Computer Science
Internal Assessment

Birthday Card Generator

Word Count: 1850

Criterion A - Problem Specification

Problem Scenario
As class captain, I understand the importance of acknowledging students' birthdays with a thoughtful card. Currently, I maintain a spreadsheet to track student names and birthdates, and I manually create personalized cards using a word processor. While this approach ensures each card is unique, it is extremely time-consuming and inefficient. To streamline the process and ensure that every student feels celebrated on their special day, I am planning to develop an automated solution. This system will generate personalized birthday cards for selected students, incorporating different birthday messages to add a personal touch. This will save time and ensure that no student's birthday is overlooked. Moreover, if I am not at school, any other student, or my homeroom teacher, could easily use this product

Computational Context
The most suitable solution would be a standalone application, as no Internet connectivity is required. I am planning to create a graphical user interface (GUI) where I can track students' information and generate cards as needed. I intend to display the records neatly using a table.

I plan to use Java to develop this solution, as Java is platform-independent and the product will run on macOS or Windows. Additionally, Java is an object-oriented programming (OOP) language and will allow me to separate the logic from the view using the Model-View-Controller (MVC) concept. There is a vast collection of Java libraries, and I will be using one that allows me to generate PDF files. For data storage, I will likely use a CSV file, though another possibility would be an Excel spreadsheet. I will use NetBeans to create the GUI, as it will simplify the process.

Words: 272

Success Criteria
1.Display records in an organized layout
2.Add a new student
3.Data Validation
4.Calculate the student’s age
5.Search for a student
6.Sort out records by first name
7.Delete a record
8.Generate cards as PDF for the chosen student
9.Cards should have different messages

Criterion B - Planning

Decomposing Problems

1.Planning

- Define project scope and success criteria.
- Create a project timeline.
- Research and choose a suitable library for PDF generation

2.Design

- Create a system diagram.
- Design the user interface prototypes.
- Identify major algorithms

3.Implementation

- Design the Graphical User Interface (GUI), which includes Main Application Window, Input Panel, Button Panel, Table Panel
- Construct the Person Class (Attributes and Behaviors)
- Construct Database class to include add(), search(), delete(), load(), save(), display() methods
- Construct PDF Generation algorithm

4.Testing

- Unit Testing: Test individual components for correctness.
- Integration Testing: Ensure all components work together as expected.
- User Acceptance Testing (UAT): Validate the application with end-users.

5.Evaluation

- Collect feedback from users.
- Evaluate the performance and usability of the application.
- Make necessary adjustments based on feedback.

Planning the computational thinking process

Task	Duration	Dependencies	Concurrencies
Planning (1 week)			
- Define scope and success criteria	1 day		
- Create a timeline	2 days	Define scope and success criteria	
- Allocate resources and tasks	2 days	Create a timeline	
Design (1 week)			
- System architecture diagram	2 days		
- GUI design	3 days	System architecture diagram	
- Database schema	1 day	GUI design	
GUI Design (1 week)			
- Main Application Window	1 day		Database Management
- Input Panel	2 days	Main Application Window	Database Management
- Button Panel	1 day	Input Panel	Database Management
- Table Panel	1 day	Input Panel	Database Management
Database Management (3 weeks)			
- Person Class	2 days		PDF Generation
- PersonDatabase Class	10 days	Person Class	PDF Generation
PDF Generation (1 week)			

- Content Drawing, Font Loading	3 day		Database Management
Integration and Testing (1 week)			
- Integrate GUI with Database	2 days	Database Management, GUI Design	
- Integrate PDF Generation	1 day	Integrate GUI with Database	
- Unit Testing	1 day	Integrate PDF Generation	
- Integration Testing	2 days	Unit Testing	
- User Acceptance Testing (UAT)	2 days	Integration Testing	
Evaluation (1 week)			
- Collect feedback	1 day	User Acceptance Testing	
- Performance evaluation	1 day	Collect feedback	
- Final adjustments	1 day	Performance evaluation	

