Computer Science
Internal Assessment

Trip Organizer

Word Count: 1893

Criterion A - Problem Specification

Problem Scenario

My mom is a school teacher who organizes 1 or 2 overnight trips within the country each year. These trips are highly engaging but come with their own set of challenges, particularly when it comes to assigning students to hotel rooms and buses. Each bus has varying seat numbers, ranging from 10 to 40, and hotels can accommodate different numbers of students per room, from 2 to 6. As the trip leader, my mom spends a significant amount of time assigning students to buses and rooms, considering numerous constraints. Some of these constraints relate to student behavior, necessitating that certain students do not share the same bus or room. My idea is to automate this process by developing an application. This application would take a list of students and constraints as input, allowing it to generate a suitable assignment of students.

Computational Context

My proposed solution will feature a Graphical User Interface and will function as a standalone application, eliminating the need for internet connectivity. The application will accept the list of students either through a file upload or by copy/pasting into a text field, along with the list of constraints. The algorithm will pair students based on bus and room capacities while adhering to specified constraints. Java is chosen as the programming language due to its robust Object-Oriented nature, supporting modularity by separating logic from the interface. Each function will be implemented as an individual method, leveraging Java's standard class libraries. To present the final assignment list, I plan to generate an HTML file that can be easily viewed in any browser and printed by the trip leader. Additionally, I aim to incorporate a feature for automatically generating and emailing the list to all trip leaders.

Words: 286

Success Criteria

1.Display the student and constraints list
2.Update the student and constraints list
3.Allocate a bus/room for each student considering the constraints
4.Search for a student to find the room/bus allocation
5.Convert the generated list to HTML suitable for printing
6.E-mail the bus/room list to all trip leaders

Criterion B - Planning

Decomposing Problems

Main Components

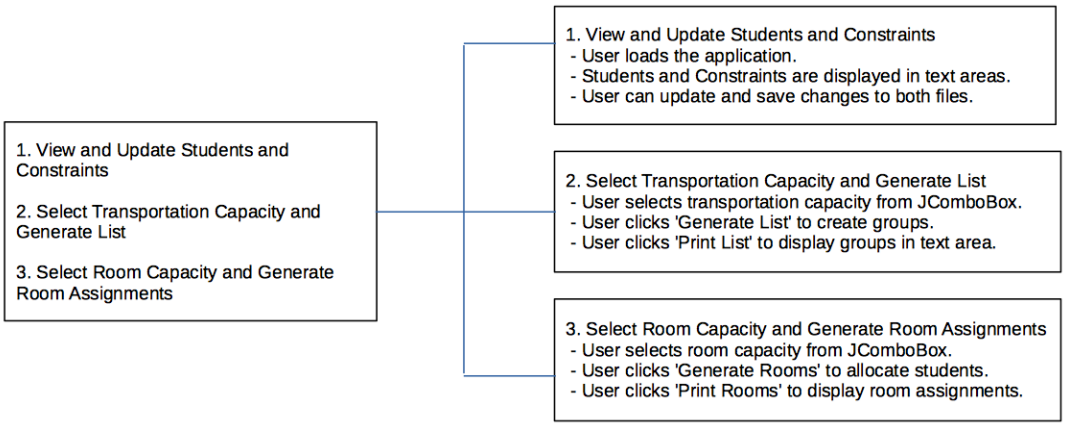
1. GUI Framework: The graphical user interface that users interact with.
2. File Handling: Reading from and writing to text files for students and constraints.
3. Room Assignment Logic: Allocating students to rooms based on constraints.
4. Transportation Management: Handling transportation capacity and generating transportation lists.

Functional Parts and Decomposition
1. GUI Framework
MainFrame Class: The main window containing all tabs and components.
Students Tab:
    Display student list from students.txt.
    Display constraints from constraints.txt.
    Update functionality for both text areas.
Transportation Tab:
    JComboBox for transportation capacity.
    Text area for displaying transportation lists.
    Generate List and Print List buttons.
Accommodation Tab:
    JComboBox for room capacity.
    Text area for displaying room assignments.
    Generate Rooms and Print Rooms buttons.

2. File Handling
    File Reading: Methods to read students.txt and constraints.txt.
    File Writing: Methods to update students.txt and constraints.txt.

3. Room Assignment Logic
RoomAssignment Class: Logic for assigning students to rooms based on constraints.
    Method to read students from a file.
    Method to read constraints from a file.
    Method to allocate students to rooms respecting constraints.
    Method to check if adding a student to a room violates constraints.

