Pharmacy Inventory Manager

CS 342 Group Project

Group 5

4/21/15

Version 2.0

Thomas Misiaszek

Abraheem Irheem

Daniel Moreno

Marcell Purham

Joseph Huff

1. Introduction

Taken from the project PDF:

Dr. Arsenius Alchemist runs a small pharmacy in rural Wisconsin. In the past Dr. Alchemist has lost money because he ran out of popular medications (e.g., Aspirin, Alka-Seltzer, Amoxicillin), which his customers wanted to purchase. Other times, he overstocked on medications, which subsequently expired before he could sell them. Typically, Dr. Alchemist buys medications in bulk. For instance, a box of Advil cans may contain 50 to a 100 cans of different sizes. He would like to have an inventory management system, keeping track at least of his stock of most popular medications (about two dozen in total) and their expiration dates. The system would advise him as to when it is appropriate to buy additional medications either because the current stock is running low or because it is about to expire.

The objective of our group for this assignment was to develop an application that could manage the inventory of pharmaceutical supplies, with capabilities of adding, removing, and notifying when drugs need to be replaced due to low stock or being close to their expiry dates.

The hardware used for development specifically was used on the following system specs:

Thomas Misiaszek's Hardware Specifications:

CPU: AMD FX-8320

GPU: ATI Radeon 7850

RAM: 8GB DDR3 1600

Motherboard: ASUS 990FX

Non-Volatile Memory: Samsung 850 EVO SSD 250GB

OS: Windows 8 Pro

Abraheem Irheem's Hardware Specifications:

CPU: Intel Core i7

GPU: Intel 4000

RAM: 8GB DDR3 1600

Motherboard: MSI Z87-G45

Non-Volatile Memory: Intel 730 Series SSD 240GB

OS: Windows 7 Pro

<u>Joseph Huff's Hardware Specifications:</u>

CPU: Intel Core i5

GPU: GT 520m

RAM: 8GB DDR3 1600

Motherboard: MSI Z87-G45tv

Non-Volatile Memory: Intel SSD 120GB

OS: Windows 8.1

<u>Marcell Purham's Hardware Specifications:</u>

CPU: Intel Core i7

GPU: GTX-960

RAM: 16GB DDR3 1866

Motherboard: ASUS Z87-Plus

Non-Volatile Memory: Intel 730 Series SSD 120GB

OS: Windows 8.1 Pro

<u>Daniel Moreno's Hardware Specifications:</u>

CPU: Intel Core i7

GPU: Intel(R) HD Graphics 4400

RAM: 16GB DDR3

Motherboard: Hewlett-Packard 228E

Non-Volatile Memory: Intel 730 Series SSD 120GB

OS: Windows 8.1 Pro

However, this program will be able to run on most modern computers that support Java 8 since this program will be created using Java 8 so the hardware specifications are rather unimportant. The operating system is unimportant as well since Java programs will work on the popular operating systems such as Windows, Linux, and Mac OS since those operating systems have Java SE support so as long as the user has Java 8 support installed in their system, this program will function as intended on such systems.

The main deliverable class, Client, is the interface between the user and the classes that are used to generate the desired output and actions for the user input. The Client class will take user input and depending on what the user provides, Client will call the appropriate methods on the Inventory object instance that is in the Client Class. By limiting the calls to only the Client class we are able to maintain a balanced and controlled way of editing the list. Another way that we made sure to safely handle the stored data was by using the Singleton design pattern to make sure that we only have one instance of the class ArrayListIterator. After the methods called by Client are executed we will use the Observer pattern and the Iterator pattern to iterate through our list checking if any changes made to the quantity and expiration date of all the medications in stock would meet the conditions necessary to trigger the restock notification. The restock notification is triggered by using the Observer design pattern and notifying the DisplayOutOfStock or DisplayExpiry notification. The verify stock command also calls the update from the Observer display classes.

2. Requirement specifications

The program will keep track of a pharmacies drug inventory, including the amounts of each popular medication they have and the expiration dates.. It will accept sales made by the pharmacy, decreasing inventory, and purchases by the pharmacy, increasing the inventory, as inputs. The program will also output warnings if the pharmacy needs to restock a medication, due to low inventory or soon to expire inventory. The program can also output the entire stock if requested by the client.

Inputs - Drug purchases and sales made by the pharmacy

Outputs - Warnings when running low or stock close to expiration, List of commands, List of current stock, status of current stock.