Criterion C - System Overview

System Models

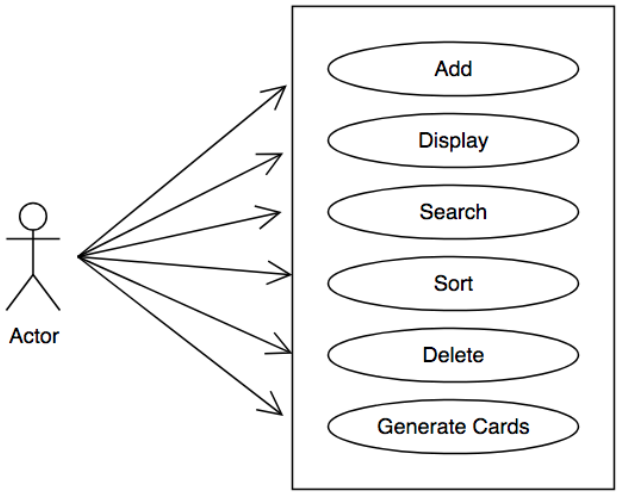

Image 1: UML Use Case Diagram

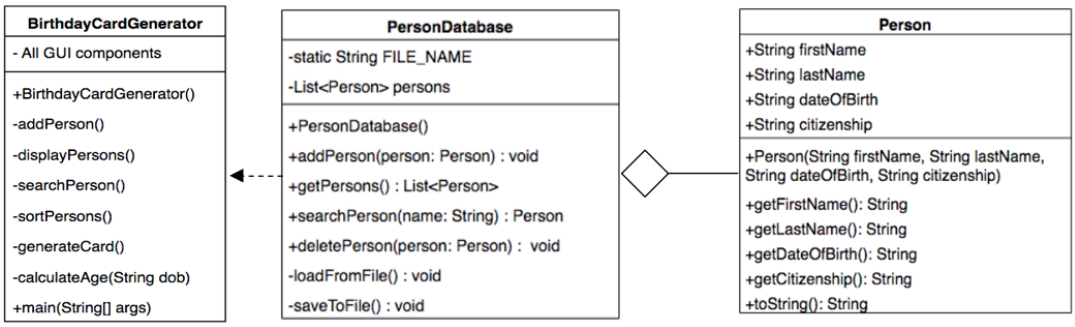

Image 2: UML Class Diagram

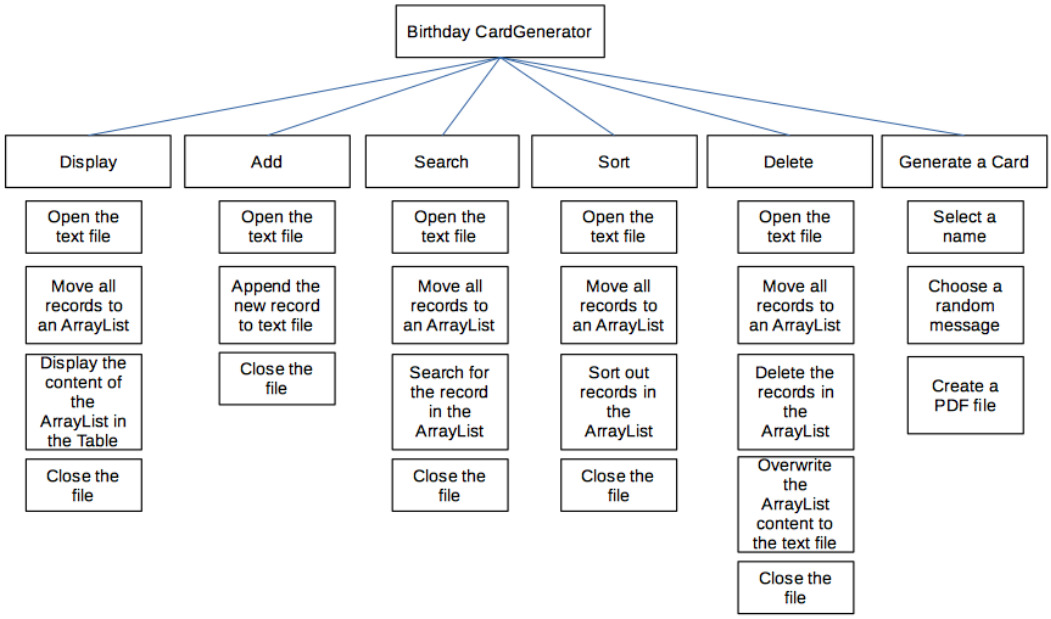

Image 3: UML Class Diagram

An ArrayList is used to store the records in the memory for data manipulation, such as for Adding, Deleting, Searching,Sorting and Displaying Records.

