**Term Project: *ChocAn***

Design Document

**Table of Contents**

1 Introduction 3

1.1 Purpose and Scope 3

1.2 Target Audience 3

1.3 Terms and Definitions 3,4

2 Design Considerations 5

2.1 Constraints and Dependencies 5

2.1.1 Constraints 5

2.1.2 Dependencies 5

2.1.2.1 Code Reuse and Licensing 5

2.1.2.2 Hardware and Operating Systems 6

2.2 Methodology - Scrum 6

2.2.1 Scrum Phase 1: Outline, Planning, and Architectural Design 7

2.2.2 Scrum Phase 2: Sprints 7, 8

2.2.3 Scrum Phase 3: Project Closure 8

3 System Overview 9

4 System Architecture 10

4.1 Entity 10, 11

4.1.1 Patient 11

4.1.2 Provider 12

4.2 Service 12

4.3 Transaction 13

4.4 Database Wrapper 14

4.5 Terminals 14, 15

4.5.1 Provider Terminal 15

4.5.2 Manager Terminal 15

4.5.3 Operator Terminal 15

4.6 Database 16, 17

4.7 Automated Report Generation 17

5 Detailed System Design 18

5.1 Entity Superclass 18

5.1.1 Patient Object 18

5.1.1.1 Provider Terminal 19

5.1.2 Provider Object 19

5.1.2.1 Provider Error Handling 19

5.2 Service Object 19, 20

5.2.1 Service Error Handling 20

5.3 Transaction Object 20

5.3.1 Transaction Error Handling 20

5.4 Database Wrapper Object 21

5.4.1 Error Handling 21

5.4.2 Add Operations 21

5.4.3 Get Operations 22, 23

5.4.4 Update Operations 23

5.4.5 Remove Operations 23

5.5 Terminal 23

5.5.1 Provider Terminal 23, 24

5.5.2 Manager Terminal 24, 25

5.5.3 Operator Terminal 25, 26

5.6 Database 26

**1.** **Introduction**

This document describes the overall structural design and implementation details for the Chocoholics Anonymous software. It discusses how system specifications will be converted into executable systems in order to facilitate services, billing, and account management for Chocoholic Anonymous patients. In order to formulate a comprehensive plan, the design considerations, system overview, system architecture, and detailed system design for CA have been analyzed.

**1.1 Purpose and Scope**

The purpose of the software design document is to provide a plan for converting project specifications to executable programs as well as a direction for organization and workflow given the implementation details in the development of CA. The scope of the CA project is constrained to the requirements and their dependencies as outlined in “Requirements\_Doc\_FINAL\_10.18.16”.

**1.2 Target Audience**

The target audience for this document is the CA development team, with a secondary audience of CA stakeholders and the technical staff of Chocoholics Anonymous.

**1.3 Terms and Definitions**

**Accounts Payable Report** – A list of all payments due to Providers during a one-week period.

**Acme**–Acme Accounting Services, a third-party organization responsible implementing EFTs and administers the membership of ChocAn patients.

**ChocAn**–Chocoholics Anonymous, the organization.

**CA**–Chocoholics Anonymous, the software product specified in this document.

**CA Data Center–**The server hosting the CA database within the data center, owned and operated by ChocAn.

**CRUD**–Create, Read, Update, Delete. The responsibilities of CA.

**EFT**–Electronic Funds Transfer.

**EFT Report** – A record of services charged by Providers during a one-week period.

**Manager**–Supervisors at ChocAn who receive patient, provider, EFT, and accounts payable reports.

**Operator**–Employees of ChocAn who create, update, and delete patients and providers.

**Patient**–Any individual who accesses services and is billed for them.

**Patient Reports** **--** A report of all services provided to a Patient during a one-week period.

**Provider**–Any company or individual who provides services and bills patients.

**Provider Report --** A report of all services provided by a Provider during a one-week period.

**Provider Directory**–A list of Services provided by ChocAn.

**Service**–Any procedure or treatment performed by a provider for a patient.

**Software Development Team**–The group of programmers in charge of creating the software.

**Transaction**–A single billable service given by a provider during a visit.

**2. Design Considerations**

This section describes the software development methodology used to develop the CA software and the constraints placed upon the project as dictated by functional and nonfunctional specifications outlined during requirements gathering.