Menu commands and output: (Input highlighted)

1 - Add drug

    Enter a drug name:

    String

    Enter drug quantity:

    int

        if(invalid quantity) reprompt

    Enter expiration date(Day/Month/Year):

    int/int/int

        if(invalid quantity) reprompt

    Drug registered!

2 - Subtract drug

    Would you like to decrease inventory of a drug or remove an expired drug from list(remove notification)?

    Enter 1 to decrease inventory or Enter 2 to remove expired drug:

    int

user inputs 1

    Enter a drug name or a drug index from the list:

    String

        if(invalid name) reprompt

    int

        if(invalid) reprompt

    Enter drug quantity:

```
    int
        if(invalid quantity) reprompt
    Enter expiration date(only if string for name was provided)(day/month/year):
    int/int/int
user inputs 2
    Enter name of drug:
    string
        if valid name
            if expiration date passed
                remove drug
            else
                don't remove drug
        else
            print error message
```

3 - Display drug inventory

displays the drug inventory showing the name, quantity, expiration, and index for each drug

4 - Verify stock

checks if there is stock close to expiring(within seven days of expiration) and if stock is running low(five or less items in stock)

5 - Change current date

changes the current date(if not used, will default to a default date)

6 - Help

displays the available commands for the program

7-Input

x - quit

exits the program

<u>Program Simulation Example:</u> (Input highlighted)

Welcome to the Pharmacy Inventory Manager!

```
Here are the available commands:
1 - Add drug
2 - Subtract drug
3 - Display drug inventory
4 - Verify stock (Checks for quantity and expiration date)
5 - Change current date
6 - Help
7 - Input from file
 x - quit

Enter a command: 3
There is nothing in the inventory!

Enter a command: 1
Enter a drug name: Tylenol
Enter drug quantity: 50
Enter expiration date(day/month/year): 9/23/2016
Drug registered!

Enter a command: 6
Here are the available commands:
1 - Add drug
2 - Subtract drug
3 - Display drug inventory
4 - Verify stock (Checks for quantity and expiration date)
5 - Change current date
6 - Help
 x - quit
```

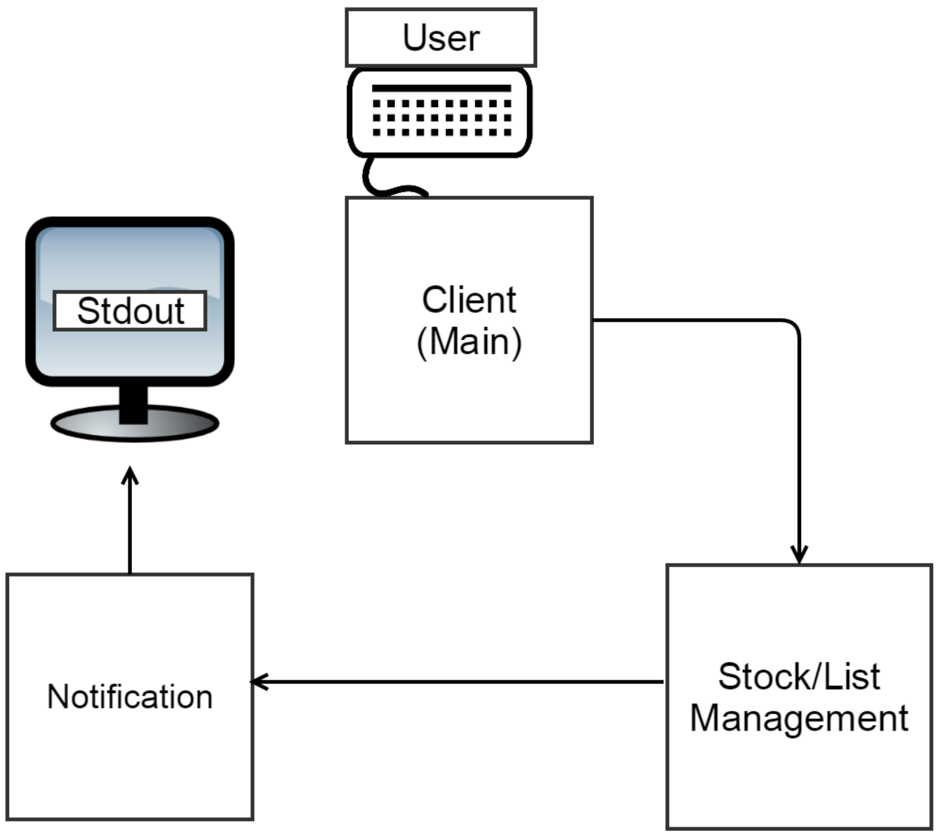
```
Enter a command: 3
Current stock:
ID:     Name:           Quantity:       Expiration Date:
1       Tylenol         50              9/23/2016

Enter a command: 4
Stock is okay!

Enter a command: 7
Enter file name:

Enter a command: x
Exiting....
```