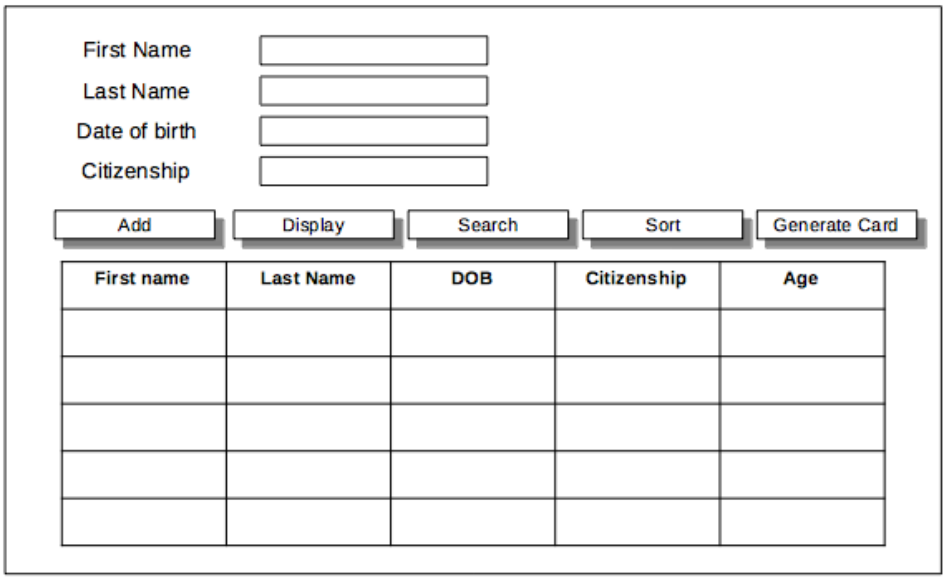

Image 4: User Interface Prototype

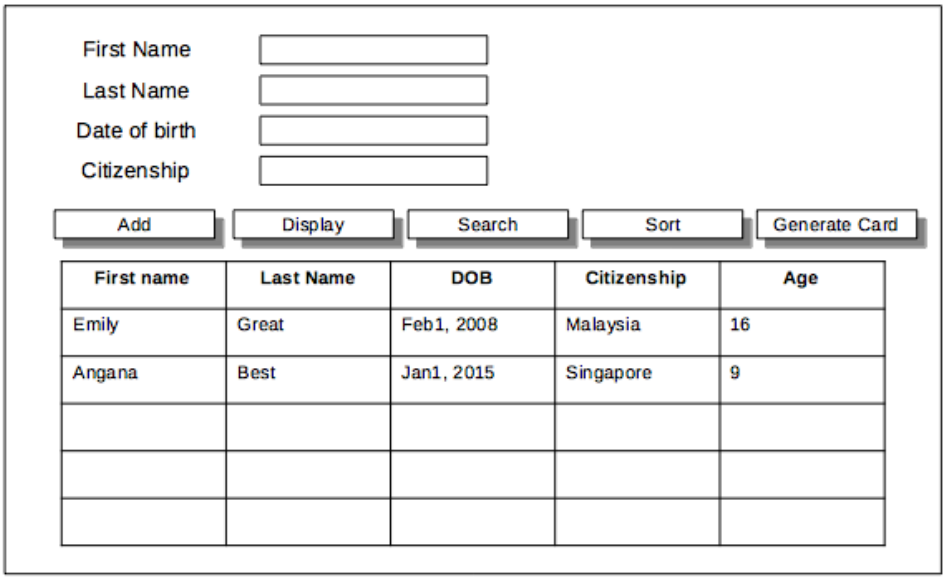

Image 5: User Interface Prototype including sample data set

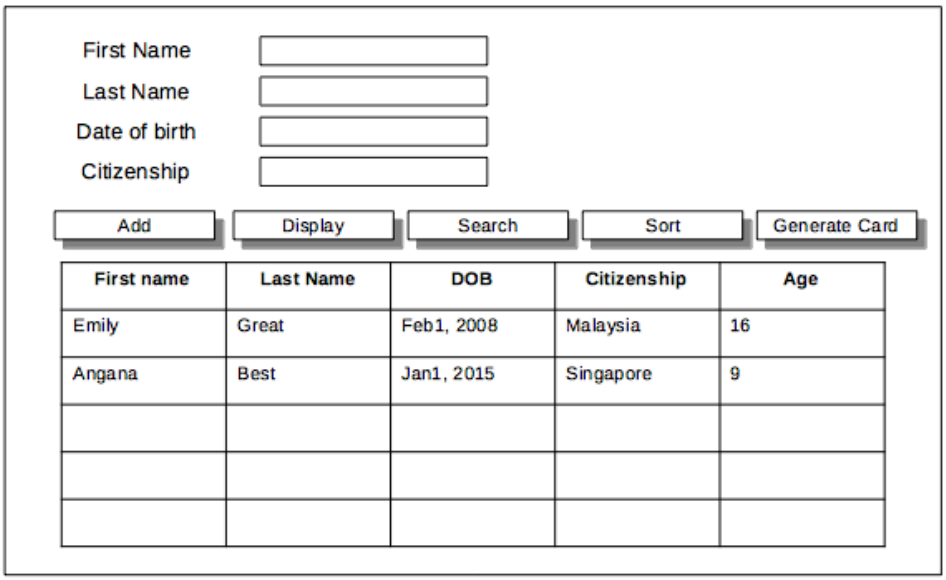

Image 6: User Interface Prototype including sample data set and age:

The age of the student is automatically calculated based on computer's current date and student's date of birth

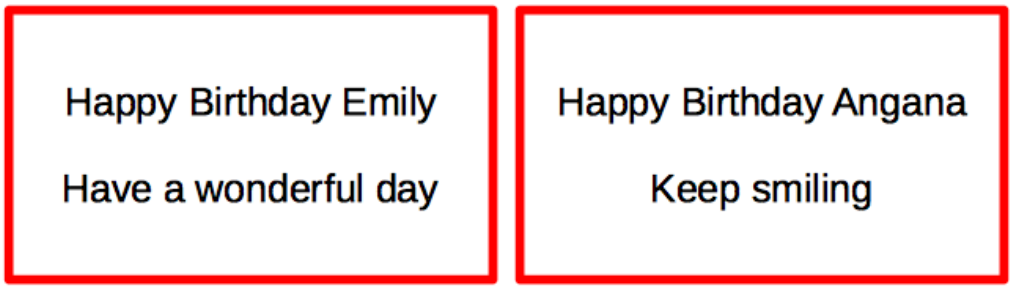

Image 7: Examples of personalized birthday cards

Birthday cards will be personalized to include the student's first name, and random greeting message.

