Term Project: *ChocAn*

Design Document

Written by Al Khatab Al Rashdi, Eleanore Clark, Michael Fulton, Ivan Lopez, Erin Rasmussen, and Tristan Smith

Table of Contents

1. Introduction **2**

2. Design Considerations **5**

2.1 Constraints and Dependencies 6

2.2 Methodology 6

3. System Overview **7**

4. System Architecture **9**

4.1 Architecture Overview 10

4.2 Main 10

4.2.1 ChocAnTerminal—Base class 11

4.2.2 ProviderTerminal 12

4.2.3 Manager Terminal 13

4.3 Provider 14

4.4 Member 15

4.5 ServiceRecord 16

4.6 Service 17

5.1 ChocAn Main **19**

5.1.1 Main Methods 19

5.2 ChocAnTerminal class **19**

5.2.1 ChocAnTerminal Fields 19
5.2.2 ChocAnTerminal Methods 20

5.3 ManagerTerminal class 21
5.3.1 ManagerTerminal methods 21

5.4 ProviderTerminal class 22
5.4.1 ProviderTerminal Fields 22
5.4.2 ProviderTerminal Methods 23

5.5 ChocAnList class 23
5.5.1 ChocAnList Methods 23

5.6 ServiceRecord class 24
5.6.1 ServiceRecord Fields 24
5.6.2 ServiceRecord Methods 24

5.7 Provider class 25
5.7.1 Provider Fields 25
5.7.2 Provider Methods 25

5.8 Member class 26
5.8.1 Member Fields 26
5.8.2 Member Methods 26

5.9 Service class 27
5.9.1 Service Fields 27
5.9.2 Service Methods 27

1. Introduction

The following document provides an overview of the design for the software product requested by Chocoholics Anonymous (ChocAn), the Chocoholics Data Processing Software (CDPS). This software will handle all data-processing requests made through any ChocAn-associated terminal. There are two types of terminals, labeled provider and manager, that both have different data-access permissions. The system will store this data in files; these can be used in the report-generation process for both managers and providers. These operations are explained in greater detail in the requirements document created for this same software.

Unlike the requirements document, this document does not aim to provide a detailed list of specifications, but instead outlines the system's design in greater detail. This document is broken down into five sections: Introduction, Design Considerations, System Overview, System Architecture, and Detailed System Design. Each section aims to discuss, in varying detail, either the design of the software itself or the design considerations for the software. This document also uses UML diagrams to show the architecture of the CDPS and outline some of the methods that will be used. The Detailed System Design section thoroughly outlines the ways in which classes and their methods from the UML diagrams shall be designed and implemented.

1.1 Purpose and Scope

This document is meant to provide a detailed outline of the design that shall be used in the development of the CDPS. The information is broken down into different levels, from high- to low-level design. These levels start at the System Overview section and end at the Detailed System Design. That said, this document does not include an entire list of functional and non-functional requirements, as this has already been outlined in the requirements document for the same software. This is to be used as a reference for the CDPS design and design considerations.

1.2 Target Audience

The target audience for this document is stakeholders of the software, and, most likely, their development team. However, anyone in need of CDPS design documentation can benefit from this document, especially in conjunction with the requirements document for the same software.

1.3 Terms and Definitions

The following are the terms and definitions of frequently used vocabulary in the following document.

- ChocAn:
    - Shorthand for Chocoholics Anonymous.
- CDPS:
    - Chocoholics Data Processing Software; the software requested by ChocAn.
- Constraint:
    - Something that limits the degree of freedom available in providing a solution.
- Dependency:
    - A type of relationship between activities that requires them to be performed in a particular order.
- EFT:
    - Electronic Funds Transfer.
- Manager:
    - An employee of ChocAn who works with the CDPS.
- Member:
    - A person seeking access to medical services through ChocAn.
- Provider:

    - A medical provider working with ChocAn to provide services to members.
- Software:
    - A computer program and its related documentation.
- Stakeholder:
    - A user of the software, including members, providers, and managers.

2. Design Considerations

This section will provide an overview of the constraints, dependencies, and methodology that will be used in designing the CDPS. The constraints will include the limitations that are required to design and implement the program, functional or non-functional. The dependencies will include any actions that need to be performed in a particular order. The methodology will describe the methods that will be used to design and implement the software.

