ASSIGNMENT PROJECT REPORT

Software Requirements Specification

COMP3211 Software Engineering
Department of Computing
The Hong Kong Polytechnic University

Koon Him MOO
WU Jiasheng
Ho Yu YIP
Sarah Bless Sarmiento Monsod

November 25, 2023

CONTENTS

1 Preface 3

2 Introduction 4

3 Glossary 5

4 System Requirements Definition 6-7

5 System Architecture 8-9

6 System Requirements Specification 10-15

7 System Relationship Models 16-18

Appendices

1 Preface

Expected Relationship: PIM Users, Software Managers, Investors
Version: 1.0 beta
Updates: N/A
Change Rationale: N/A

2 Introduction

2.1 Purpose

To develop a command-line-based personal information manager (PIM).

2.2 Scope

- Provide users an efficient platform to use the PIM, user can modify PIRs, search or print the information in PIRs, import or export PIM file in PIM.
- The users shall be able to manage PIRs in a single location, do any action in a single platform.
- The PIM fulfill the user requirements mention in user stories for the PIM.

2.3 Overview

This Software Requirements Specification is organized into five parts:

- The third section will define the technical terms used in the document to help users understand the specification.
- The fourth section will describe the user requirements according to the user stories for the PIM.
- The fifth section will provide an overview of the system architecture from a high level through illustrating system module components.
- The sixth section will provide a more detailed description of functional and non-functional requirements from a system's perspective.
- The seventh section will provide the system relationship model, a UML class diagram for show both the structure of and the relationship among the major code components in the PIM, a UML sequence diagram for outline the process of an example use of the PIM.

3 Glossary

This section mainly defines the technical terminology in this document and corresponds its abbreviation to its definition to help the users can better understand the document.

Abbreviation	Meaning
CLI	Command Line Interface
GUI	Graphical User Interface
JVM	Java Virtual Machine
MVC	Model-View-Controller
NFR	Non-Functional Requirement
OOP	Object-Oriented Programming
PIM	Personal Information Manager
PIR	Personal Information Record
REQ	Requirement
SR	System Requirement
TNFR	Types of Non-Functional Requirements
UR	User Requirement

Table 1: Abbreviation Table

4 User Requirements Definition

4.1 User Requirements

- UR-01: The user can create different types of PIRs in the PIM so that all the information the user cares about can be managed in a single location.
- UR-02: The user can create new plain texts as PIRs so that the user can use the PIM to take quick notes.
- UR-03: The user can create new tasks with the corresponding descriptions and deadlines as PIRs so that the user can use the PIM to manage to-dos.
- UR-04: The user can create new events with the corresponding descriptions, starting times, and alarms as PIRs so that the user can use the PIM to manage the schedule.
- UR-05: The user can create new contacts with the corresponding names, addresses, and mobile numbers as PIRs so that the user can use the PIM to manage contacts.
- UR-06: The user can modify the data in existing PIRs so that the user can keep the PIRs up to date.
- UR-07: The user can search for PIRs based on criteria concerning their types and the data stored in their fields.
- UR-08: The user can print out detailed information about a specific PIR or all PIRs.
- UR-09: The user can delete a specified PIR.
- UR-10: The user can store the PIRs in a file so that the user can access them using the PIM in the future.
- UR-11: The user can load the PIRs from a file with the extension name ".pim" so that the user can continue working with the PIRs stored earlier.

4.2 User Non-functional Requirements

- UR-12: The PIM runs as a command-line.
- UR-13: When the user search for PIRs, PIM need to have a criterion to check whether a piece of text (stored in a note, a description, a name, an address, or a mobile number) contains a string, whether a time (stored in a deadline, a starting

time, or an alarm) is before (<), after (>), or equal to (=) another given point in time, or whether a condition combining multiple other conditions via logical connectors and (&&), or (||), and negation (!) is satisfied.

- UR-14: When the user store the files, PIM should automatically add the extension name ".pim" for them.