**2.1 Constraints and Dependencies**

**2.1.1 Constraints**

* Since the CA team does not have access to dedicated servers, the ChocAn database will be simulated and hosted on a local machine
* As specified by ChocAn, the three CA terminals (provider, manager, and operator) will be simulated in a single command-line application
* The ChocAn membership cards and card-readers will be simulated in the CA command-line application by entering Patient IDs manually
* The ChocAn database and information therein will not be encrypted due to limitations in time and expertise
* In an actual deployment, Acme needs to access the database to record membership payments and update membership statuses
* In an actual deployment, the CA system needs to be running consistently to print reports each Friday night at midnight
* In an actual deployment, the CA system needs to be accessible over the Internet to provide access to ChocAn offices and Providers

**2.1.2 Dependencies**

**2.1.2.1 Code Reuse and Licensing**

CA will use several existing frameworks and libraries:

* **Java Runtime Environment (JRE) Version 8:** The framework on which CA will run. JRE is cross-platform, and is required on any machine running CA. The Oracle Binary Code License Agreement for the Java SE Platform products and JavaFX states that Oracle grants “a non-exclusive, non-transferable, limited license without license fees to reproduce and use internally the Software complete and unmodified for the sole purpose for running Programs” upon the condition that the terms are accepted.
* **SQLite relational database management system:** This is a standalone application that will manage the CA database. The database will store ChocAn data in Patient, Provider, Transaction, and Service tables. SQLite has been dedicated to the public domain by its authors.
* **Java Database Connectivity API (JDBC):** The API used to interface with the SQLite database. Oracle grants permission to use the software upon agreement of specified terms. The [SQLite JDBC](https://bitbucket.org/xerial/sqlite-jdbc/overview) driver is written by xerial.org under the Apache License 2.0, which requires the following:
  + include a copy of the license in any redistribution you may make that includes this software
  + provide clear attribution to xerial.org for any distributions that include this software

CA itself is proprietary software and will not be released under any public license. The code cannot be used or redistributed, commercially or otherwise.

**2.1.2.2 Hardware and Operating Systems**

To use CA, a computer running a minimum of Windows 7, macOS 10.11.6, or Ubuntu 14.04 is required.

**2.2 Methodology - Scrum**

The CA team will be employing a derivative of the Scrum method in agile development with heavy emphasis on object-oriented design. While the high-level overview of the project is developed by the team as a whole, implementation will be executed in a modular fashion with each developer focusing on a particular feature. Scrum is well-suited for this because of the rigorous yet flexible planning phase followed by incrementally-developed versions of the software (“sprints”). The plan for CA’s development is described in the following three sections.

**2.2.1 Scrum Phase 1: Outline, Planning, and Architectural Design**

Requirements documentation and architectural design will be well-defined by this phase. The nature of the CA software requires a reasonable amount of planning beforehand but shall allow for changes as problems are encountered and more effective implementations are discovered.

**2.2.2 Scrum Phase 2: Sprints**

The bulk of CA will be implemented in a series of “sprints,” or cycles. Where our methodology differs from Scrum is that we will not have a Scrum Master, an individual who is given the organizational responsibilities for the series of sprints. Our team is small enough that organizational matters may be distributed among developers.

At the beginning of each sprint, the CA team will hold a meeting to discuss what should be achieved during the cycle. This includes prioritizing features, conversations with the client as needed, and delegation of tasks to sub-teams.

Developers will be divided into sub-teams of two at most and will be responsible for implementing their respective features. This modular approach will help the isolation of bugs and create a malleable system that can easily adapt to change.

At the end of each sprint, the modular features will be integrated into a cohesive structure. Once the new feature set is approved, the changes will be merged into the Git master branch.

This process of sprints will repeat until the CA software has been fully implemented, tested, and finalized.

**2.2.3 Scrum Phase 3: Project Closure**

The final phase will be a review of the overall process and a presentation to ChocAn, the client. The product will be assessed by ChocAn to ensure all requirements are met and any further documentation needed will be written.

**3. System Overview**

Individual interfaces for Managers, Operators, and Providers will be simulated using a command-line terminal to interact with CA. Using their respective simulated interfaces, each user will enter commands into the terminal. CA will then process and respond to these commands, querying the local database where appropriate (see Figure 1).

Every interface has defined privileges restricting which functionality of CA they may access. These user-privilege relationships are defined by the *User-privileges* table below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *User Privileges* | **CRUD Patient** | **CRUD Provider** | **CRUD Service** | **Print Reports** | **Request Provider Directory** | **Bill For Service** |
| **Manager** |  |  | ✓ | ✓ | ✓ |  |
| **Operator** | ✓ | ✓ |  |  |  |  |
| **Provider** |  |  |  |  | ✓ | ✓ |
| *A ‘✓’ indicates accessible functionality by the user* | | | | | | |