Algorithms as components in a system model

An ArrayList is used to store the records in the memory for data manipulation, such as for Adding, Deleting, Searching,Sorting and Displaying Records.

```
method add()
{
     record=input(firstName, LastName, DOB, Citizenship)
     textFile.open()
     textFile.append(record)
     textFile.close()
}

method display()
{
     textFile.open()
     While !textFile.EOF
          record=textFile.readLine()
          arrayList.add(record)

     table(arrayList)
     textFile.close()
}


method search()
{
     targetName=input()
     textFile.open()
     While !textFile.EOF
          record=textFile.readLine()
          arrayList.add(record)

     While arrayList.hasNext()
          record=arrayList.getNext()
          If record.name==targetName
               Display that record

     textFile.close()
}
```

```
method sort()
{
     While !textFile.EOF
          record=textFile.readLine()
          arrayList.add(record)
     arrayList.sort()

     textFile.close()
}

method generateCard()
{
     targetName=input()
     If targetName=""
          Print "select a name"
     Else
          message=messageArray[random]
          Create a PDF file
          Print "card is generated"
}
```

```
Method delete()
{
     targetName=input()
     textFile.open()
     While !textFile.EOF
          record=textFile.readLine()
          arrayList.add(record)

     While arrayList.hasNext()
          record=arrayList.getNext()
          If record.name==targetName
               arrayList.remove()

     textFile.clear()

     While arrayList.hasNext()
          record=arrayList.getNext()
          textFile.append(record)

     textFile.close()
}
```

Words: 80

Testing Strategies

Success Criteria	Testing Strategy	Expected Result
1.Display records in an organized layout	Press the “display” button and visually inspect the output in the table Search for a student and visually inspect the output in the table	Records will be displayed in the table, neatly organized, when displayed as well as when searched
2.Add a new student	Enter first and last names and, DOB, citizenship and press the “Add” button.	New Records will be displayed in the table, as well as will be added to the text file If first or last names are left blank, an error message will be displayed
3.Data Validation	Add a student without entering the first and last names Generate a birthday card without selecting a student	An error message will be displayed
4.Calculate the student’s age	Compare the student’s DOB with the computer's current date.	Age displayed will be calculated by subtracting student DOB from computer’s current date
5.Search for a student	Enter the student’s name and press the Search button.	If the name is found, then the entire record will be displayed. If no name is entered an error message will be displayed
6.Sort out records by first name	Press the Sort button	All records will be displayed in the table in alphabetical order
7.Delete a record	Left click on the student record to select, Right click and choose delete, to delete the record	Deleted record will no longer be displayed in the table, and will also be deleted from the text file
8.Generate cards as PDF for the chosen student	Select a student and press Generate Card button	If no student is selected an error message will be displayed. Otherwise a new PDF file will be created. Double click to open the card

9.Cards should have different messages	Select a student and press Generate Card button	Visually inspect all cards to make sure that cards are personalized, and the greeting messages are different.

Criterion D - Development

Person Class

```
public class Person implements Serializable {

    // Instance variables
    private String firstName;
    private String lastName;
    private String dateOfBirth;
    private String citizenship;

    // Constructor to initialize a Person object
    public Person(String firstName, String lastName, String dateOfBirth, String citizenship) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.dateOfBirth = dateOfBirth;
        this.citizenship = citizenship;
    }

    // Getter method for firstName
    public String getFirstName() {
        return firstName;
    }

    // Getter method for lastName
    public String getLastName() {
        return lastName;
    }

    // Getter method for dateOfBirth
    public String getDateOfBirth() {
        return dateOfBirth;
    }

    // Getter method for citizenship
    public String getCitizenship() {
        return citizenship;
    }
```

This class encapsulates the data related to a person, such as first name, last name, date of birth, and citizenship. This helps in managing and accessing person-related information in a structured and organized manner. It also ensures consistency in how person data is stored and manipulated throughout the application. This reduces the risk of data inconsistency and redundancy. The Person class can be reused across different parts of the application wherever person-related information is required. This promotes code reusability and reduces code duplication. If you need to add or modify person-related properties, you only need to update the Person class, rather than making changes in multiple places.
The testing strategy does not specifically test the class encapsulation.

saveToFile() method:

```
// Private method to save persons data from list to file
private void saveToFile() {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME))) {
        for (Person person : persons) {
            // Write person details separated by commas and newline
            writer.write(String.join(",", person.getFirstName(), person.getLastName(),
                                     person.getDateOfBirth(), person.getCitizenship()));
            writer.newLine();
        }
    } catch (IOException e) {
        e.printStackTrace(); // Print stack trace if file writing fails
    }
}
```