5 System Architecture

5.1 Architectural Patterns

We have implemented the MVC architectural pattern for our PIM. Here's an explanation of the MVC pattern in the context of our PIM:

Model:
The Model component represents the data and business logic of the application. It consists of the following classes located in the pim.Model package:

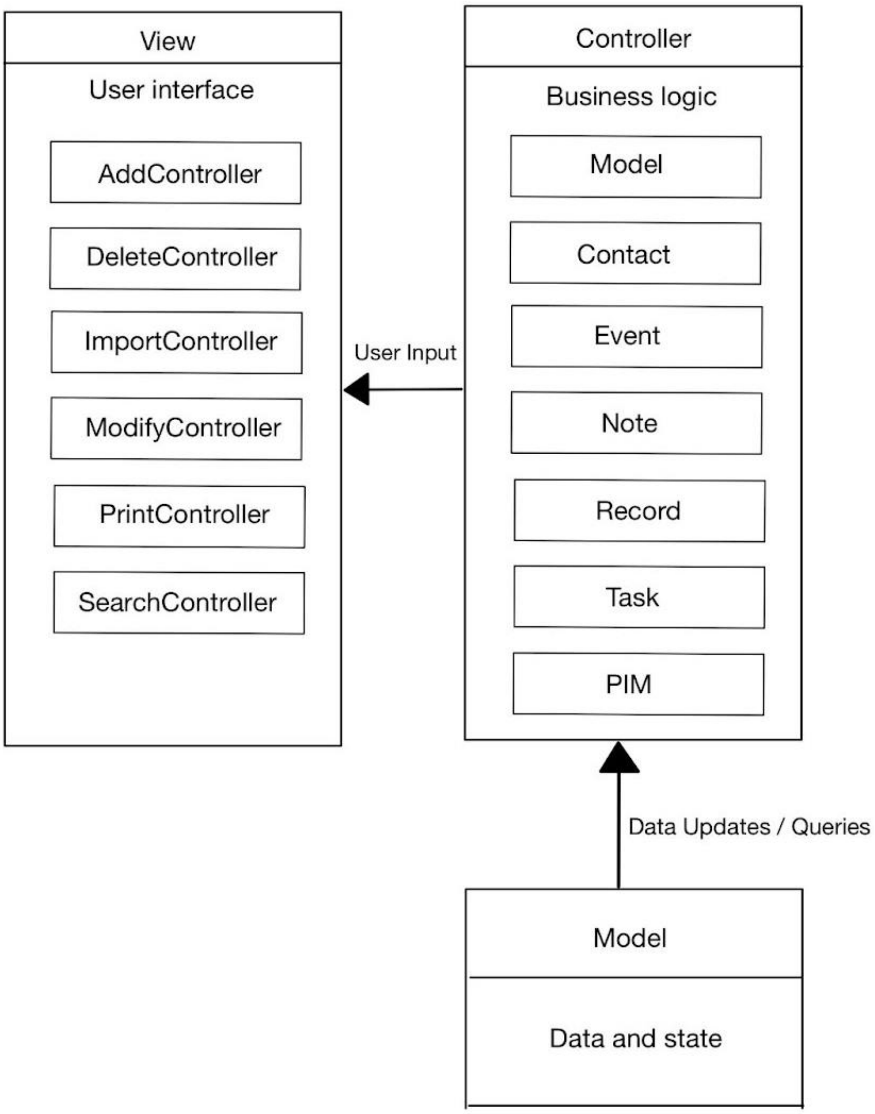
- Contact.java: Represents a contact in the PIM.
- Event.java: Represents an event in the PIM.
- Note.java: Represents a note in the PIM.
- Record.java: Represents a record in the PIM.
- Task.java: Represents a task in the PIM.
- PIM.java: Acts as the central model class that interacts with the data and provides methods for manipulating the PIM's entities.

View:
The View component represents the user interface of the application. It consists of the following controller classes located in the pim.Controller package:

- AddController.java: Handles the logic for adding new items to the PIM.
- DeleteController.java: Handles the logic for deleting items from the PIM.
- ImportController.java: Handles the logic for importing data into the PIM.
- ModifyController.java: Handles the logic for modifying existing items in the PIM.
- PrintController.java: Handles the logic for printing/displaying items from the PIM.
- SearchController.java: Handles the logic for searching/filtering items in the PIM.
- Controller: execute interface

The Controller component acts as the intermediary between the Model and the View. It handles user input, performs necessary operations on the data, and updates the View accordingly. The controller classes mentioned above are responsible for controlling the flow of information between the Model and the View based on user actions.
By implementing the MVC pattern, you separate the concerns of data manipulation (Model), user interface (View), and user interaction/control flow (Controller). This separation allows for better maintainability, reusability, and testability of your codebase.