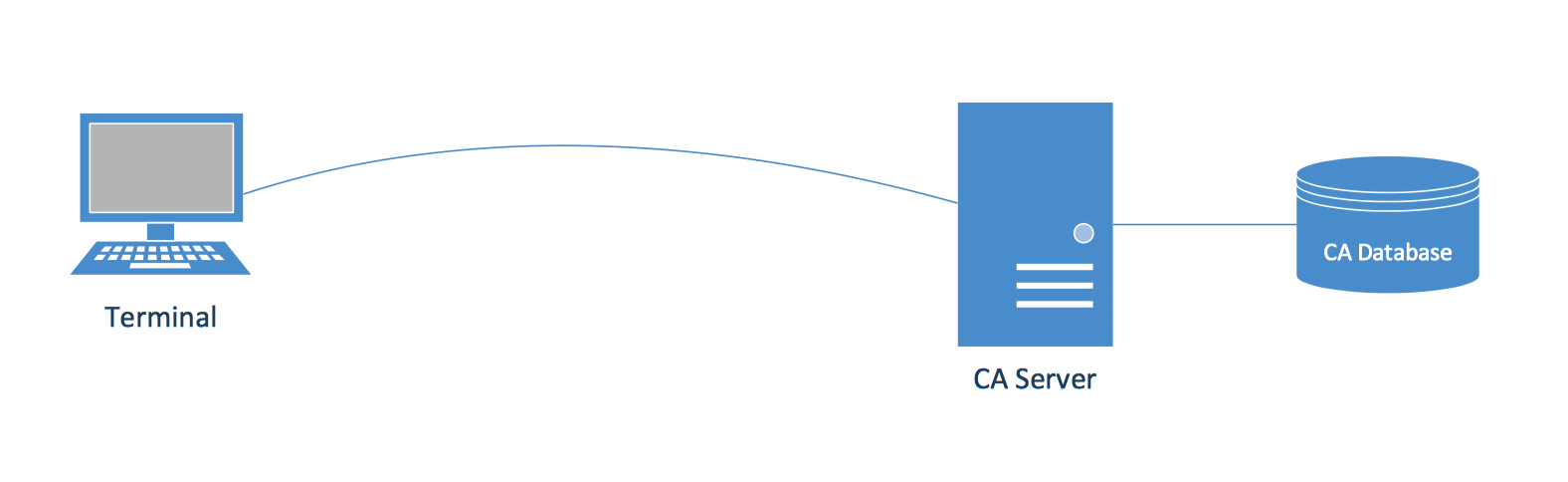


Figure 1

**4. System Architecture**

This section outlines the objects and subsystems that CA will possess. The CA system will separate system concerns into Patient, Provider, Service, and Transaction objects that encapsulate required data fields. The Database Wrapper object will query and update data from the Database subsystem, and the Terminal subsystem will allow Providers, Managers, and Operators a means to generate reports, query the database, and update the database based on their respective permissions.

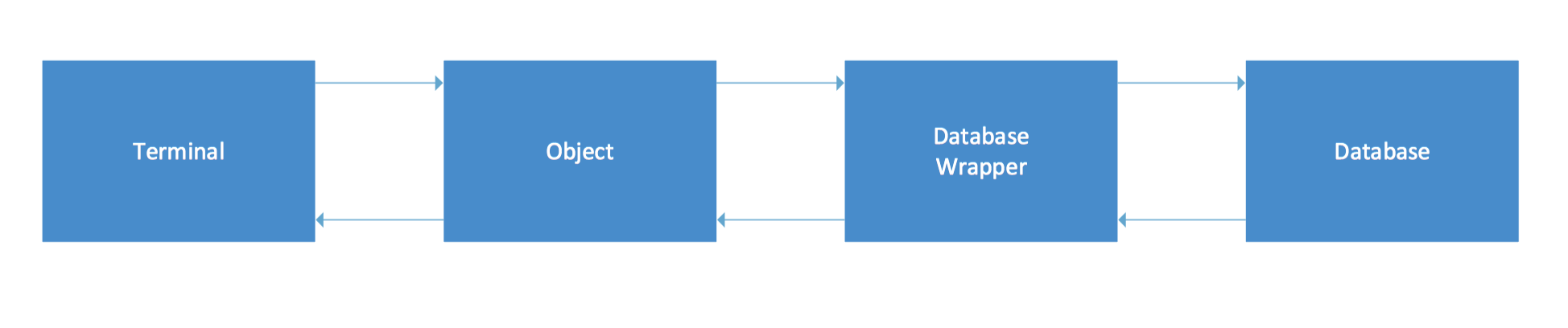


Figure 2

**4.1 Entity**

Entity is the abstract superclass to the Patient and Provider classes. It defines a set of member fields and methods common to Patient and Provider. The private member fields of Entity are idNumber, address, city, state, zipcode, and status. Entity defines public methods for getting and setting each field. Any method that sets private member variables in Entity ensures valid input prior to assignment.

