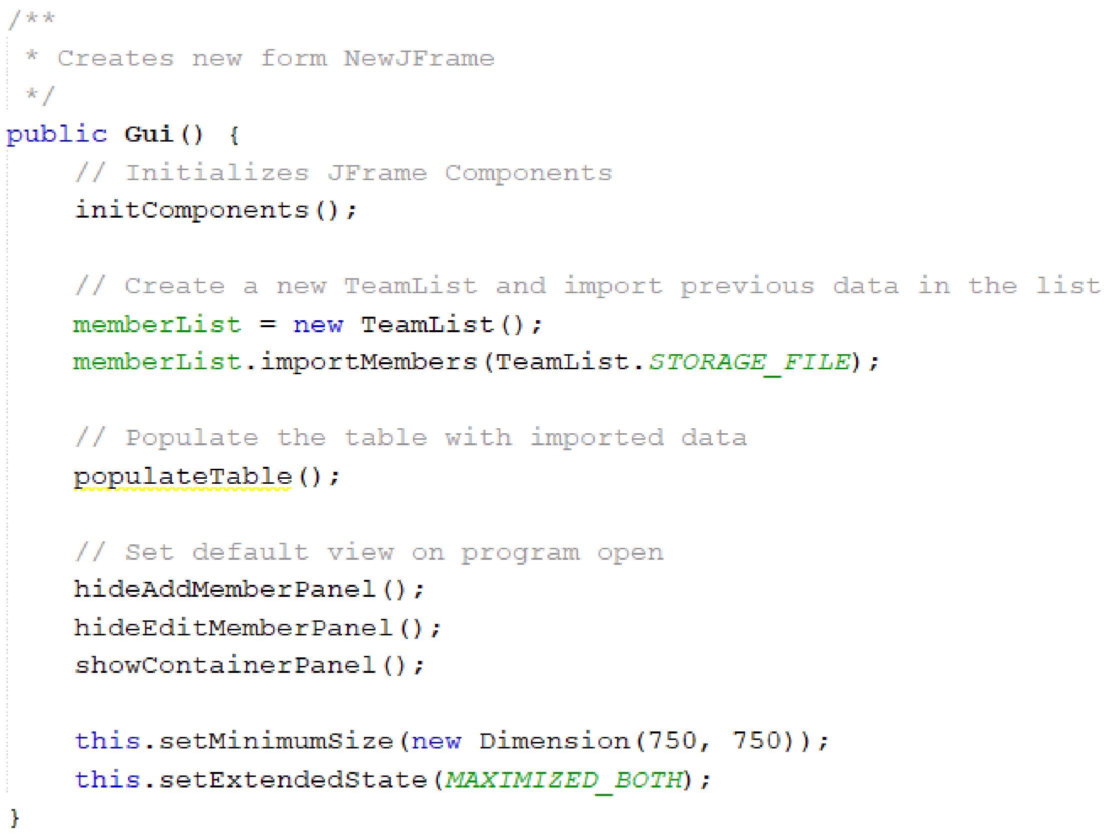
When program is closed, it exports Members to a storage CSV file using an ActionListener. When the program is opened, the Gui constructor imports Members from the storage file if it exists.

Gui Automatic Export:



```java
/**
 * When gui window is closed, export existing members to a storage file so they will be available when the program is opened again
 * @param evt
 */
private void formWindowClosing(java.awt.event.WindowEvent evt) {
    System.out.println("Closing Program");
    memberList.exportMembers();
    System.out.println("Export successful");
    System.exit(0);
}
```

Gui Constructor



```java
/**
 * Creates new form NewJFrame
 */
public Gui() {
    // Initializes JFrame Components
    initComponents();

    // Create a new TeamList and import previous data in the list
    memberList = new TeamList();
    memberList.importMembers(TeamList.STORAGE_FILE);

    // Populate the table with imported data
    populateTable();

    // Set default view on program open
    hideAddMemberPanel();
    hideEditMemberPanel();
    showContainerPanel();

    this.setMinimumSize(new Dimension(750, 750));
    this.setExtendedState(MAXIMIZED_BOTH);
}
```

Word Count: 1029