ARCHITECTURE DIAGRAM:



In this diagram, the components are organized according to the project's structure. The View component represents the user interface and contains individual controllers (AddController, DeleteController, ImportController, ModifyController, PrintController, SearchController) responsible for handling user input and interacting with the Controller component.

The Controller component serves as the bridge between the View and the Model. It contains the business logic and communicates with the Model to update or query data.

The Model component represents the data and state of the application. It consists of various model classes (Contact, Event, Note, Record, Task, PIM) that define the structure and behavior of the PIM's data.

6 System Requirements Specification

This section describes functional and non-functional requirements in more details and futher details will added to the non-functional requirements.

6.1 System Functional Requirements

ID	SR-01
Title	PIM Panel
Requirement	The PIM should contain a panel which allow the user to modify multiple types of personal information records, which include the function of creating, editing, and deleting records.
Rationale	Gives a better understanding for the user to distinguish which parts do which actions, can let the user to manage in a single location.
Reference(SR,UR)	UR-01
Priority	1

ID	SR-02
Title	A menu for creating PIRs in PIM
Requirement	The PIM should contain a menu which allow the user to choose whether they want to create plain texts, tasks, events or contacts.
Rationale	Gives a better understanding for the user to choose which PIRs he or she want to create.
Reference(SR,UR)	UR-01, UR-02, UR-03, UR-04, UR-05
Priority	1

ID	SR-03
Title	Record
Requirement	The PIM should contain a record which is used to keep the data in the data item in a consistent state, that after user

	modify or search PIRs, or import or export files, record is keep update and save in the system.
Rationale	For keeping records
Reference(SR,UR)	UR-01 to UR-11
Priority	1

ID	SR-04
Title	Create plain texts
Requirement	The PIM should allow the user to create new plain texts as personal information records. By creating the plain tasks, the user can type the content for quick notes, which the system receives what user type and save it in record.
Rationale	Satisfy the user requirements for making quick notes.
Reference(SR,UR)	UR-01, UR-02
Priority	-

ID	SR-05
Title	Create new tasks
Requirement	The PIM should allow the user to create new tasks. By creating new tasks, the user can type and save the content for description and set up the date for the deadline, which the system receives what user type and save it in record.
Rationale	Satisfy the user requirements for managing to-dos.
Reference(SR,UR)	UR-01, UR-03
Priority	-

ID	SR-06
Title	Create new events
Requirement	The PIM should allow the user to create new events. By creating new events, the user can type and save the content for descriptions, set up the date and time for starting time,

	which the system receives what user type and save it in record.
Rationale	Satisfy the user requirements for managing schedule.
Reference(SR,UR)	UR-01, UR-04
Priority	-

ID	SR-07
Title	Create new contacts
Requirement	The PIM should allow the user to create new contacts. By creating new contacts, the user can type and save the content for the corresponding names, addresses, and mobile numbers, which the system receives what user type and save it in record.
Rationale	Satisfy the user requirements for managing contacts.
Reference(SR,UR)	UR-01, UR-05
Priority	-

ID	SR-08
Title	Modify the data in existing PIRs
Requirement	The PIM should allow the user to modify the data in existing PIRs that the user created, which let the user can choose the existing PIRs to modify according to each type's requirements.
Rationale	Satisfy the user requirements for keeping the PIRs up to date.
Reference(SR,UR)	UR-06
Priority	-

ID	SR-09
Title	Search for PIRs

Requirement	The PIM should allow the user to search for PIRs based on criteria concerning their types and the data stored in their fields, which let the user to choose the type of PIRs. After that, the system should identify the criteria and keyword that the user type, search the data that contain the keyword in record and match the criteria, and display the result to the user.
Rationale	Satisfy the user requirements for searching for PIRs based on criteria concerning their types and the data stored in their fields.
Reference(SR,UR)	UR-07, UR-13, SR-03
Priority	-

ID	SR-10
Title	Printing
Requirement	The PIM should allow the user to printing out detailed information about a specific PIR or all PIRs, which the system will show all the existing PIRs by using index to identify them, let the user to choose which PIR want to print out all the information in specific PIR or all PIRs by typing the index, and the system should identify the index, and print out the required information in PIRs in record.
Rationale	Satisfy the user requirements for printing out detailed information about a specific PIR or all PIRs.
Reference(SR,UR)	UR-08
Priority	-

ID	SR-11
Title	Delete PIRs
Requirement	The PIM should allow the user to delete the specified PIR, which the system will show all the existing PIRs by using index to identify them, let the user to choose which PIR want to delete PIR by typing the index, and the system should identify the index, and delete required PIRs in record.