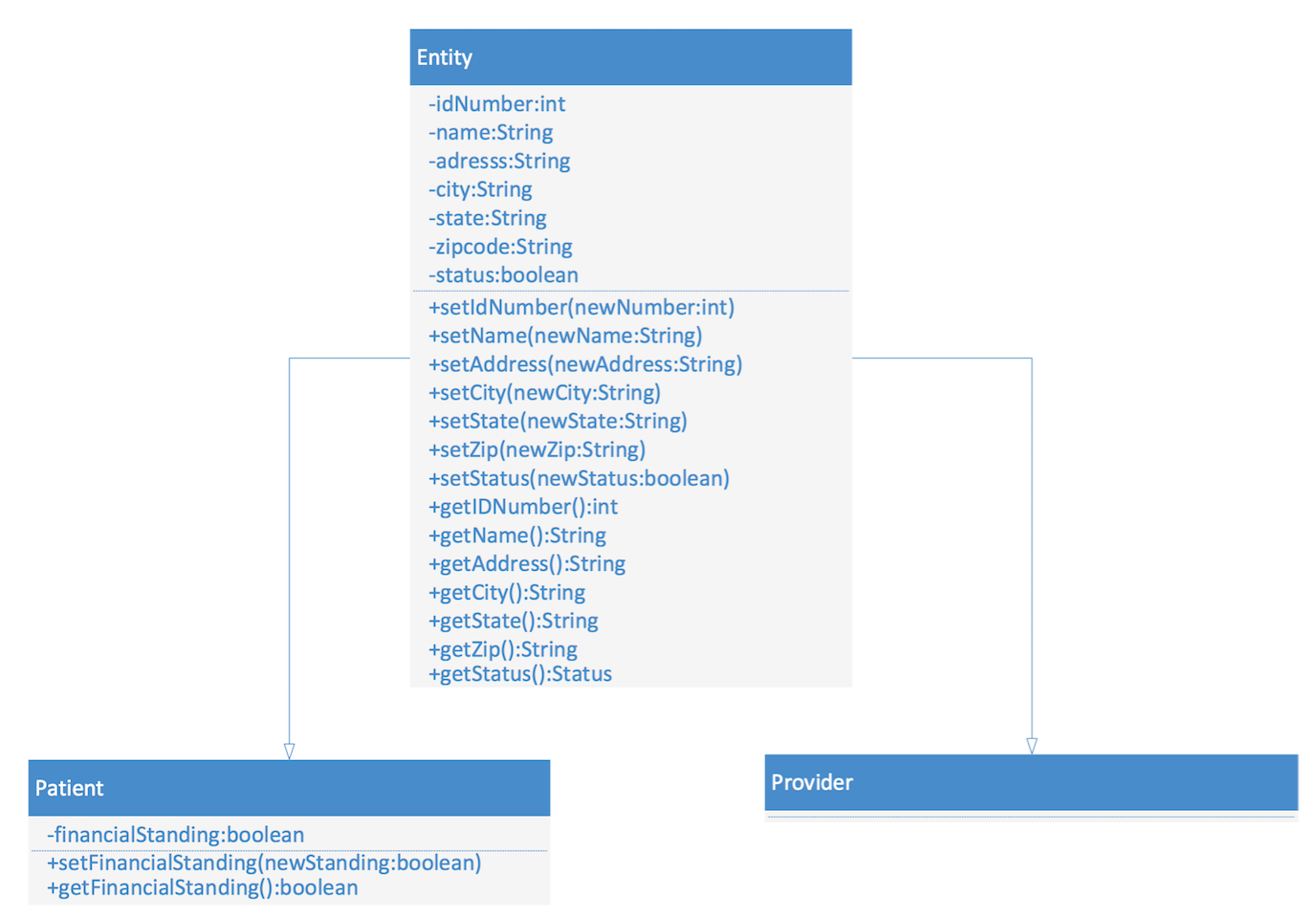


Figure 4.1

**4.1.1 Patient**

The Patient class extends Entity. The Patient object is responsible for temporarily storing Patient information during the execution of CA functionality. Patient information includes the fields inherited from the Entity class and a financialStanding field, which indicates the Patient’s current membership standing.

The Patient object is used with the Provider and Service objects to create Transaction objects. In addition, the Patient object is used with Transaction objects to generate Patient Reports. The Patient UML class diagram shown in Figure 4.1 defines the member fields and methods for the class.

**4.1.2 Provider**

The Provider class extends Entity. The Provider object is responsible for temporarily storing Provider information during the execution of CA functionality. Provider information includes the fields inherited from the Entity class.

The Provider object is used with the Patient and Service objects to create Transaction objects. In addition, the Patient object is used with Transaction objects to generate Provider Reports. The Provider UML class diagram shown in Figure 4.1 defines the member fields and methods for the class.

**4.2 Service**

The Service class defines the member fields and methods for a Service object. The private member fields include name, code, and fee. The Service object is used with the Patient and Provider objects to create Transaction objects. The Service UML class diagram shown in Figure 4.2 defines the member fields and methods for the class.



Figure 4.2

**4.3 Transaction**

The Transaction class defines the member fields and methods for a Transaction object. The private member fields include idNumber, patientID,providerID, consultationNumber,date,serviceDate and comments. The date field is a timestamp for when a Transaction object is instantiated. The serviceDate is the actual date when the service was provided. The comments field is for any Provider remarks and may be left empty.

A Transaction object acts as a temporary record of a single service provided to a patient. Each Transaction object stores information about an associated Patient, Provider and Service. Transactions are grouped into consultations, and are added to the database using a transactional model. That is, a consultation is prepared with any number of Transactions and the Transactions are sent to the Database Wrapper object as a group. Each Transaction object in a consultation has the same consultationNumber.

The Transaction UML class diagram shown in Figure 4.3 defines the member fields and methods for the class.