This method ensures that the data stored in the application's in-memory list of persons (persons) is saved to a file. This allows the data to be persisted between application runs, meaning that the data will not be lost when the application is closed and reopened. If the application grows and the method of data storage needs to be changed, having a dedicated method for saving data makes it easier to implement such changes.
The testing strategy is very effective as it allows the check if a new student record is successfully stored in a text file.

loadFromFile() method

```
// Private method to load persons data from file into the list
private void loadFromFile() {
    try (BufferedReader reader = new BufferedReader(new FileReader(FILE_NAME))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(","); // Split line by comma
            if (parts.length == 4) { // Ensure line has correct number of parts
                String firstName = parts[0];
                String lastName = parts[1];
                String dateOfBirth = parts[2];
                String citizenship = parts[3];
                persons.add(new Person(firstName, lastName, dateOfBirth, citizenship));
                // Create Person object and add to list
            }
        }
    } catch (IOException e) {
        e.printStackTrace(); // Print stack trace if file reading fails
    }
}
```

This method ensures that the data saved by the saveToFile() method is read back into the application when it starts. This allows the application to restore its state, making previously saved data available for use without requiring users to re-enter it. Each line is split into parts based on the comma delimiter, and a new Person object is created if the line contains the expected number of parts. This ensures that the data is correctly parsed and loaded into the list of persons.
The testing strategy is very effective as it allows the check if records are successfully read from a text file and subsequently displayed in the table.

searchPerson() method:

```
// Method to search for a person by first name or last name
public Person searchPerson(String name) {
    for (Person person : persons) {
        // Case-insensitive check if the name matches first name or last name
        if (person.getFirstName().equalsIgnoreCase(name) || person.getLastName().equalsIgnoreCase(name)) {
            return person; // Return the person if found
        }
    }
    return null; // Return null if person not found
}
```

This method allows users to quickly find specific records from potentially large datasets. Instead of manually browsing through the entire list of persons, users can specify search criteria to efficiently locate the desired information. By using equalsIgnoreCase(), the method ensures that the search is case-insensitive, making it more user-friendly and accommodating different input styles.
The testing strategy is very effective as it allows the check if records are found if searched by first or last names.

ArrayList:

```
public class PersonDatabase {
    // File name where persons data is stored
    private static final String FILE_NAME = "persons.txt";
    private List<Person> persons; // List to store Person objects

    // Constructor to initialize the database and load data from file
    public PersonDatabase() {
        persons = new ArrayList<>(); // Initialize the list of persons
        loadFromFile(); // Load persons data from file into the list
    }
```

This data structure automatically resizes itself as elements are added or removed, eliminating the need to manually manage the size of the collection. This dynamic resizing is crucial for this application where the number of Person objects can vary significantly.
ArrayList provides constant-time complexity (O(1)) for accessing elements by index. This is beneficial for this application which requires frequent access to individual Person objects, such as retrieving or updating a specific record.
The testing strategy is very effective as it allows the check if records are displayed, searched, deleted, added, as ArrayList structure is used for all these operations.

calculateAge():

```
// Method to calculate age based on date of birth
private int calculateAge(String dob) {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    LocalDate birthDate = LocalDate.parse(dob, formatter);
    LocalDate currentDate = LocalDate.now();
    return Period.between(birthDate, currentDate).getYears();
}
```

This method creates a DateTimeFormatter object configured to parse dates in the "yyyy-MM-dd" format. It then uses this formatter to parse the DOB string into a LocalDate object representing the person's date of birth (birthDate). I used Period class to compute the difference between birthDate and currentDate in years using Period.between().