Rationale	Satisfy the user requirements for deleting a specified PIR.
Reference(SR,UR)	UR-09
Priority	-

ID	SR-12
Title	store PIRs
Requirement	The PIM should allow the user to store the PIRs in a file, which the system will require the user to enter new file name for the file by typing, and the system should combine the user type and ".pim" for storing the file.
Rationale	Satisfy the user requirements for accessing the stored PIRs file by using the PIM in the future.
Reference(SR,UR)	UR-10, UR-14
Priority	-

ID	SR-13
Title	load PIRs
Requirement	The PIM should allow the user to load the PIRs in a file with the extension name ".pim", which the system will require the user to enter new file name for the file plus ".pim" by typing, and the system should load the required file to the PIM and load all the record in that file, and save to the record in PIM.
Rationale	Satisfy the user requirements for continuing working with the PIRs stored earlier.
Reference(SR,UR)	UR-11, SR-03
Priority	-

6.2 System Non-functional Requirements

ID	SR-14
Title	Check user enter format
Requirement	Check the user input is it meet the requirement of capital letter or small letter, date is "xx/xx/xxxx", time is "xx:xx".

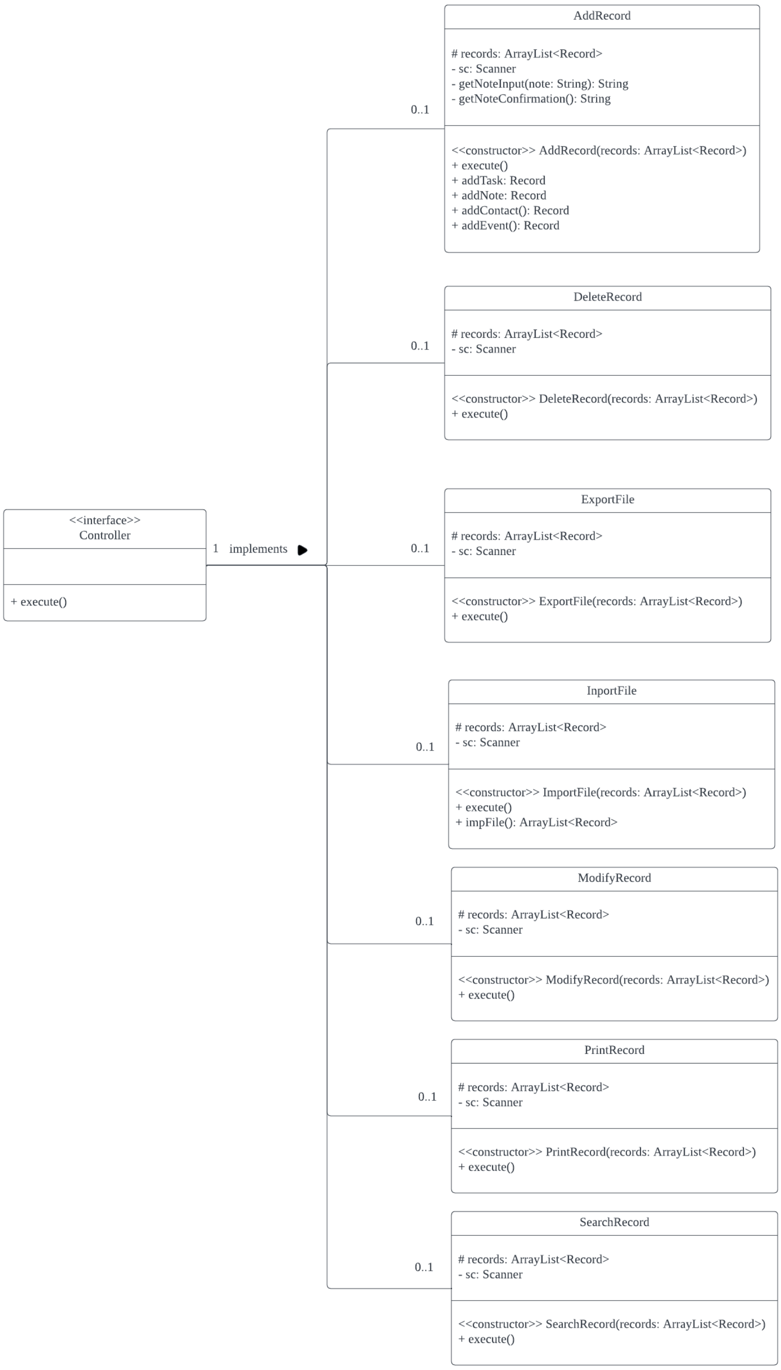
Rationale	check the correct input
Reference(SR,UR)	-
Priority	1