4. Transportation Management
Transportation Logic:
    JComboBox to select transportation capacity.
    Method to generate transportation list based on selected capacity.
    Method to print transportation list to the text area.


1. View and Update Students and Constraints
2. Select Transportation Capacity and Generate List
3. Select Room Capacity and Generate Room Assignments
1. View and Update Students and Constraints
- User loads the application.
- Students and Constraints are displayed in text areas.
- User can update and save changes to both files.
2. Select Transportation Capacity and Generate List
- User selects transportation capacity from JComboBox.
- User clicks 'Generate List' to create groups.
- User clicks 'Print List' to display groups in text area.
3. Select Room Capacity and Generate Room Assignments
- User selects room capacity from JComboBox.
- User clicks 'Generate Rooms' to allocate students.
- User clicks 'Print Rooms' to display room assignments.

Planning the computational thinking process

Planning. Week 1-2

Tasks:

1. Define Requirements: Identify the functional and non-functional requirements.
2. Identify Resources: List necessary resources such as Java development environment (IDE), and specific Java class libraries.
3. Research: Conduct research on generating HTML using Java, and possible algorithms for assigning students to hotel rooms.

Dependencies: None

Resources: NetBeans, Internet for research, Text editor.

Concurrency:

• Tasks 1, 2, and 3 can be conducted in parallel.

Designing. Week 2-3

Tasks:

1. Design GUI Layout: Create prototypes for the GUI.
2. Define Class Structures: Design the class structure, including the MainFrame and Model classes.
3. Plan User Interactions: Detail how users will interact with the GUI (buttons, text areas, combo boxes).

Dependencies: Completion of Planning stage

Resources: PowerPoint or any other drawing application, UML diagram tools.

Concurrency:

• Tasks 1, 2, and 3 can be done concurrently.

Developing. Week 3-5

Tasks:

1. Set Up Project Structure: Initialize the project repository and set up the development environment.
2. Implement GUI Layout: Develop the initial layout for the GUI using Swing.
3. Integrate File Handling: Implement file reading and writing functionality for students and constraints.
4. Develop Core Logic: Implement the Model class logic and integrate it with the MainFrame class.
5. Add User Interactions: Implement event listeners for buttons and other interactive components.

Dependencies: Completion of Designing stage

Resources: Java IDE, Swing documentation.

Concurrency:

• Tasks 1 and 2 can start concurrently.
• Task 3 can begin after initial progress on Task 2.
• Task 4 depends on the completion of the previous tasks.

- Task 5 can be done concurrently with Task 4 once the initial layout is done.

Testing. Week 5-6

Tasks:

1. Unit Testing: Write and execute unit tests for individual methods.
2. Integration Testing: Test the integration between GUI components and core logic.
3. User Acceptance Testing (UAT): Have users test the application and provide feedback.
4. Bug Fixing: Address any issues identified during testing.

Dependencies: Completion of Developing stage

Resources: JUnit for unit testing, Test cases, Feedback from users.

Concurrency:

- Tasks 1 and 2 can start concurrently.
- Task 3 can start once initial unit and integration testing are done.
- Task 4 is ongoing as feedback is received.

Evaluating. Week 6-7

Tasks:

1. Gather Feedback: Collect feedback from users.
2. Evaluate Performance: Assess the performance and efficiency of the application.
3. Document Findings: Document the strengths and weaknesses of the project, and identify improvement.

Dependencies: Completion of Testing stage

Resources: Feedback forms, Performance assessment tools.

Concurrency:

- Tasks 1 and 2 can start concurrently.
- Task 3 can begin once sufficient feedback is gathered.
- Task 4 can start as findings are documented.

Week/Task	1	2	3	4	5	6	7
Planning							
- Define Requirements							
- Identify Resources							
- Research							
Designing							
- Design GUI Layout							
- Define Class Structures							
- Plan User Interactions							
Developing							
- Set Up Project Structure							
- Implement GUI Layout							
- Integrate File Handling							
- Develop Core Logic							
- Add User Interactions							
Testing							
- Unit Testing							
- Integration Testing							
- User Acceptance Testing							
- Bug Fixing							
Evaluating							
- Gather Feedback							
- Evaluate Performance							
- Document Findings							