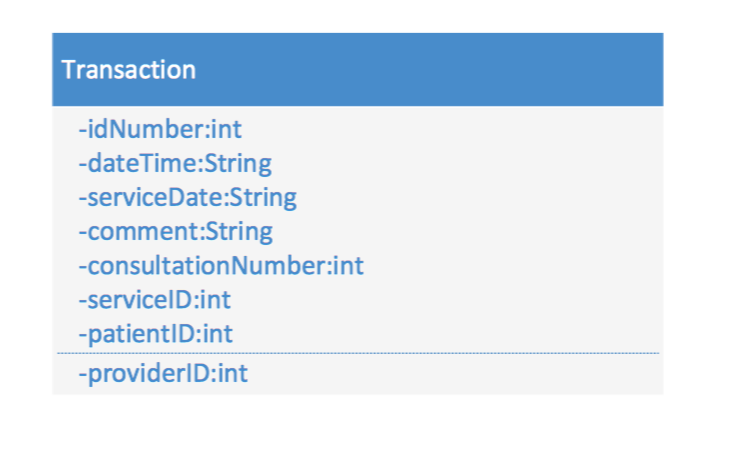


Figure 4.3

**4.4 Database Wrapper**

The Database Wrapper is an *interface* between the terminal and the database. The Database Wrapper handles CRUD calls from the terminal. These are implemented with the add, get, update, and remove methods. The backend to the Database wrapper is an SQLite database.

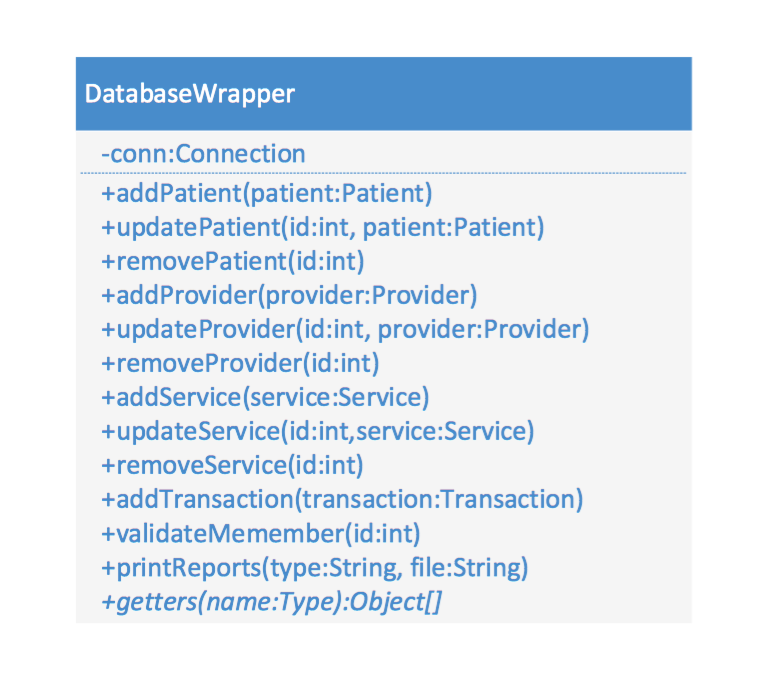


Figure 4.4

**4.5 Terminals**

The CA system shall simulate a single terminal application with command-line interface. The main menu will include three options:

1. Provider Terminal

2. Manager Terminal

3. Operator Terminal

When an option is chosen, a submenu with additional options or instructions will appear on the screen. All options will be numbered in ascending order as they appear on the screen, and the user will be asked to enter their desired option. The terminal will also display any errors associated with wrong or invalid inputs.

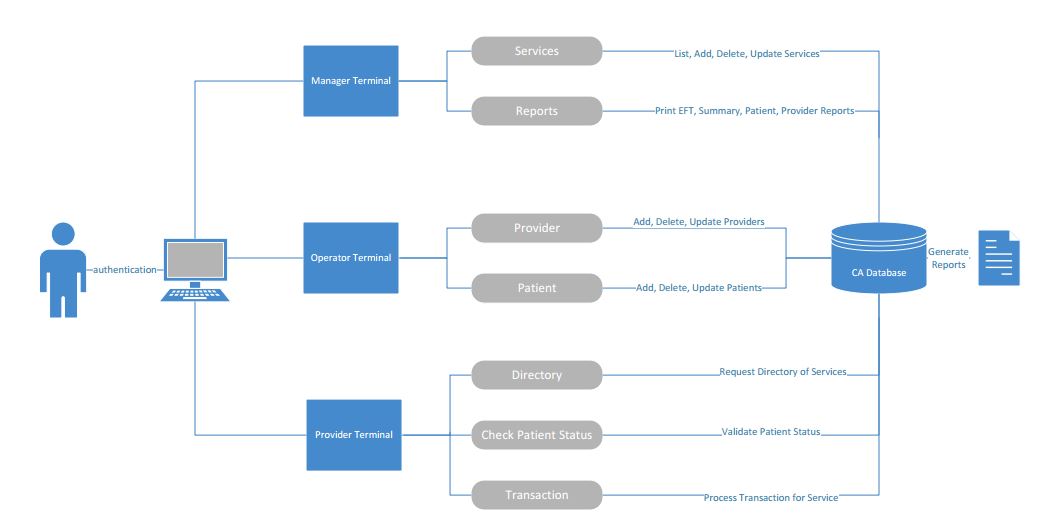
A detailed graphical representation of terminal menu flow is represented follows:

Figure 4.5

**4.5.1 Provider Terminal**

The provider terminal shall have options that allow providers to perform their duties through CA system. This will include the ability to display a Provider Directory, check member statuses, start a new transaction and bill for a service.