ID	SR-15
Title	Criterion for searching
Requirement	When the user searching, the system should check the keyword is it contain a string for text stored in the existing note, for a description stored in the existing task or in the existing events, or for a name, an address, or a mobile number stored in existing contacts. Also, the system should check the keyword is it contain before "<", after ">", or equal to "=" another given point in time, or whether a condition combining multiple other conditions via logical connectors and (&&), or (||), and negation (!) is satisfied for the deadline stored in existing task, or for the starting time, or the alarm stored in the existing event.
Rationale	Check the user's keyword meet criterion for searching or not.
Reference(SR,UR)	UR-07, UR-13

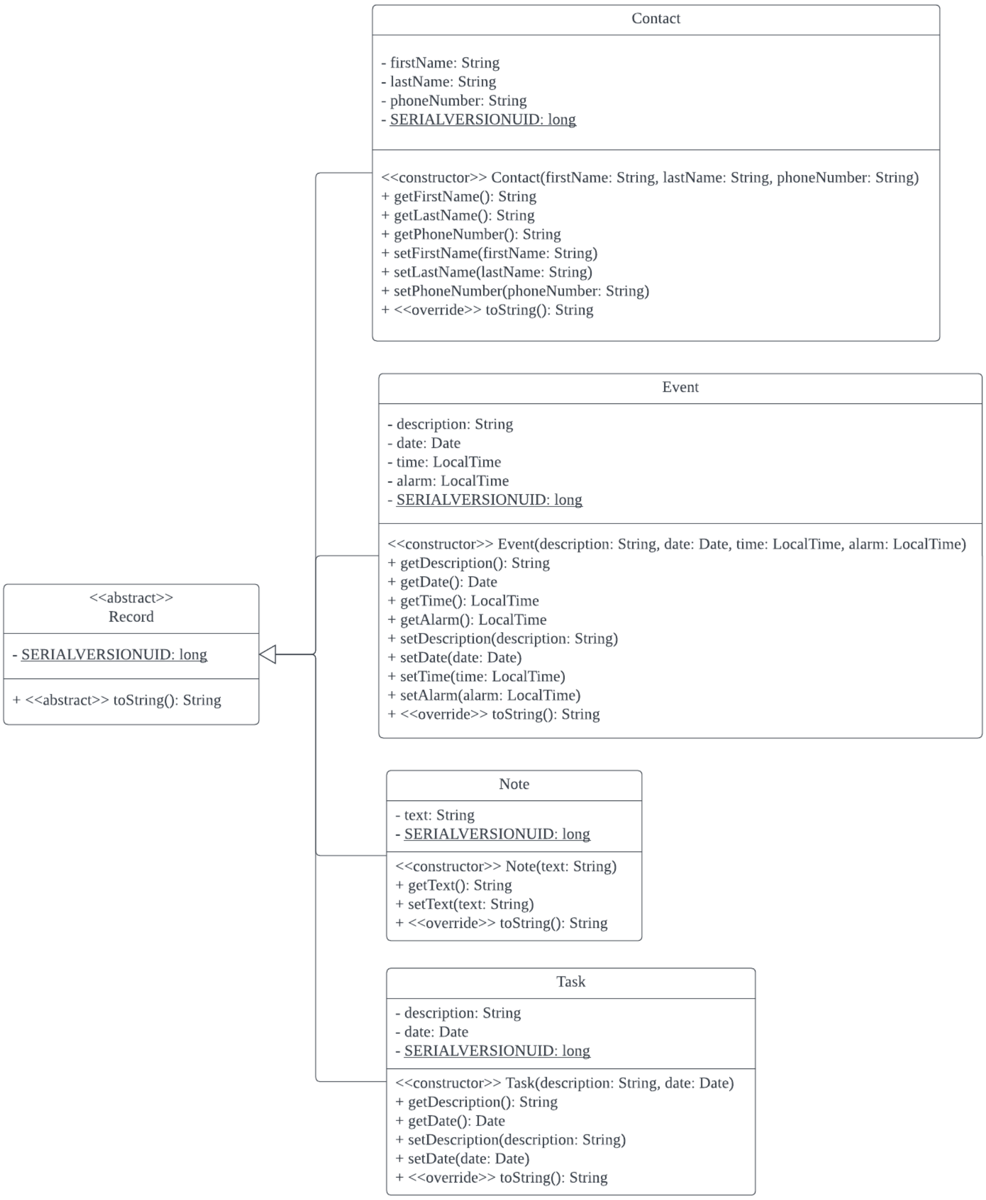
ID	SR-16
Title	Automatically add the extension name ".pim" for storing
Requirement	When user store the file, the system should automatically combine the user keyword and the extension name ".pim" to prevent user forget add ".pim" when saving the file.
Rationale	Prevent user forget add ".pim" when saving the file.
Reference(SR,UR)	SR-12
Priority	-