2.1 Constraints and Dependencies

- The membership fee payment process will be the responsibility of Acme Accounting Services, a third-party organization.
- The data from a provider's terminal must be simulated by keyboard input. Data to be transmitted to a provider's terminal must appear on the screen.
- A manager's terminal must be simulated by the same keyboard and screen.
- The software will be implemented in Java.
- The Acme computer must update the relevant ChocAn Data Center membership records each evening at 9 P.M.
- Each provider and member report must be written to its own file. The name of the file should begin with the provider or member name, followed by the date the report was generated.
- The Provider Directory must be created as a file.
- The EFT data must include the following: the provider name, provider number, and the amount to be transferred.

2.2 Methodology

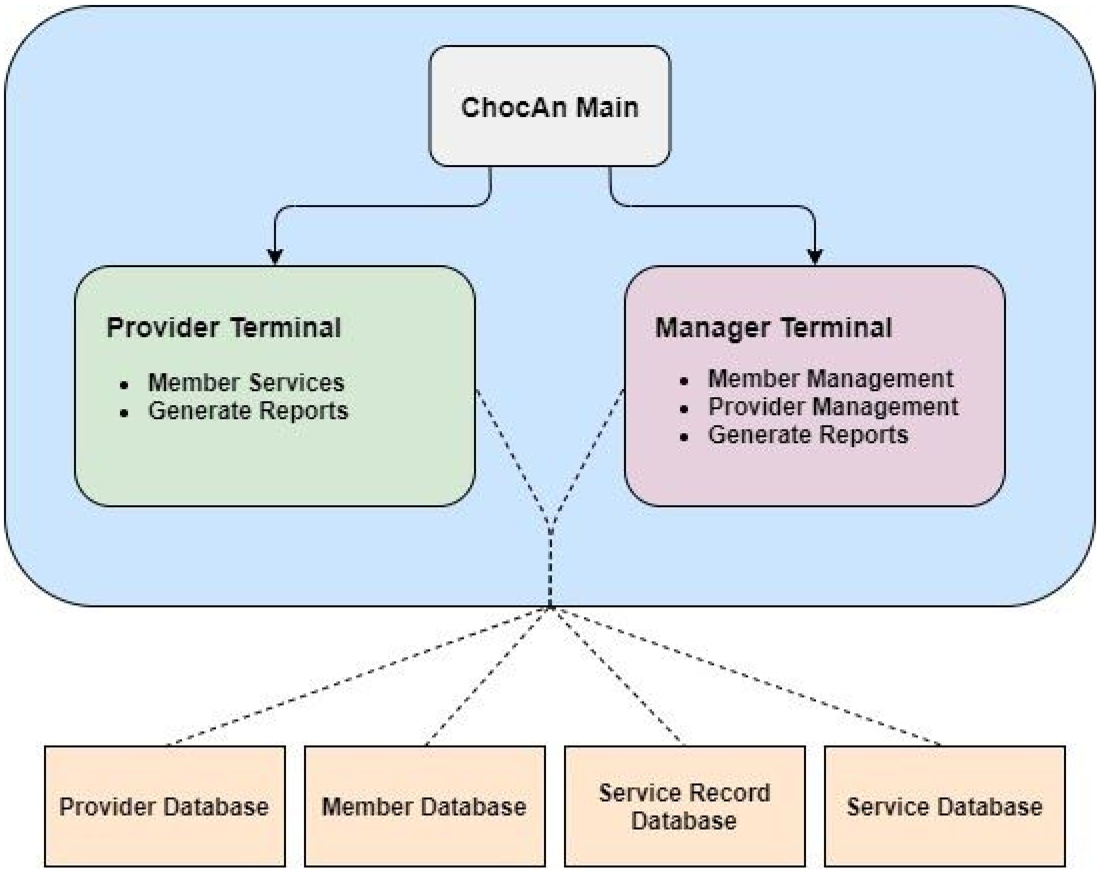
The production of the CDPS will follow the modified waterfall development methodology. As such, it will move through five stages: requirements specification, design, implementation, verification, and maintenance. This methodology provides a clear process that is easy to understand. It is also well-suited to small projects with short development times. It is the best option for the CDPS.