The getYears() method extracts the number of complete years in the period between the two dates. This method leverages Java's java.time package (LocalDate, Period) which provides robust support for date and time operations. This ensures accurate computation of age, including leap years and different month lengths.
The testing strategy is very effective as it allows the check if the student's age is correctly calculated.

sortPerson()

```
private void sortPersons() {
    List<Person> persons = database.getPersons(); // Retrieve all persons from database

    // Sort persons list alphabetically by First Name
    Collections.sort(persons, new Comparator<Person>() {
        @Override
        public int compare(Person p1, Person p2) {
            return p1.getFirstName().compareTo(p2.getFirstName());
        }
    });

    // Clear the table before adding sorted data
    tableModel.setRowCount(0);

    // Add sorted data to the table model
    for (Person person : persons) {
        tableModel.addRow(new Object[]{person.getFirstName(), person.getLastName(), person.getDateOfBirth(),
         person.getCitizenship(), calculateAge(person.getDateOfBirth())});
    }
}
```

Sorting the list of Person objects improves data organization, making it easier for users to find and access specific records. For example, sorting by name alphabetically allows for quicker visual scanning and retrieval of a particular person's details. I used the compareTo() method to compare the names of Person objects p1 and p2 lexicographically. The compareTo method returns a negative integer, zero, or a positive integer if the first string is less than, equal to, or greater than the second string.
The testing strategy is very effective as it allows the check if records can be sorted in alphabetical order.

generateCard():

```
PDDocument document = new PDDocument(); // Create a new PDF document
PDPage page = new PDPage(PDRectangle.A4); // Create a new page (A4 size)
document.addPage(page); // Add page to document

// Load custom TTF font
InputStream fontStream = getClass().getResourceAsStream(FONT_FILE); // Load font file
if (fontStream == null) {
    throw new IOException("Font file not found: " + FONT_FILE); // Throw exception if font file not found
}
PDType0Font font = PDType0Font.load(document, fontStream); // Load font

PDPageContentStream contentStream = new PDPageContentStream(document, page); // Create content stream for drawing

// Draw red border around card
PDColor red = new PDColor(new float[]{1, 0, 0}, PDDeviceRGB.INSTANCE); // Define red color
contentStream.setStrokingColor(red); // Set stroking color (border color)
contentStream.setLineWidth(5); // Set border line width
contentStream.addRect(50, 600, 500, 200); // Draw rectangle (x, y, width, height)
contentStream.stroke(); // Stroke (draw) the rectangle

// Set font and font size for text
contentStream.setFont(font, 20); // Set font and size

// Draw birthday message
contentStream.beginText(); // Begin text mode for drawing text
contentStream.newLineAtOffset(100, 700); // Set text position (x, y)
contentStream.showText("Happy Birthday to " + selectedName + "!"); // Display birthday message
contentStream.endText(); // End text mode
```

To generate a PDF document I used Apache PDFBox (PDDocument). First it loads a custom TrueType font (Arial Bold.ttf) for the birthday card, which could be changed as needed. It draws a red border around the card, sets font and font size, and writes birthday messages using beginText(), showText(), and endText() methods. It saves the generated PDF with a filename based on the selected person's name and displays a success message or an error message if any exception occurs during the process.
The testing strategy is very effective as it allows the check if PDF files are successfully generated.

Mouse listener for right click option

```
// Mouse listener for table (right-click to delete)
table.addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent me) {
        if (SwingUtilities.isRightMouseButton(me)) {
            int row = table.rowAtPoint(me.getPoint());
            if (row >= 0 && row < table.getRowCount()) {
                table.setRowSelectionInterval(row, row);
                JPopupMenu popup = new JPopupMenu();
                JMenuItem deleteItem = new JMenuItem("Delete");
                deleteItem.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent ev) {
                        int selectedRow = table.getSelectedRow();
                        if (selectedRow != -1) {
                            String firstName = (String) tableModel.getValueAt(selectedRow, 0);
                            String lastName = (String) tableModel.getValueAt(selectedRow, 1);
                            Person person = database.searchPerson(firstName);
                            if (person != null && person.getLastName().equals(lastName)) {
                                database.deletePerson(person);
                                tableModel.removeRow(selectedRow);
                            }
                        }
                    }
                });
                popup.add(deleteItem);
                popup.show(me.getComponent(), me.getX(), me.getY());
            }
        }
    }
```

I implemented right-click context menus that are intuitive for actions like deletion, providing users with a familiar and efficient way to manage data. I added MouseListener to a JTable, allowing the user to delete a record by right-clicking on a table row. It checks if the right mouse button is pressed (SwingUtilities.isRightMouseButton(me)) and finds the row where the mouse click occurred using table.rowAtPoint(me.getPoint()). JPopupMenu appears when the user right-clicks, which contains an item labeled "Delete" to the popup menu.
The testing strategy is very effective as it allows the check if right click displays a popup menu, and a record is deleted the delete option is chosen.