7 System Relationship Model

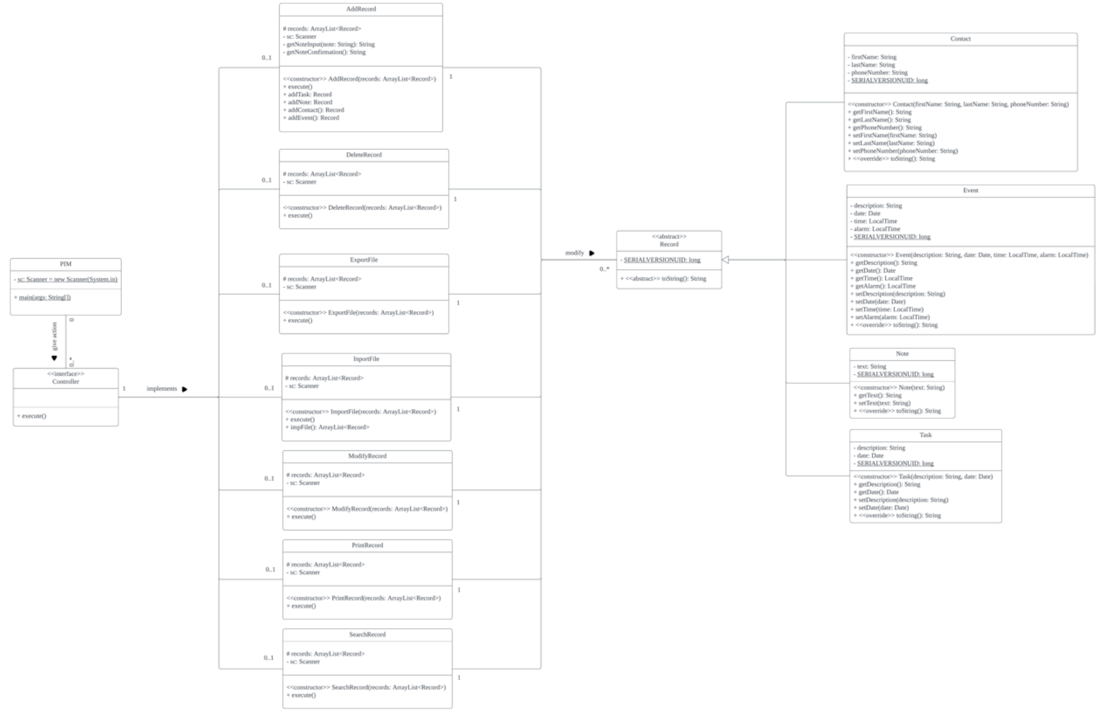
7.1 UML Class Diagram


AddRecord
# records: ArrayList<Record>
- sc: Scanner
- getNoteInput(note: String): String
- getNoteConfirmation(): String
<<constructor>> AddRecord(records: ArrayList<Record>)
+ execute()
+ addTask: Record
+ addNote: Record
+ addContact(): Record
+ addEvent(): Record
0..1
DeleteRecord
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> DeleteRecord(records: ArrayList<Record>)
+ execute()
0..1
<<interface>>
Controller
+ execute()
1 implements
ExportFile
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> ExportFile(records: ArrayList<Record>)
+ execute()
0..1
InportFile
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> ImportFile(records: ArrayList<Record>)
+ execute()
+ impFile(): ArrayList<Record>
0..1
ModifyRecord
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> ModifyRecord(records: ArrayList<Record>)
+ execute()
0..1
PrintRecord
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> PrintRecord(records: ArrayList<Record>)
+ execute()
0..1
SearchRecord
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> SearchRecord(records: ArrayList<Record>)
+ execute()
0..1

Class “Controller” is to help user to implment the function of AddRecord, DeleteRecord, ExportFile, ImportFile, ModifyRecord, PrintRecord and SearchRecord.