Criterion C - System Overview

System Models

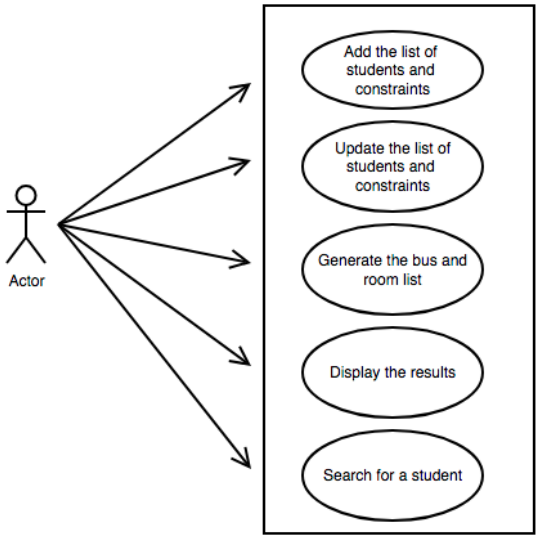


Image 1: UML Use Case Diagram

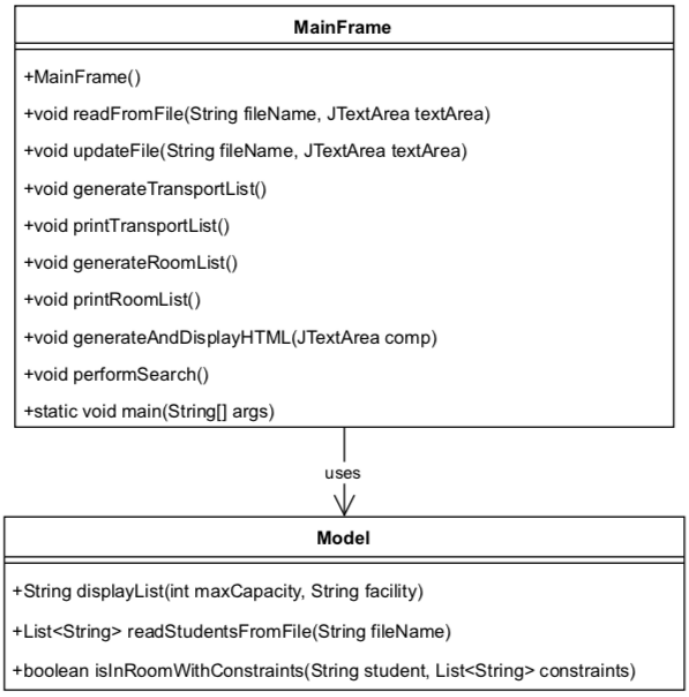


Image 2: UML Class Diagram

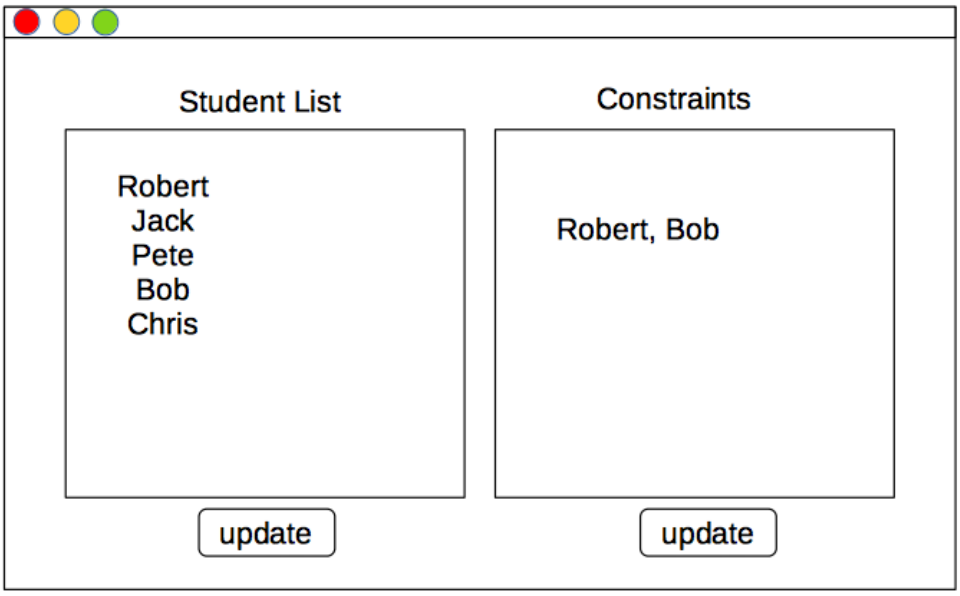


Image 3: Add and update the list of students and constraints

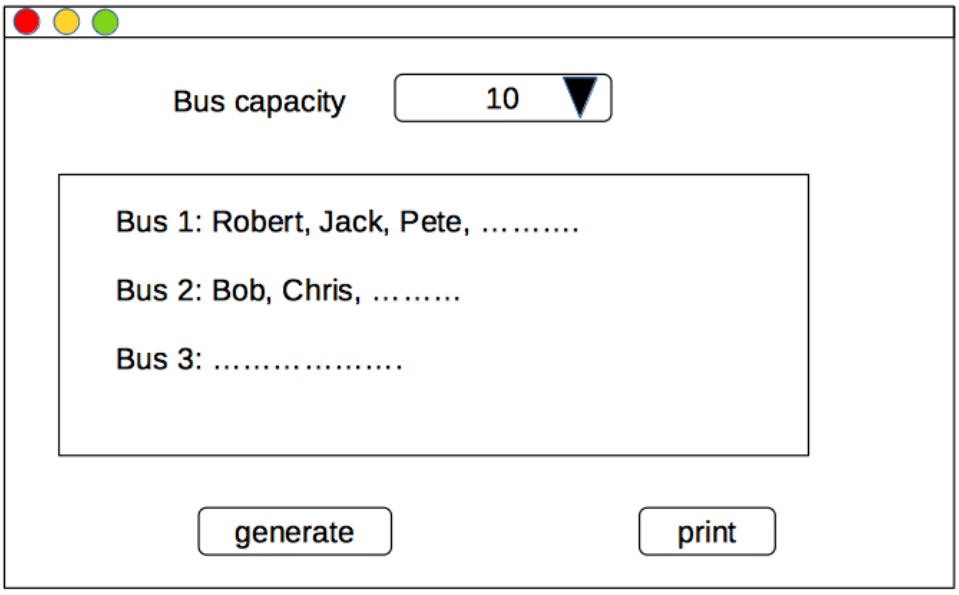


Image 4: Generate a bus list based on constraints

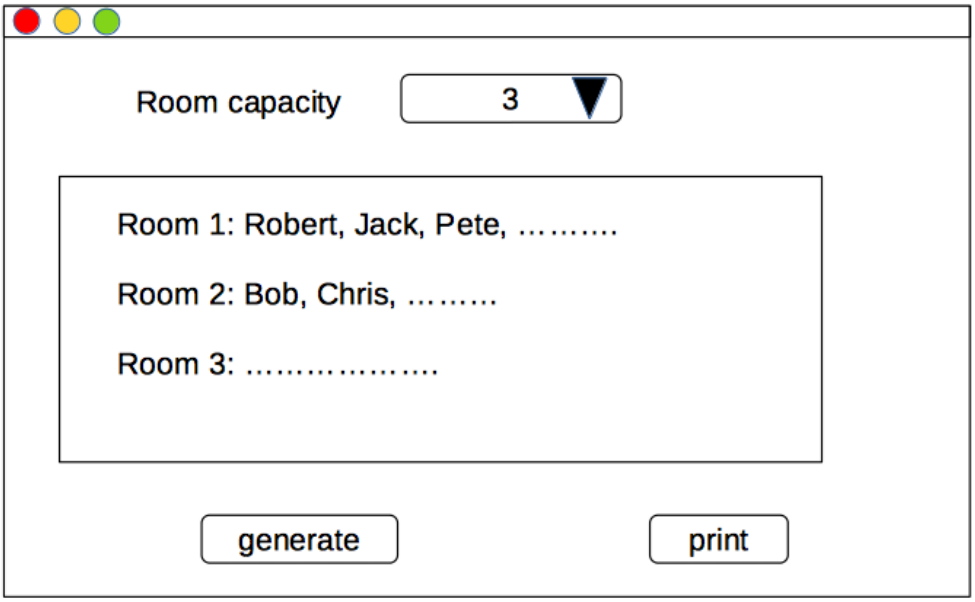


Image 5: Generate a hotel room list based on constraints

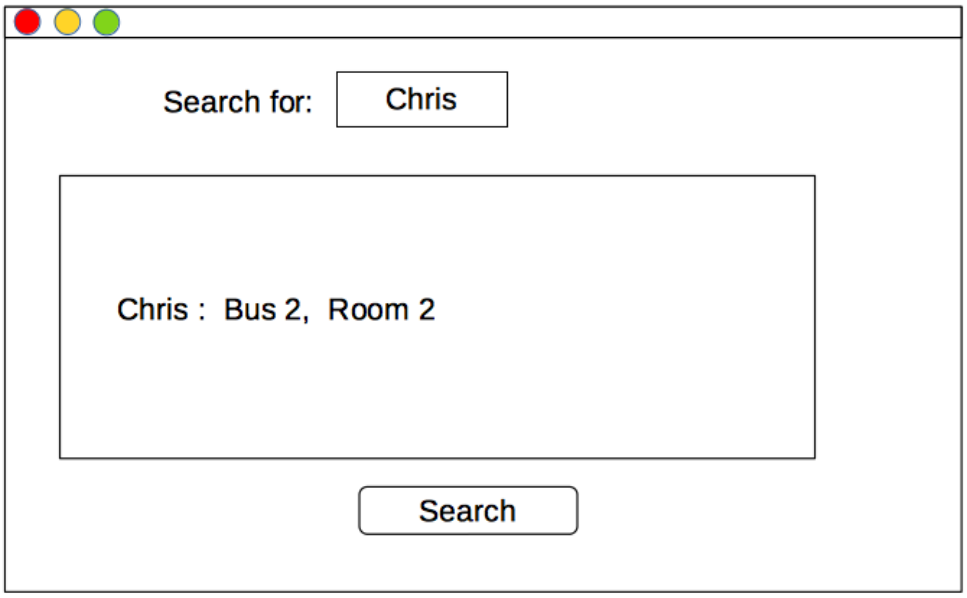


Image 6: search for a students and assigned bus/room list

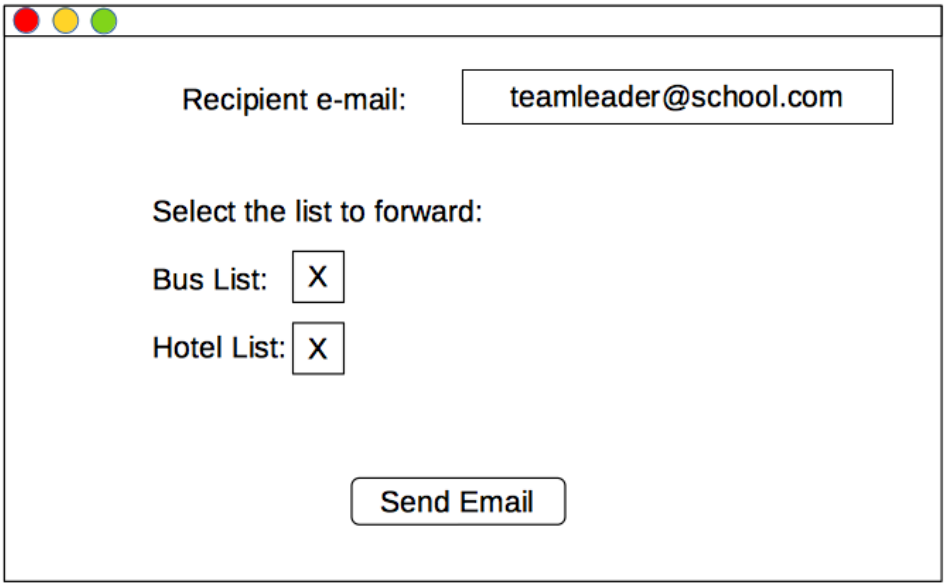


Image 7: e-mail the bus/room list

Algorithms as components in a system model

To check if adding a student to a room violates any constraints.

```
Function isInRoomWithConstraints(student, room, constraints)
    for each s in room
        if student is in constraints and s is in constraints[student]
            return true
    return false
```

To read constraints from a specified file and return them as a map where each student is mapped to a set of students they cannot room with.

```
Function readConstraintsFromFile(fileName)
    constraints = empty map
        open file reading
        while there is a next line in the file
            line = next line from file
            line = trim(line)
            if line is not empty
                studentsArray = split line by ","
                if length of studentsArray > 1
                    student1 = trim(studentsArray[0])
                    student2 = trim(studentsArray[1])
                    if student1 not in constraints
                        constraints[student1] = empty set
                    add student2 to constraints[student1]
                    if student2 not in constraints
                        constraints[student2] = empty set
                    add student1 to constraints[student2]
    return constraints
```

To read a list of students from a specified file and return it as a list.

```
Function readStudentsFromFile(fileName)
    students = empty list
    try
        open file for reading
        while there is a next line in the file
            line = next line from file
            line = trim(line)
            if line is not empty
                add line to students
    return students
```

To allocate students to rooms based on the maximum capacity and facility constraints, then return a formatted string of the room allocations.