Aggregation:

```
public class PersonDatabase {
    // File name where persons data is stored
    private static final String FILE_NAME = "persons.txt";
    private List<Person> persons; // List to store Person objects
```

PersonDatabase class has a reference to another class (Person) as an attribute. PersonDatabase is designed to manage a collection of Person objects. By aggregating Person objects, it encapsulates the functionality to add, retrieve, search, and delete

persons from its internal list (List<Person>). PersonDatabase can easily accommodate future enhancements or changes to Person management logic without impacting other parts of the application. For instance, additional methods or attributes can be added to Person or PersonDatabase independently.
The testing strategy does not specifically test the relationships between classes, which is a fundamental concept of OOP.

displayPersons():

```
// Method to display all persons in the database in the table
private void displayPersons() {
    List<Person> persons = database.getPersons(); // Retrieve all persons from database
    tableModel.setRowCount(0); // Clear existing table rows
    for (Person person : persons) { // Iterate through each person and add to table model
        tableModel.addRow(new Object[]{person.getFirstName(), person.getLastName(),
         person.getDateOfBirth(), person.getCitizenship(), calculateAge(person.getDateOfBirth())});
    }
}
```

This method is designed to update a table with the latest data retrieved. It clears all existing rows in the table model by setting the row count to zero. This is crucial to avoid displaying outdated or duplicate information. The loop iterates over each Person object in the list, ensuring that every person retrieved is processed.
The testing strategy is very effective as it allows the check if data is successfully read from the file and displayed in a table, as well as to check if the content is updated when a record is deleted.

Words: 1070

Criterion E - Evaluation

Meeting Success Criteria

1.Display records in an organized layout
Achieved - all records are neatly displayed in a table. Column width can be easily adjusted. The same table is also used to display found records when the search function is performed.

2.Add a new student
Achieved - a record is successfully added to the text file, and the table automatically updates and displays existing records. A new record can not be added if the first or last names are left blank. Citizenship and DOB can be chosen from the combobox, which eliminates any errors while typing.

3.Data Validation
Achieved - This is used when a new record without a name is attempting to be added, or when a new birthday card is being generated without having selected the student.

4.Calculate the student's age
Achieved - The student age is correctly calculated using the computer's local date/time and the result is displayed automatically as soon as the record is added. This feature works correctly in any time zone.

5.Search for a student
Achieved - Students can be searched by first or last names. If there are multiple students with the same first or last names, all matching records are displayed. If no name is found, no record is displayed.

6.Sort out records by first name
Achieved - All records are displayed in alphabetical order of their first names.

7.Delete a record
Achieved - When a student is selected and deleted, it removes the record from the text file and updates the content of the table.

8.Generate cards as PDF for the chosen student
Achieved - a birthday card is created as PDF for the selected student. The content of the card is personalized, and the file name contains the name of the student.

9.Cards should have different messages
Achieved - birthday cards have different messages which are randomly picked from the available messages.

All success criteria are achieved.

Suggesting improvements
Currently, all records are permanently stored in a text file. Using a database like MySQL to store the records would be much more efficient. This change would also enable the creation of a web-based application, allowing authorized individuals to access the product easily.

Another improvement would be to implement different access levels for various users, such as teachers and class captains. Teachers would have full permissions to modify data, while students would only have read-only access. This setup would prevent students from unintentionally modifying records.

Additionally, it would be beneficial to highlight students whose birthdays are within the next 2-3 days as a notification feature.

To make this product suitable for school-wide use, a field representing grade level or section would need to be created.

Words: 390