Modified waterfall methodology is very straightforward; it starts at the first stage (the requirements stage) and moves linearly until it reaches the last stage (the maintenance stage). At the time of writing this document, the requirements stage has been completed, and the results may be viewed in the requirements document for this same software. The second stage, design, is underway, and its culmination is the CDPS design document. It will provide the System Architecture using UML diagrams and provide details of the functions, classes, and data types that will be used to process the software data.

3. System Overview

The ChocAn system, also called the CDPS, is a data-processing software application commissioned by ChocAn to help people recovering from chocolate addiction. The CDPS provides tools for medical providers, working with ChocAn, to provide services for recovering chocolate-addict members covered by the ChocAn care plan. The CDPS provides the following features:

- The ability to choose between using the Provider and Manager Terminals.
- The ability to manage, create, and track information regarding service records, providers, members, and service databases.
- The creation of weekly, member, provider, EFT, and summary reports.
- The validation of providers and members using the CDPS.
- Tools for saving and loading information from files into databases at run-time.



The CDPS architecture consists of the ChocAn Main module, the Provider and Manager Terminal, and the different databases for the provider, service record, member, and provider. What follows is a general overview of each of these modules:

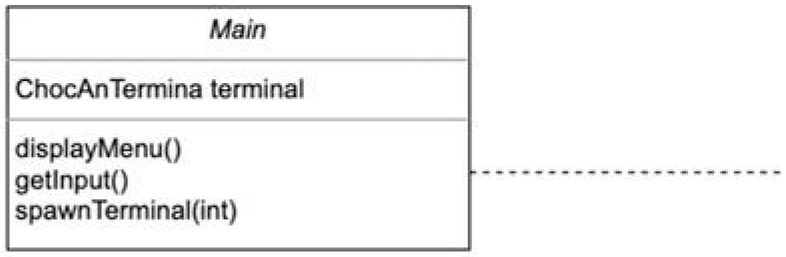
- ChocAn Main is the “main menu” of the CDPS. It runs at startup and whenever a user exits a terminal; it also allows the user to switch between the two types of terminals.
- The Manager Terminal allows adding, editing, and removal of members and providers from the CDPS database. It is meant to be used by managers.
- The Provider Terminal allows providers with valid ID to extend services to members. Services provided through the provider terminal will be automatically recorded based on the member’s ID.
- The Provider Database contains all information that is relevant to the provider and saves and load provider information from the disk at run time.
- The Member Database contains all information that is relevant to the member and saves and loads member information from the disk at run time.
- The Service Record contains the history of each transaction that a provider submits. Each record will contain and manage a number of data elements.
- The Service Database contains all information that is relevant to the service and saves and loads service information from the disk at run time.

4. System Architecture

4.1 Architecture Overview

The CDPS system architecture consists of a main terminal program with two user modes and four database files. The user modes provide different functionalities for providers and managers, while the files contain data on providers, provider services, members, and member service records.

4.2 Main



Main will create a terminal to match the one selected in its menu.

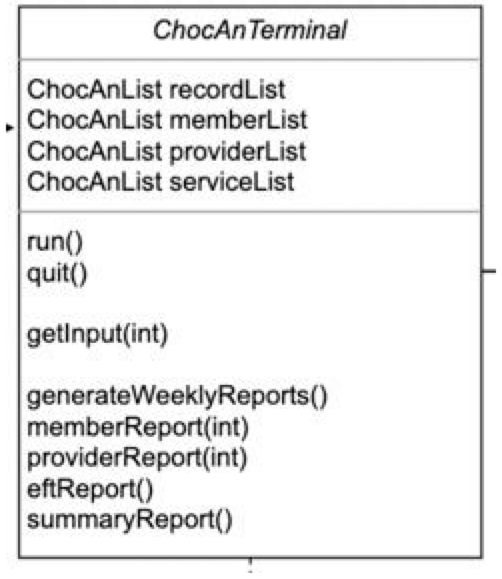
After backing out of terminal, main will give the option to spawn a different terminal.