**4.5.2 Manager Terminal**

The manager terminal shall have options that allow managers to perform their duties through the CA system. This will include the ability to send or access reports for providers and members, and add, delete or update services.

**4.5.3 Operator Terminal**

The operator terminal shall have options that allow operators to perform their duties through the CA system. This will include the ability to add, delete or update a member or provider.

**4.6 Database**

The ERD defines the entities represented in the database. Each entity is a table with a row for each of its fields.

The relationships between entities reflects the transactional nature of the CA system. Data is stored in Patient, Provider and Service tables and is created, read, updated and deleted where appropriate by Managers and Operators. Transactions are generated by Providers and can be read by Managers. Note that Transactions store information about the Patient, Provider and Service involved in the transaction.

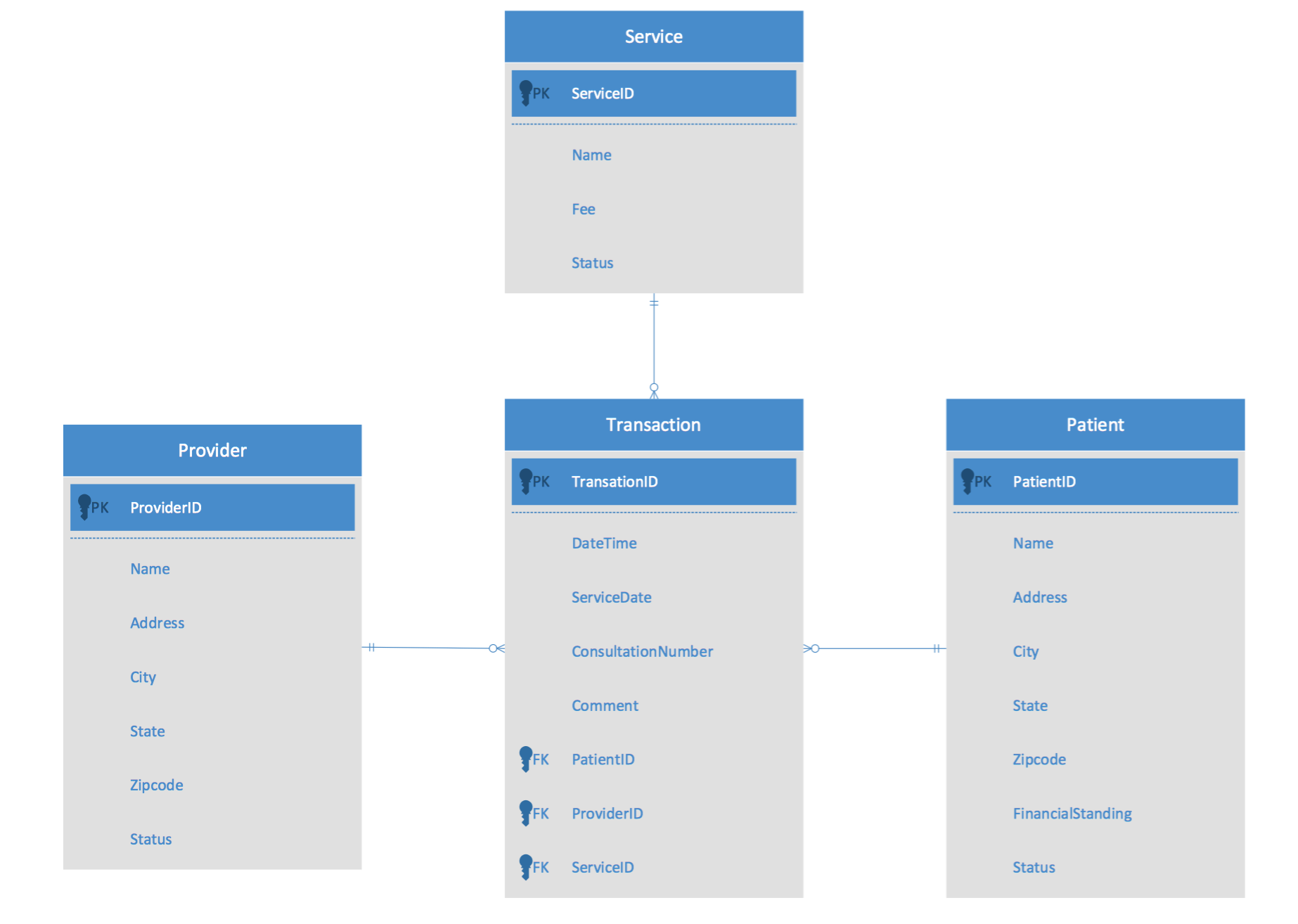


Figure 4.6

The following business rules were used to develop the cardinalities for the ERD:

* A Patient may have never been provided a ChocAn service
* A Patient may be provided multiple ChocAn services
* A Provider may have never billed for a ChocAn service
* A Provider may bill for multiple ChocAn services
* A Service may have never been billed
* A Service may be billed multiple times
* A Transaction is associated with one and only one Patient
* A Transaction is associated with one and only one Provider
* A Transaction is associated with one and only one Service

**4.7 Automated Report Generation**

In actual deployment, the ChocAn Data Center would run accounting procedures at midnight on Friday. These procedures would produce a set of reports saved to file and emailed to Providers, Patients and Managers. Because the CA system is a simulation, automated reports will not be implemented.

**5. Detailed System Design**

**5.1 Entity Superclass**

Entity is the abstract superclass to the Patient and Provider classes. It includes the following fields:

* idNumber, a 9-digit integer
* name, a string containing 25 characters or fewer
* address, a string containing 25 characters or fewer
* city, a string containing 14 characters or fewer
* state, a string containing exactly 2 characters
* zipcode, a string containing exactly 5 characters
* status, a boolean value, where true is **Active** and false is **Inactive**

**5.1.1 Patient Object**

The Patient object represents a single Patient. In addition to the fields inherited from the Entity superclass, the Patient object includes the field:

* financialStanding, a boolean value where true is **Good Standing**  and false is **Suspended**

Whenever CA handles an operation that involves a Patient, the data is encapsulated in a Patient object. For example, an SQL query for a Patient based on their idNumber number will return a Patient object. When a Patient is added to the database, a Patient object will be passed to the Database Wrapper’s addPatient function. At no point will bare strings or ID numbers be passed to the Database Wrapper. If a query involves a Patient, the entire object will be passed.

**5.1.1.1 Patient Error Handling**

Invalid Patient data is handled by an InputException class in the Patient constructor. For example, an address with more than 25 characters is handled as follows:

// ...

try {

// ...

if (address.length() > 25) {

throw new InputException(“Address too long.”);

}

// ...

} catch(InputException e) {

return;

}

Patient objects are validated when they are created. Since only Patient objects can be passed to the Database wrapper, only validated data will be passed to the Database for storage. Error checking during create and read operations ensures database integrity.

**5.1.2 Provider Object**

The Provider object represents a single Provider. It inherits all of its fields from the Entity superclass.

CA treats Provider objects in a similar manner to Patient objects. Whenever CA performs an operation involving a Provider, the data is encapsulated in a Provider object.

**5.1.2.1 Provider Error Handling**

Invalid Provider data is handled in the same way invalid Patient data is by using an InputException class in the Provider constructor.

**5.2 Service Object**

The Service object represents a single Service. It includes the following fields:

* name, a string containing 20 characters or fewer
* code, a 6-digit integer
* fee, a float (decimal) number
* status, a boolean value, where true is **Active** and false is **Inactive**

**5.2.1 Service Error Handling**

Invalid Service data is handled in the same way as invalid Service data by using a InputException class in the Service constructor.

**5.3 Transaction Object**

The Transaction object represents a single transaction. It has the following fields:

* idNumber, a 9-digit integer
* patientID, a 9-digit integer
* consultationID, a 9-digit integer
* providerID, a 9-digit integer
* serviceID, a 6-digit integer
* date, a 19-character String formatted MM-DD-YYYY HH:MM:SS
* serviceDate, a 10-character String formatted MM-DD-YYYY
* comment, a 100-character max String

**5.3.1 Transaction Error Handling**

Invalid Transaction data is handled in the same way a invalid Patient data by using a InputException class in the Transaction constructor. Note that many fields that make up a Transaction come from Provider, Patient or Service objects are automatically generated and do not need further validation. The date,serviceDate,and comment fields must be validated. The Database Wrapper ensures a Transaction’s idNumber is valid.

**5.4 Database Wrapper Object**

The Database Wrapper Object contains methods to perform CRUD operations on Patient, Provider, Service and Transaction entries in the database. The database will be queried and updated using the JDBC (Java Database Connectivity) library.

**5.4.1 Error Handling**

All calls to the Database wrapper will be enclosed in a try block. All failed operations will immediately exit out of the block without making any changes to the database. The catch blocks will implement error handling, and will use three types of exceptions: SQLException, AlreadyExistsException, and DoesNotExistException.

* SQLException encapsulates all failures with SQL operations. These are failures in the backend that should never be triggered during normal operation.
* AlreadyExistsException encapsulates errors in add operations where an object matching the new object already exists in the database.
* DoesNotExistException encapsulates errors in addTransaction operations where a Transaction with an invalid ID field is passed to the Database wrapper. It also encapsulates errors in get, update, and remove operations where the operation is passed an invalid ID.