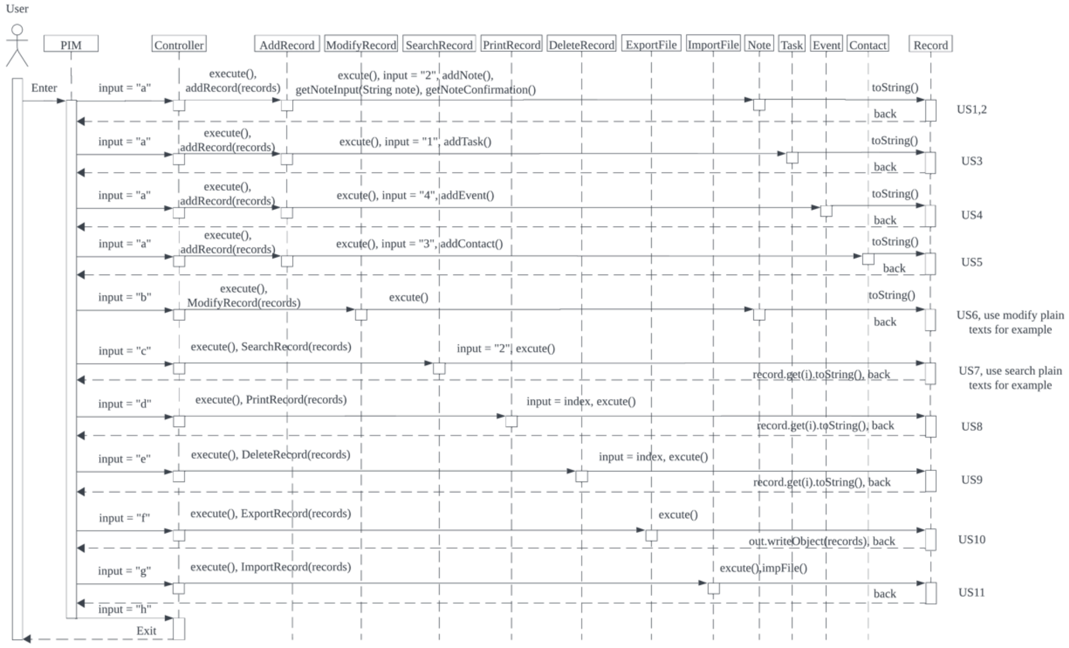

Contact
- firstName: String
- lastName: String
- phoneNumber: String
- SERIALVERSIONUID: long
<<constructor>> Contact(firstName: String, lastName: String, phoneNumber: String)
+ getFirstName(): String
+ getLastName(): String
+ getPhoneNumber(): String
+ setFirstName(firstName: String)
+ setLastName(lastName: String)
+ setPhoneNumber(phoneNumber: String)
+ <<override>> toString(): String
Event
- description: String
- date: Date
- time: LocalTime
- alarm: LocalTime
- SERIALVERSIONUID: long
<<constructor>> Event(description: String, date: Date, time: LocalTime, alarm: LocalTime)
+ getDescription(): String
+ getDate(): Date
+ getTime(): LocalTime
+ getAlarm(): LocalTime
+ setDescription(description: String)
+ setDate(date: Date)
+ setTime(time: LocalTime)
+ setAlarm(alarm: LocalTime)
+ <<override>> toString(): String
<<abstract>>
Record
- SERIALVERSIONUID: long
+ <<abstract>> toString(): String
Note
- text: String
- SERIALVERSIONUID: long
<<constructor>> Note(text: String)
+ getText(): String
+ setText(text: String)
+ <<override>> toString(): String
Task
- description: String
- date: Date
- SERIALVERSIONUID: long
<<constructor>> Task(description: String, date: Date)
+ getDescription(): String
+ getDate(): Date
+ setDescription(description: String)
+ setDate(date: Date)
+ <<override>> toString(): String

Class “Record” help user to serialize the data of Contact, Event, Note, Task.