1.Reads a list of students from a file.
2.Reads constraints (students who should not be in the same room) from a file.
3.Calculates the number of rooms needed.
4.Allocates students to rooms while respecting constraints.
5.Output the result

```
Function displayList(maxCapacity)
    students = readStudentsFromFile("students.txt")
    constraints = readConstraintsFromFile("constraints.txt")

    numStudents = length of students
    numRooms = ceiling(numStudents / maxCapacity)

    roomsList = ArrayList of rooms. Each room contains students
    remainingStudents = ArrayList of students

    for i from 0 to numRooms-1
        room = empty list
        iter = iterator of remainingStudents
        while iter has next and length of room < maxCapacity
            student = next element from iter
            if not isInRoomWithConstraints(student, room, constraints)
                add student to room
                remove student from remainingStudents

        add room to roomsList

     Output the content of roomsList
```

To search for a query in the transportation and accommodation lists and display the results.

```
Function performSearch()
    input(studentName)
    Get text from transportationListTextArea and convert to lowercase
    Get text from roomListTextArea and convert to lowercase
    Split transportation text by new lines into transportationLines
    Split accommodation text by new lines into accommodationLines
    For each line in transportationLines
        If line contains query
            Append "Bus Assignment: " and line to results
            Break
    For each line in accommodationLines
        If line contains query
            Append "Room Assignment: " and line to results
            Break
    Output line
```

To generate and display an HTML representation

```
1.Split the text of the JTextArea into lines.
2.Build an HTML string with the lines formatted as HTML.
3.Write the HTML string to a temporary file.
4.Open the temporary file in the default web browser.
```

To send the generated list by e-mail to the specific recipient:

```
Function sendEmailWithContent(textAreaContent, recipientEmail)
        Initialize email properties

        Create an authenticator with username and password
        Create a new session with email properties and authenticator

        Create a new MimeMessage with the session
        Set the From, To, Subject fields to your email address
        Create a Transport object to send the message
        Call send method on the Transport object with the message

        Print "Email sent successfully."
```

Words: 80

Testing Strategies

Success Criteria	Testing Strategy	Expected Result
1.Display the student and constraints list	Visually inspect if the list of students and constraints are displayed.	Upon running the program the list of students and constraints will be read from the text file and displayed in text areas
2.Update the student and constraints list	Modify the list of students and constraints directly in the text field and press the update button. When the application is restarted the updated list is displayed	Updated list is displayed when the application is restarted.
3.Allocate a bus/room for each student considering the constraints	Visually check, to confirm that the generated list does not contradict the constraints.	No students in the constraints list should be on the same bus and assigned to the same room
4.Search for a student to find the room/bus allocation	Enter the student's name. If the student is found, the bus and room numbers will be displayed	The bus number and the room number of the student found will match the generated bus and room list
5.Convert the generated list to HTML suitable for printing	Press the Print and check if the bus and room lists are displayed as HTML in a browser	The browser will automatically open and will display the bus and room list, which later could be printed out.
6.E-mail the bus/room list to all trip leaders	Enter the recipient e-mail, and check available options. Press the E-mail button, and check if the recipient received an e-mail that contains the bus and room lists	The recipient will receive and e-mail containing the bus and room list

Criterion D - Development

List<String> and List<List<String>>

```
17          // Allocate students to rooms respecting constraints
18          List<List<String>> rooms = new ArrayList<>();
19          List<String> remainingStudents = new ArrayList<>(students); // Copy of students list
20          for (int i = 0; i < numRooms; i++) {
21              List<String> room = new ArrayList<>();
22              Iterator<String> iter = remainingStudents.iterator();
23              while (iter.hasNext() && room.size() < maxCapacity) {
24                  String student = iter.next();
25                  if (!isInRoomWithConstraints(student, room, constraints)) {
26                      room.add(student);
27                      iter.remove(); // Remove from remaining students
28                  }
29              }
30              rooms.add(room);
```

Appendix.pdf, page 1:
The `remainingStudents` list is a copy of the original `students` list. This ensures that the original list remains unchanged, which is important for preserving data integrity. The `remainingStudents` list can be dynamically updated (students can be removed as they are assigned to rooms), which is necessary for the iterative allocation process. A `List` is a suitable data structure for this purpose because it allows random access and efficient removal of elements, both of which are required for this algorithm.

Each room is represented as a `List<String>`, and `rooms` is a list of these room lists. This structure naturally represents the hierarchical relationship where each room contains multiple students. Using a list of lists allows easy access and manipulation of individual rooms and their contents. For example, it is straightforward to iterate over each room and each student within a room. The number of rooms is determined at runtime, and using a list of lists allows the number of rooms to be dynamically adjusted as needed during the allocation process.