Main is the module that runs at initial startup of the ChocAn system, displays the main menu options, and allows the user to select either a provider or manager terminal. When a user exits their terminal, they are returned to this menu to reselect a different terminal, or exit the program entirely and shut down the system.

The ChocAn Main module has one variable, and three methods:

- *ChocAnTerminal* terminal
- displayMenu()
- getInput()
- spawnTerminal()

4.2.1 ChocAnTerminal—Base class



The ChocAnTerminal is a base class for the two terminal types. It provides the basic terminal functionality common to both. ChocAnTerminal has four lists that hold relevant provider and member information, populated at runtime from database files. The lists implement and extend the Java list interface into a type called *ChocAnList*, and are called:

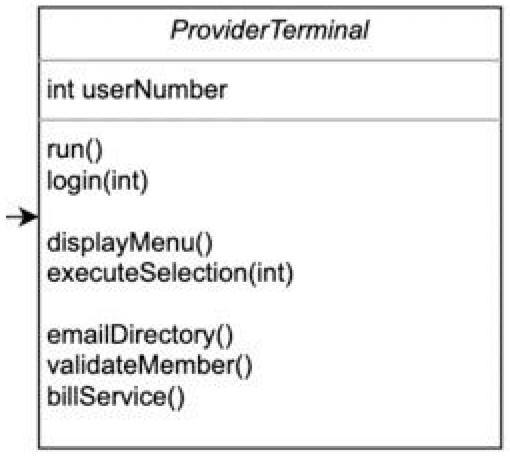
- *ChocAnList* recordList
- *ChocAnList* memberList
- *ChocAnList* providerList
- *ChocAnList* serviceList

ChocAnTerminal also contains the following methods:

- run()
- quit()
- getInput()
- generateWeeklyReports()
- memberReport()

- providerReport()
- eftReport()
- summaryReport()

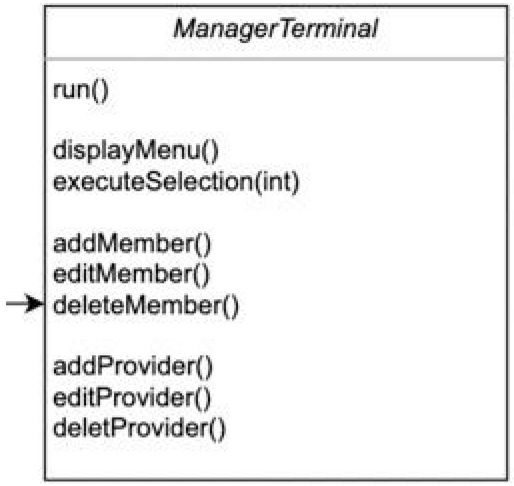
4.2.2 ProviderTerminal



ProviderTerminal extends the ChocAnTerminal base class to implement provider-specific functionality. Providers are prompted to enter their provider number for validation, and upon a success, they are given a menu of options to extend service to a member. Member ID numbers are entered once when services are initiated, and once again at the end, when a service record is created. ProviderTerminal has one variable and seven methods:

- *int* userNumber
- run()
- login()
- displayMenu()
- executeSelection(*int*)
- emailDirectory()
- validateMember()
- billService()

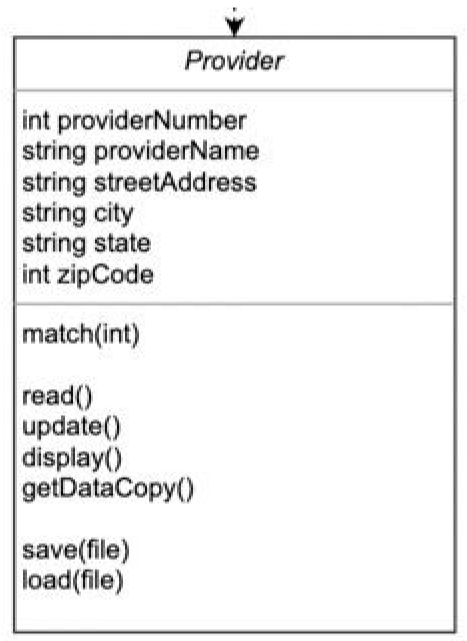
4.2.3 Manager Terminal



ProviderTerminal extends the ChocAnTerminal base class to implement manager-specific functionality. Members and providers may be added, deleted, or have their records edited and updated. ProviderTerminal has no variables and nine methods:

- run()
- displayMenu()
- executeSelection(*int*)
- addMember()
- editMember()
- deleteMember()
- addProvider()
- editProvider()
- deleteProvider()

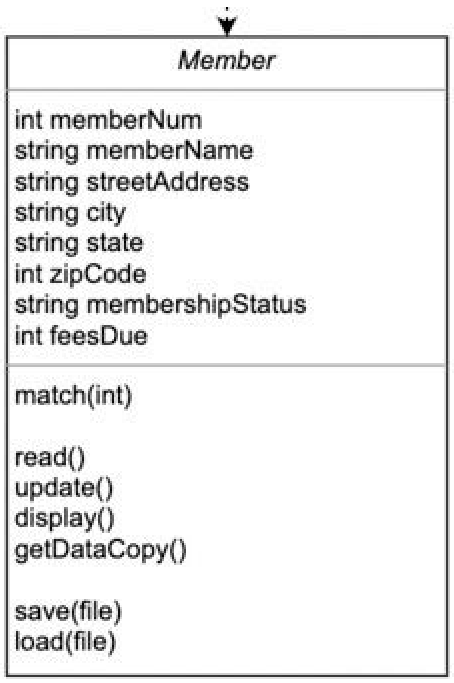
4.3 Provider



The Provider class is a class containing all relevant information about a ChocAn provider. On terminal launch, the data for each provider is read from file and stored in an instance of the Provider class, and on terminal exit it is written back to disk. The Provider class contains six variables and seven methods:

- *int* providerNumber
- *string* providerName
- *string* streetAddress
- *string* city
- *string* state
- *int* zipCode
- match(*int*)
- read()
- update()
- display()
- getDataCopy()
- save()
- load()

4.4 Member

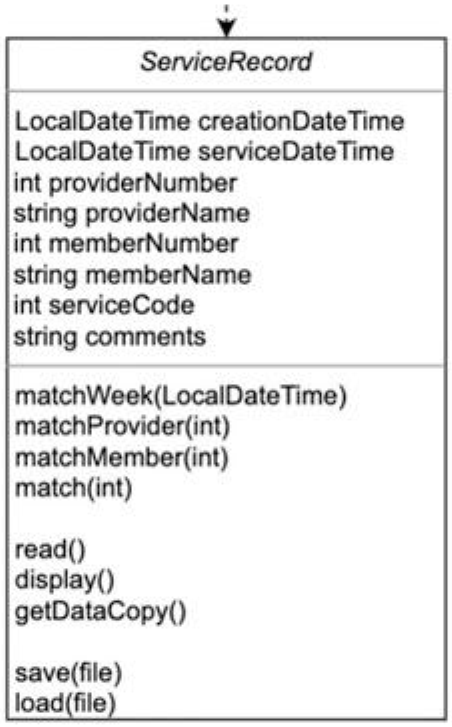


The Member class is a class containing all relevant information about a ChocAn member. On terminal launch, the data for each member is read from file and stored in an instance of the Member class, and on terminal exit it is written back to disk. The Member class contains eight variables and seven methods:

- *int* memberNumber
- *string* memberName
- *string* streetAddress
- *string* city
- *string* state
- *int* zipCode
- *string* membershipStatus
- *int* feesDue
- match(*int*)
- read()
- update()
- display()
- getDataCopy()

- save()
- load()

4.5 ServiceRecord

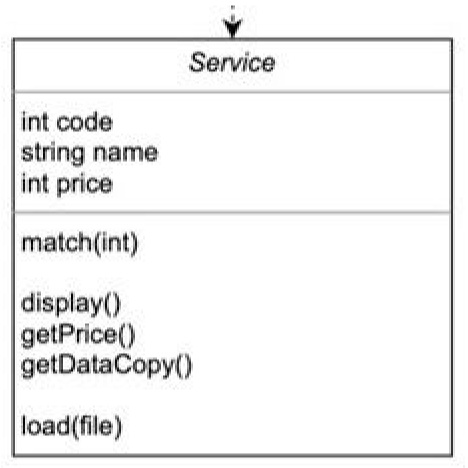


The ServiceRecord class is a class containing all relevant information about a single service that was provided to a ChocAn member. On terminal launch, the data for each provided service is read from file and stored in an instance of the ServiceRecord class, and on terminal exit it is written back to disk. The ServiceRecord class contains eight variables and nine methods:

- *LocalDateTime* creationDateTime
- *LocalDateTime* serviceDateTime
- *int* providerNumber
- *string* providerName
- *int* memberNumber
- *string* memberName
- *int* serviceCode
- *string* comments

- matchWeek(*LocalDateTime*)
- matchProvider(*int*)
- matchMember(*int*)
- match(*int*)
- read()
- display()
- getDataCopy()
- save()
- load()

4.6 Service



The Service class is a class containing all relevant information about a type of service that may be provided to a ChocAn member. On terminal launch, the data for each type of service is read from file and stored in an instance of the Service class, and on terminal exit it is written back to disk. The ServiceRecord class contains three variables and five methods:

- *int* code
- *string* name
- *int* price
- match(*int*)
- display()

- getPrice()
- getDataCopy()
- load()

5. Detailed System Design 1

5.1 ChocAn Main

The program's main method will allow the users to generate weekly reports, start the provider terminal, or start the manager terminal.

5.1.1 Main Methods

- **displayMenu** will output the following options to the command line and wait for the user's choice. It will loop until the exit option is selected.
    - Provider Terminal
    - Manager Terminal
    - Generate Weekly Reports
    - Exit
- **getInput** will take user input and handle bad input. It will return input that is correct.
- **spawnTerminal** will begin running the terminal requested by the user. It is also able to generate weekly reports, though in this case, it simply creates a base terminal class that generates the reports, then exits back to the main menu.

5.2 ChocAnTerminal class

The base terminal class is a container that holds all the ChocAn data—including service records, members, providers, and the service directory—and allows it to be acted upon. The methods will be limited to run, quit, get input, and generate weekly reports.

5.2.1 ChocAnTerminal Fields

The terminal class will have four discrete containers for data: service records, members, providers, and the service directory. Each of these will be a list extended from the Java List class (section 5.5). The various fields are described below.

5.2.2 ChocAnTerminal Methods

- **run** will have additional functions in derived terminal classes, but in the base class, it is limited in its duties. Run will populate all available lists by running their load function to pull data from files into memory. Next, it will run the generateWeeklyReport method; then it will quit.
- **quit** will save all to disk all service record, member, provider, and directory data, then exit out of the terminal.
- **generateWeeklyReports** will call the various report methods to create all possible reports.
  - First, it will create a report for each member that took part in a service through the last week. It will iterate through the member list, and for each member in the list it, will check to see if there is at least one record for the current member in the current week. If it finds one, it'll run the memberReport method against the member.
  - It will repeat the above process for providers,
  - then it will call the eftReport method,
  - and finally it will call the summaryReport method.
- **memberReport** will take a member ID as an argument. It will then attempt to grab the member's info from the list, and if the ID isn't found, it will abort. Next, it will create a file and write the member info. Then, the method will iterate through the service record list, searching for services associated with the given ID; for each service matching the current week, the service record will be written to the file.
- **providerReport** will take a provider ID as an argument. It will then attempt to grab the provider's info from the list, and if the ID isn't found, it will abort. Next, it will create a file and write the provider info. Then, the method will iterate through the service record list, searching for services associated with the given ID; for each service matching the current week, the service record will be written to the file. Additionally, it will tabulate the totals fees for services provided during the week and add these to the end of the report.

- **eftReport** will first create and open a file to represent the report. It will then iterate through the list of providers, and for each provider ID, it will check the service record to find services matching the ID and week. If at least one is found, it will tabulate in the report the total fees in each service record for the week. Then, the provider name, ID and total fees for the week will be added to the report.
- **summaryReport** will first create and open a file to represent the report. It will then iterate through the list of providers, and for each provider ID, it will check the service record to find services matching the ID and week. The total costs and number of consultations will be tabulated then added to the report. Finally, the total fee from all providers will be calculated and added to the report.