AddRecord
# records: ArrayList<Record>
- sc: Scanner
- getNoteInput(note: String): String
- getNoteConfirmation(): String
<<constructor>> AddRecord(records: ArrayList<Record>)
+ execute()
+ addTask: Record
+ addNote: Record
+ addContact(): Record
+ addEvent(): Record
DeleteRecord
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> DeleteRecord(records: ArrayList<Record>)
+ execute()
ExportFile
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> ExportFile(records: ArrayList<Record>)
+ execute()
InportFile
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> ImportFile(records: ArrayList<Record>)
+ execute()
+ impFile(): ArrayList<Record>
ModifyRecord
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> ModifyRecord(records: ArrayList<Record>)
+ execute()
PrintRecord
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> PrintRecord(records: ArrayList<Record>)
+ execute()
SearchRecord
# records: ArrayList<Record>
- sc: Scanner
<<constructor>> SearchRecord(records: ArrayList<Record>)
+ execute()
0..1
1
PIM
- sc: Scanner = new Scanner(System.in)
+ main(args: String[])
give action
0
0..*
<<interface>>
Controller
+ execute()
1
implements
modify
0..*
<<abstract>>
Record
- SERIALVERSIONUID: long
+ <<abstract>> toString(): String
Contact
- firstName: String
- lastName: String
- phoneNumber: String
- SERIALVERSIONUID: long
<<constructor>> Contact(firstName: String, lastName: String, phoneNumber: String)
+ getFirstName(): String
+ getLastName(): String
+ getPhoneNumber(): String
+ setFirstName(firstName: String)
+ setLastName(lastName: String)
+ setPhoneNumber(phoneNumber: String)
+ <<override>> toString(): String
Event
- description: String
- date: Date
- time: LocalTime
- alarm: LocalTime
- SERIALVERSIONUID: long
<<constructor>> Event(description: String, date: Date, time: LocalTime, alarm: LocalTime)
+ getDescription(): String
+ getDate(): Date
+ getTime(): LocalTime
+ getAlarm(): LocalTime
+ setDescription(description: String)
+ setDate(date: Date)
+ setTime(time: LocalTime)
+ setAlarm(alarm: LocalTime)
+ <<override>> toString(): String
Note
- text: String
- SERIALVERSIONUID: long
<<constructor>> Note(text: String)
+ getText(): String
+ setText(text: String)
+ <<override>> toString(): String
Task
- description: String
- date: Date
- SERIALVERSIONUID: long
<<constructor>> Task(description: String, date: Date)
+ getDescription(): String
+ getDate(): Date
+ setDescription(description: String)
+ setDate(date: Date)
+ <<override>> toString(): String

A full view of class diagram, which show both the structure of and the relationship among the major code components in the
PIM

7.2 Sequence Diagram



A full view of sequence diagram of system reacts to user requirements, which outline the process of an example use of the PIM

Appendices

User Stories:

US1. As a user, I want to create different types of personal information records (PIRs) in the PIM so that all the information I care about can be managed in a single location.
US2. As a user, I want to create new plain texts as PIRs so that I can use the PIM to take quick notes.
US3. As a user, I want to create new tasks with the corresponding descriptions and deadlines as PIRs so that I can use the PIM to manage my to-dos.
US4. As a user, I want to create new events with the corresponding descriptions, starting times, and alarms as PIRs so that I can use the PIM to manage my schedule.
US5. As a user, I want to create new contacts with the corresponding names, addresses, and mobile numbers as PIRs so that I can use the PIM to manage my contacts.
US6. As a user, I want to modify the data in existing PIRs so that I can keep the PIRs up to date.
US7. As a user, I want to search for PIRs based on criteria concerning their types and the data stored in their fields. A criterion may check whether a piece of text (stored in a note, a description, a name, an address, or a mobile number) contains a string, whether a time (stored in a deadline, a starting time, or an alarm) is before (<), after (>), or equal to (=) another given point in time, or whether a condition combining multiple other conditions via logical connectors and (&&), or (||), and negation (!) is satisfied.
US8. As a user, I want to print out detailed information about a specific PIR or all PIRs.
US9. As a user, I want to delete a specified PIR.
US10. As a user, I want to store the PIRs in a file with the extension name ".pim" so that I can access them using the PIM in the future.
US11. As a user, I want to load the PIRs from a file with the extension name ".pim" so that I can continue working with the PIRs I stored earlier.