`List<String> remainingStudents` is used to keep track of students who have not yet been assigned to a room, allowing for dynamic updates and ensuring that the original list of students remains unchanged.
`List<List<String>> rooms` is used to represent the allocation of students into rooms, providing a clear and flexible hierarchical structure that can dynamically grow and be easily manipulated.
The testing strategy effectively tests this feature, to make sure there are no violations of constraints.

Map<String, Set<String>> constraints

```
4   public class Model {
5
6       public String displayList(int maxCapacity, String facility) {
7           // List of students read from file
8           List<String> students = readStudentsFromFile("students.txt");
9
10          // Constraints map read from file
11          Map<String, Set<String>> constraints = readConstraintsFromFile("constraints.txt");
12
13          // Calculate number of rooms needed
14          int numStudents = students.size();
15          int numRooms = (int) Math.ceil((double) numStudents / maxCapacity);
16
```

Appendix.pdf, page 1:
A `Map` provides O(1) average time complexity for lookups, which means you can quickly check if a student has any constraints and retrieve the corresponding set of students they should not be roomed with. Each key in the map represents a student, and the value is a `Set` of students that the key student should not be roomed with. This creates a clear and direct mapping of constraints.
Using a `Set` for the values ensures that each constrained student is listed only once. This prevents duplicate entries and maintains a clean set of constraints.
The `Set` automatically handles duplicate entries, so you don't have to manually check for and remove duplicates.
The `Map<String, Set<String>>` structure is well-suited for representing many-to-many relationships, which is essential when multiple students have constraints with multiple other students.
The testing strategy effectively tests this feature, to make sure there are no violations of constraints.

The readConstraintsFromFile method

```
59      // Read constraints from file
60      private static Map<String, Set<String>> readConstraintsFromFile(String fileName) {
61          Map<String, Set<String>> constraints = new HashMap<>();
62          try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
63              String line;
64              while ((line = reader.readLine()) != null) {
65                  line = line.trim();
66                  if (!line.isEmpty()) {
67                      String[] studentsArray = line.split(",");
68                      if (studentsArray.length > 1) {
69                          String student1 = studentsArray[0].trim();
70                          String student2 = studentsArray[1].trim();
71                          constraints.computeIfAbsent(student1, k -> new HashSet<>()).add(student2);
72                          constraints.computeIfAbsent(student2, k -> new HashSet<>()).add(student1);
73                      }
74                  }
75              }
76          } catch (IOException e) {
77              System.err.println("Error reading file: " + e.getMessage());
78          }
79          return constraints;
80      }
```

Appendix.pdf, page 2:
The `readConstraintsFromFile` method reads constraints from a file and returns them as a map where each key is a student and the corresponding value is a set of students they cannot room with. A new `HashMap` is created to store the constraints. Each key in the map represents a student, and the value is a `Set` of students they cannot room with. For each valid constraint (lines with more than one student), the student names are added to each other's constraint sets using `computeIfAbsent` to ensure the sets are initialized if they do not already exist.
The testing strategy effectively tests this feature, to make sure there are no violations of constraints.

Lambda Expression

```
71                      constraints.computeIfAbsent(student1, k -> new HashSet<>()).add(student2);
72                      constraints.computeIfAbsent(student2, k -> new HashSet<>()).add(student1);
73                  }
```

Appendix.pdf, page 1:
The lambda expression `k -> new HashSet<>()` provides a concise way to define the mapping function inline without the need for a separate method or class. This makes the code shorter and easier to read. The lambda expression clearly shows that if the key is absent, a new `HashSet` should be created. This makes the intention of the code immediately apparent to anyone reading it.

Lambda expressions have also been used here:
Appendix.pdf, page 7:

```
161             updateStudentsButton.addActionListener(e -> updateFile("students.txt", studentsTextArea));
162             updateConstraintsButton.addActionListener(e -> updateFile("constraints.txt", constraintsTextArea));
163             generateTransportListButton.addActionListener(e -> generateTransportList());
164             printTransportListButton.addActionListener(e -> printTransportList());
165             generateRoomListButton.addActionListener(e -> generateRoomList());
166             printRoomListButton.addActionListener(e -> printRoomList());
167             searchButton.addActionListener(e -> performSearch());
```

This makes the code shorter and easier to read.
The testing strategy has no provision to test this programming technique. There is nothing to test