5.3 ManagerTerminal class

The manager terminal will extend the ChocAnTerminal. It will display the options for a manager terminal, wait for user input, and execute the user selection.

5.3.1 ManagerTerminal methods

- **run** will first load all the data into the lists as described in 5.2.1. Next, it will enter a loop; it will call displayMenu, then wait for user input, and then execute the user selection. Upon exit, it will return to the main menu (5.1.1).
- **displayMenu** will display the menu options for the manager terminal. When a selection is made, it will be executed by calling the appropriate method. The options are as follows:
    - Add member
    - Edit member
    - Delete member
    - Add provider
    - Edit provider
    - Delete provider
    - Weekly report

    - Member report
    - Provider report
    - EFT report
    - Exit
- **addMember** will create a new member item and populate its fields by calling its *read* method (5.8.2). It will then attempt to insert it into the member list, aborting if there is already a member in the member list with the given ID.
- **editMember** will take input from the user in the form of a member ID. It will attempt to grab the member with this ID from the list, and if it finds one, it will call its *update* method (5.8.2); it will abort if it cannot find a matching member.
- **deleteMember** will take input from the user in the form of a member ID. It will attempt to remove a member with a matching ID from the list; it will abort if it cannot find a matching member.
- **addProvider** will take input from the user, formatting it as a provider and attempting to insert it into the provider list. It will abort if there is already a provider in the provider list with the given ID.
- **editProvider** will take input from the user in the form of a provider ID. It will attempt to grab the provider with this ID from the list and allow the user to modify its fields; it will abort if it cannot find a matching provider.
- **deleteProvider** will take input from the user in the form of a provider ID. It will attempt to remove a provider with a matching ID from the list; it will abort if it cannot find a matching provider.

The methods for individual report generation are located in the base terminal.

5.4 ProviderTerminal class

Unlike the manager terminal, this terminal requires a valid provider ID on startup. If an invalid ID is given, the user can either re-enter a number or to exit. If a valid ID is given, the provider will have access to a menu that will allow them to bill for services and request a provider directory.

5.4.1 ProviderTerminal Fields

The provider terminal will include as a field the provider ID entered on startup (5.4). All transactions done within the terminal requiring a provider ID will use this stored ID.

5.4.2 ProviderTerminal Methods

- **run** will first load all the data into the lists as described in 5.2.1. Next, it will enter a loop; it will call displayMenu, then wait for user input, and then execute the user selection. Upon exit, it will return to the main menu (5.1.1).
- **displayMenu** will display the menu options for the provider terminal. When a selection is made, it will be executed by calling the appropriate method. The options are as follows:
  - Validate card
  - Bill service
  - Email provider directory
- **emailDirectory** will take the service list and save its contents to a file, which will then be emailed to the active provider by an external service.
- **validateMember** will take a member ID as an argument and iterate through the members list. If a matching member ID is found in the list, their validation status is displayed. If no matching member is found, “Not Found” is displayed.
- **billService** will provide a way to add a new service record. It will create a new service record and call its read function, passing in the member ID and the provider ID. It will check the service entered against the service directory to ensure it is valid. If it is, the record will be added to the list of services.

5.5 ChocAnList class

This data structure extends the Java List class to include a few extra functions. All service records, members, providers, and service directory data that is stored in memory will be stored in ChocAnLists.

5.5.1 ChocAnList Methods

- **save** will take a file name as an argument and iterate through the list, calling each element's save method with a unique file name, saving all the lists content to disk.
- **load** will take a file name as an argument, reading from it and calling the load function for new data, appending each to the list until the end of file is reached.
- **match** will take a member or provider ID as an argument, iterating through the list and checking for items associated with that ID.

5.6 ServiceRecord class

The ServiceRecord class will contain the history of each transaction a provider submits. Each record will contain a number of fields, listed below, as well as having several methods to manage them.

5.6.1 ServiceRecord Fields

The fields will keep track of every aspect of a provider-submitted transaction, including the time of the transaction, the date of the service, the name and ID of the member, the name and ID of the provider, the service code, and a comments field. Dates and times will be stored as a LocalDateTime; numbers will all be integers; and everything else will be contained in strings. This class contains names and IDs, rather than simply IDs, to avoid complications in the case that a manager deletes a member or provider and the ID number is reused. All fields will be private.