**5.4.2 Add operations**

All add operations are passed as objects. If the user wishes to add a Patient to the database, the Terminal must create a Patient object and pass the object to the Database wrapper. There are no emplace commands that allow the Database Wrapper to create an object. Each add operation returns the ID of the object that was added or throws an exception.

**5.4.3 Get operations**

All get operations involve querying the database using one or more fields (ID, name, address, etc). The Database wrapper takes a table name and field, and prepares an SQL query for the database. The Database wrapper will either return an object (Patient, Provider, Service or Transaction), a vector of objects, or will throw an exception. Note that each get operation will never return an empty object or an empty vector if a matching record is not found in the database. It will always return an object or will trigger a DoesNotExistException. This simplifies and ensures consistent error handling.

The Database Wrapper will *never* allow a user to execute an arbitrary SQL command. All table calls are hard-coded inside specific functions such as getPatientByID or getProvidersByName; there will never be a generic get function that is publicly accessible.

The get operations will consist of the following:

* Patients:
  + getPatientByID(int)
  + getPatientsByName(String)
  + getPatientsByAddress(String)
  + getPatientsByCity(String)
  + getPatientsByState(String)
  + getPatientsByZipcode(String)
* Providers:
  + getProviderByID(int)
  + getProvidersByName(String)
  + getProvidersByAddress(String)
  + getProvidersByCity(String)
  + getProvidersByState(String)
  + getProvidersByZipcode(String)
* Services:
  + getServiceByID(int)
  + getServicesByName(String)
  + getServicesByPrice(float)
* Transactions:
  + getTransactionByID(int)
  + getTransactionsByDate(String)
  + getTransactionsByProvider(int)
  + getTransactionsByPatient(int)
  + getTransactionsByService(int)
  + getTransactionsByConsult(int)

**5.4.4 Update Operations**

Similar to add operations, an update operation takes an object and updates based on its idNumber. Note that the ID number in the database entry cannot change; only the object that is associated with the ID number can change. This is done to preserve the integrity of the Transaction table.

**5.4.5 Remove Operations**

The remove operation is a bit of a misnomer, since the Database Wrapper *never* removes anything. It simply sets the Status flag of the chosen row to 0, which sets a member as inactive. The object’s Status flag can be set back to 1 to reactivate a member with an update operation.

**5.5 Terminal**

**5.5.1 Provider Terminal**

The provider will be able to choose a desired option from the terminal. If one of the options is selected, the user will be asked for additional information as described below:

* Provider directory

No additional information required

* Check Patient status
  + Patient ID (9 digits):
* Start new consultation
  + Start new transaction
    - Member ID (9 digits):
    - Date of service (MM-DD-YYYY):
    - Service ID (6 digits):
    - Comments (Max. 100 characters):

*As a result, the transaction ID will be generated and stored in the database. It will be printed on the screen for personal records and future reference.*

* + End consultation

**5.5.2 Manager Terminal**

The manager will be able to choose a desired option from the terminal. If one of the options is selected, the user will be asked for additional information as described below:

* Services
  + List all available services
  + Add
    - Service name:
    - Service Fee:

*As a result, the service ID will be generated and stored in the database. It will be printed on the screen for personal records and future reference.*

* + Update
    - Service ID (6 digits):
    - Service name:
    - Service Fee:
  + Delete
    - Service ID (6 digits):
* Reports
  + Weekly Summary
  + EFT
  + Patient
    - Access
      * Patient ID (9 digits):
    - Send
      * Patient ID (9 digits)
  + Provider
    - List all available providers
    - Access
      * Provider ID (9 digits):
    - Send
      * Provider ID (9 digits):

**5.5.3 Operator Terminal**

The provider will be able to choose an option from the terminal. If one of the options is selected, the user will be asked for additional information as shown below:

* Provider
  + Add
    - Provider name:
    - Provider address:
    - Provider city:
    - Provider state:
    - Provider zip code:

*As a result, the provider ID will be generated and stored in the database. It will be printed on the screen for personal records and future reference.*

* + Update
    - Provider name:
    - Provider address:
    - Provider city:
    - Provider state:
    - Provider zip code:
  + Delete
    - Provider ID (9 digits):
* Patient
  + Add
    - Patient name:
    - Patient address:
    - Patient city:
    - Patient state:
    - Patient zip code:

*As a result, the member ID will be generated and stored in the database. It will be printed on the screen for personal records and future reference.*

* + Update
    - Patient name:
    - Patient address:
    - Patient city:
    - Patient state:
    - Patient zip code:
  + Delete
    - Patient ID (9 digits):

**5.6 Database**

The database will be implemented using the [SQLite database engine](https://sqlite.org/). SQLite does not use a server process and instead, saves the SQL database as a file on disk. Due to its simplicity, SQLite will be used in the development of the CA application for simulation purposes. In an actual deployment of the product, a more robust database management system would be selected.