Building HTML Content:

```
224     private void generateAndDisplayHTML(JTextArea comp) {
225         String[] lines = comp.getText().split("\\n");
226         StringBuilder contentBuilder = new StringBuilder();
227         contentBuilder.append("<html><head><title>Student List</title><style>")
228                 .append("body { margin: 30px 50px; font-size: 20px; }") // Increased margins and font size
229                 .append("</style></head><body><h1>Student List</h1><div>");
230 
231         for (String line : lines) {
232             contentBuilder.append(line).append("<br/>");
233         }
234 
235         contentBuilder.append("</div></body></html>");
236 
237         String htmlContent = contentBuilder.toString();
238 
239         try {
240             File tempFile = File.createTempFile("student_list", ".html");
242             FileWriter writer = new FileWriter(tempFile);
243             writer.write(htmlContent);
              writer.close();
```

Appendix.pdf, page 9:
The `StringBuilder` is used to efficiently construct the HTML content.
The `append` method is used to add HTML tags and CSS styles to the content. The `<style>` tag inside the `<head>` section increases the margins and font size for better readability. The `for` loop iterates over each line of text and appends it to the HTML content, followed by a `<br/>` tag to preserve the line breaks.
The HTML content is closed with `</div></body></html>` tags.
A temporary file with the prefix `"student_list"` and suffix `".html"` is created using `File.createTempFile`. Creating a temporary file ensures that the generated HTML content does not persist longer than necessary, avoiding clutter in the file system.
The use of `File.createTempFile` automatically handles the naming and location of the temporary file, making the code simpler and more reliable.
The testing strategy employed is very effective in testing this feature. The HTML file will always be created with a specific dataset.

Reading from and writing to a text file:

```
170     private void readFromFile(String fileName, JTextArea textArea) {
171         try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
172             textArea.read(reader, null);
173         } catch (IOException e) {
175             e.printStackTrace();
176         }
177     }
178 
179     private void updateFile(String fileName, JTextArea textArea) {
180         try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
181             textArea.write(writer);
182             JOptionPane.showMessageDialog(this, "File updated successfully.");
184         } catch (IOException e) {
185             e.printStackTrace();
186             JOptionPane.showMessageDialog(this, "Error updating file.");
        }
    }
```

Appendix.pdf, page 8:
The data in this application is relatively flat and simple (lists of students and constraints). Text files are well-suited for such flat data structures, compared to databases.
The `readFromFile` method reads the content of a specified file and displays it in a `JTextArea`. The `updateFile` method writes the content of a `JTextArea` to a specified file and shows a message dialog indicating whether the operation was successful or not. By reading directly into and writing directly from a `JTextArea`, these methods simplify the process of displaying and updating file content in the GUI, making it more intuitive for the user to interact with the text content.
The testing strategy employed is very effective in testing this feature. An error message is displayed if there are issues accessing the file.

Words: 1179

Criterion E - Evaluation

Meeting Success Criteria

1.Display the student and constraints list
Achieved. The student list and constraints list are successfully read from respective text files and the content is displayed in text areas. Alternatively, the user can paste the list of students and constraints directly into text areas.

2.Update the student and constraints list
Achieved. The list of students and constraints could directly be modified in the text field. Upon pressing the Update button, the list is successfully stored in the text file. Alternatively the user can modify the text file. Both options are working.

3.Allocate a bus/room for each student considering the constraints
Achieved. When the capacity of the bus and room are chosen, students are distributed in buses and rooms according to constraints. There are no violations of constraints regardless of bus and room capacities.

4.Search for a student to find the room/bus allocation
Achieved. When the user is searching for a student, the bus and room numbers are displayed where that student has been assigned to. The output also indicates the students who are on the same bus and rooms with the selected student. If the student is not found, an error message is displayed.

5.Convert the generated list to HTML suitable for printing
Achieved. This successfully generates an HTML file containing the bus and rooms distribution and automatically displays in the default browser. This allows the user to print out the content as needed.

6.E-mail the bus/room list to all trip leaders
Not Achieved. I was unable to complete this task due to time constraints.

Suggesting improvements

While this solution works fine for the given list of students, there is no provision to separate students by different grade levels or sections.This would be important to make sure that students from different grade levels are not assigned to the same room or bus. An improvement to this would be to accept students by grade levels or sections.

Another improvement would be to consider the fact that not all rooms or buses will have the same capacity. For example, there might be eight 3-bed rooms and ten 4-bed rooms or one 40-seater bus, and one 10-seater bus. This will more effectively distribute students.

The final improvement would be students' preference oa a roommate, in addition to

restrictions provided by teachers. The students may complete the survey indicating their preferences, that will be considered, but not guaranteed, while distributing students.

Words: 348