# Address Book Library Design Document

Xinyi Liu

## 1. Introduction

### 1.1. Purpose

This library is supposed to provide behaviors like a real hand-written address book, which can record name, postal address, phone number, email address, note, etc. This library can also be applied to scenarios like storing mailing addresses on online shopping websites.

### 1.2. Scope

- Goals
    - Create an empty address book.
    - Add an entry consisting of a name, postal address, phone number, email address, and a note.
    - Remove an entry.
    - Search entries.
    - Save the address book to a file.
    - Read the address book from a file.
- Non-Goals
    - Provide validation checks for every field.

## 2. Design

### 2.1. Structure

This library contains a data model, Contact, that serves as each data entry in the address book and a class, AddressBook, that provides functions as an address book.

### 2.2. Data

- Contact

  This class contains the following fields and all are made private and immutable.

    - Name: String
    - PostalAddress: String
    - PhoneNumber: String
    - EmailAddress: String

- Note: String

This class also includes the following public methods. There is no public constructor exposed to the users.

- Builder: None of the fields are required. However, if all Name, PostalAddress, PhoneNumber and EmailAddress are null or blank, or the EmailAddress doesn't follow the specific email format, then an error will be thrown and the instance won't be created.
- Getter: Return the value of each private field as a way to expose the contents to the users.

The equals method should be overwritten as two null fields should also be treated as equal. The hashCode method should be modified accordingly.

**Pros:**

1. This design provides an easy approach to create an instance with very basic format checking for EmailAddress.
2. This design prevents fields in instances from being modified after being created.

**Cons:**

1. The validation checks for fields other than EmailAddress are hard to perform. Since there is no clear rule that satisfy all different use cases, this library just keep it simple and trust users' inputs.
2. When updating a Contact entry in AddressBook, users need to create a new Contact, add the new Contact, and remove the old one. However, the updating is not supposed to happen very often as it is not required as an API in this library. This procedure is affordable.

- AddressBook

  This class contains the following private and immutable field.

  - Contacts: List<Contact>

  This class also includes a public constructor that initialize the list as a LinkedList. A map structure is not appropriate here because a unique key cannot be guaranteed. All fields in Contact are optional.

  Other methods provided in this class are documented in the following section.

## 2.3. APIs

- Add

    - Input: Contact contact
    - Output: -
    - Exception: NullPointerException

  This function will call List.Add to Contacts if the input is not null.

- Remove

    - Input: Contact contact
    - Output: Boolean status
    - Exception: NullPointerException

  This function will call List.remove from Contacts if the input is not null. The first element that the overridden equals method of Contacts returns true when comparing with the input will be deleted. This method will return true if remove exactly one entry in Contacts, false if no equal entry found.

- Search

    - Input: String name, String postalAddress, String phoneNumber, String emailAddress, String note
    - Output: List<Contact> results

  This function will take several strings and search each field contains the corresponding string. If the corresponding input string is null, then this field will be ignored. The method will return a list Contact each of which satisfy all the input search criteria. This can be done by filter and keep Contact when (input == null || input.equals(contact.getInputField())) for each field.

  This method only provides AND logic among the input search keywords. This is because OR logic can be achieved with this function. One may call this method multiple times, each with one field filled and others as null, and make a set of all the results.

- Save

    - Input: String path
    - Output: -
    - Exception: NullPointerException, IOException

  This function will save the Contacts list to a Json file with Gson, a Java serialization/deserialization library to convert between Java objects and JSON developed by Google. If the input path is a null string, then a NullPointerException will be thrown. If the file cannot be created or opened for the input path, an IOException will be thrown. The file will be overwritten if it already exists.

- Load

- Input: String path
- Output: -
- Exception: NullPointerException, IOException

This function will load the a list of Contact entries from a Json file with Gson and add them to the Contacts list. If the input path is a null string, then a NullPointerException will be thrown. If the file cannot be read correctly, an IOException will be thrown.

# 3. Alternative Designs

## 3.1. Make fields in Contact changeable

**Pros:**

1. It's easier for updating information in Contact.

**Cons:**

1. Users may accidentally change the data stored in AddressBook. However, if return a copy of instance after searching, then there is no way to delete the original one once user updates the copy.
2. Even if a private UUID field is added and served as the only evidence for equality to enable deletion, it will lead to confusion when users operate. They have no idea of this UUID field, so the deletion behavior may not be as expected.

Since the updating is not supposed to happen very often, the fields in Contact is made immutable to keep data safe.

## 3.2. Use JSON-JAVA to parse data

**Pros:**

1. The serialization and deserialization methods are the most general among all JSON parser options in Java.

**Cons:**

1. This library requires to create JSONObject and JSONArray and put every Contact in Contacts from AddressBook to these JSON containers when writing. It also requires to read things field by field when reading.

Compared to JSON-JAVA library, Gson can also handle this defined data model. It's easier to write with Gson.ToJson which can directly take the Contacts list as well as read with Gson.FromJson which can convert the JSON file to a list of Contacts if TypeToken<List<Contact>> is defined.