5.6.2 ServiceRecord Methods

- **matchWeek** will take no arguments and return a Boolean. The return value will be True if the service occurred during the last calendar week as measured by the system date/time. This will be used for reports which include services rendered within the last week.
- **matchProvider** will take a provider ID as an argument and return a Boolean indicating if it exists in the service record.

- **matchMember** will take a member ID as an argument and return a Boolean indicating if it exists in the service record.
- **read** will be used by providers who need to record a service rendered to a member; as such, it will take as arguments the provider ID, the provider Name, the member ID, the member name—and, finally, the service directory, which will be needed to verify the service. The provider will be prompted to fill in the fields of the service record. If there is no matching service ID, an error will be displayed, and the provider will be able to exit or try again.
- **display** will display the contents of the record in a readable form.
- **getDataCopy** will return the contents of the service, packaged into a dummy public class that will only contain a copy of the fields. This will be used for ease of access by report-generation functions.
- **save** will take a writable filename as an argument. It will save all fields from the service record into the given file.
- **load** will take a readable filename as an argument. It will load all fields from the given file, line by line, into the service record.

5.7 Provider class

The provider class will include all of the basic information about providers as well as methods to manage the fields.

5.7.1 Provider Fields

Information about the providers will be stored in the fields, including the provider ID, name, street address, city, state, zip code. The IDzip code will be integers, and all other fields will be strings. All fields will be private.

5.7.2 Provider Methods

- **match** will take a provider ID as an argument and compare it to the provider ID field. It will return a Boolean to indicate if they match.

- **read** will take user input and assign it to fields. All input will be directly added, and the only verification will be done on the ID to make sure it's the right number of digits. Any editing can only be done by managers through the manager terminal.
- **update** will allow managers to go element-by-element, prompting to update any or all of the fields.
- **display** will output all fields in a human-readable way.
- **getDataCopy** will return the contents of the provider, packaged into a dummy public class that will only contain a copy of the fields. This will be used for ease of access by report-generation functions.
- **save** will take a writable filename as an argument. It will save all fields from the provider into the given file.
- **load** will take a readable filename as an argument. It will load all fields from the given file, line by line, into the provider.

5.8 Member class

The member class will include all of the basic information about members as well as methods to manage the data.

5.8.1 Member Fields

Information about the providers will be stored in the fields, including the member ID, name, street address, city, state, zip code, membership status, and any fee due. The ID, fees owed, and the zip code will be integers, and all other fields will be strings. All fields will be private.

5.8.2 Member Methods

- **match** will take a member ID as an argument and compare it to the member ID field. It will return a Boolean to indicate if they match.
- **read** will take user input and assign it to fields. All input will be directly added, and the only verification will be done on the ID to make sure it's the right number

of digits. Any editing can only be done by managers through the manager terminal.

- **update** will allow managers to go element-by-element, prompting to update any or all of the fields.
- **display** will output all fields in a human-readable way.
- **getDataCopy** will return the contents of the member, packaged into a dummy public class that will only contain a copy of the fields. This will be used for ease of access by report-generation functions.
- **save** will take a writable filename as an argument. It will save all fields from the provider into the given file.
- **load** will take a readable filename as an argument. It will load all fields from the given file, line by line, into the provider.

5.9 Service class

The service class will represent the service directory in memory—so, unlike, the other data items, it has no reason to be editable. It primarily exists to serve other classes, which will use it to check service IDs and prices.

5.9.1 Service Fields

Services will need three fields, all private: the ID, the price, and the name. The ID will be an integer, the price will be represented by two integers (one for the dollar value and one for the cents), and the name will be a string.

5.9.2 Service Methods

- **match** will take an integer ID as an argument and search for a service in the list with that ID. The method will return a Boolean to indicate if it finds a match.
- **display** will output all fields in a human-readable way.
- **getDataCopy** will return the contents of the service, packaged into a dummy public class that will only contain a copy of the fields. This will be used for ease of access by report-generation functions.

- **getPrice** will return the value of the price field.
- **load** will take a readable filename as an argument. It will load all fields from the given file, line by line, into the service.