3. Software architecture



- Client contents: Client class
- Stock/list management contents: Inventory, Drug, ArrayListIterator,Iterator interface, Date
- Notification contents: Display class, Observer Interface, Subject Interface.

The component diagram displays the three major components of the stock management program and displays paths between components that communicate and transfer data.
The first component is the Client. The Client component involves all of the methods used to receive user input to add a new medication, adding more stock, making sales (i.e. removing medicine from stock), display a list to check which medicines will expire soon before the notification to restock appears and setting the date and time. The Client communicates with the Stock List component. When it sends new information about a medication it sends it to the Stock List components who directly updates the list of medicines and their values to reflect the

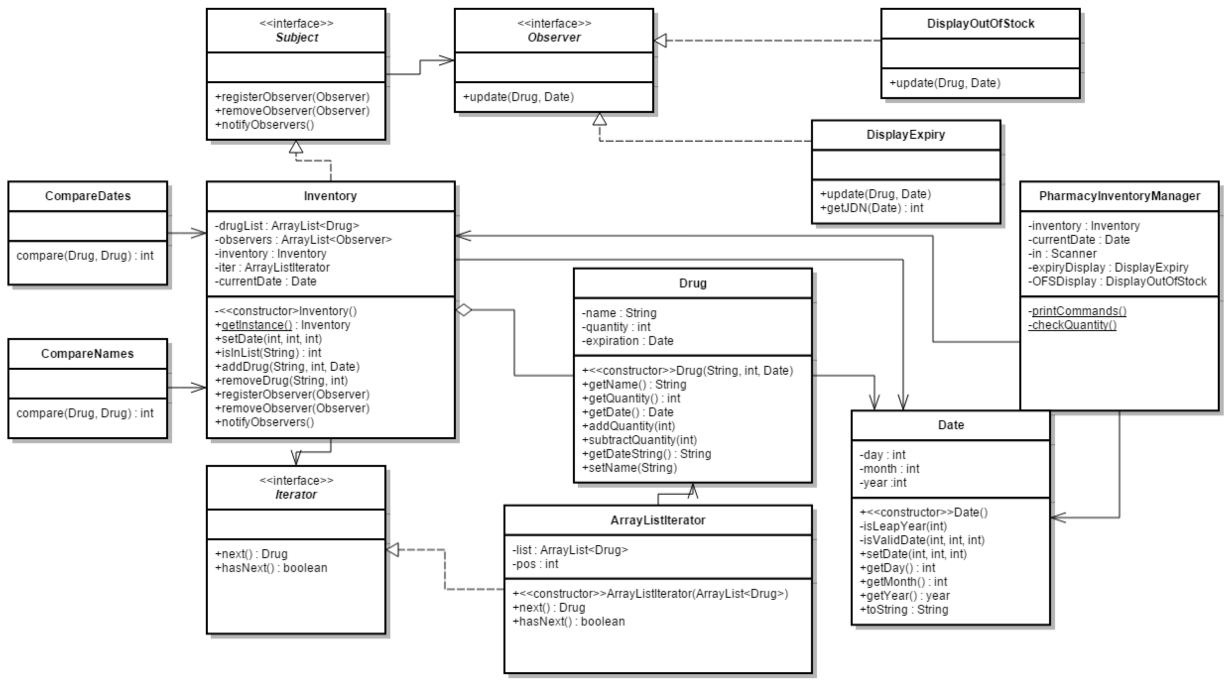
increase/decrease of medicine in stock. The methods that can be called from the Client can be addDrug, removeDrug and setDate. The Client is only used to initiate class methods that will traverse through the entire component diagram calling other method calls along the way to perform the action the User desired.

The second component is the Stock/List Management which is responsible for maintaining the list of medications up to date. This component calls all the methods needed to execute the commands that the User wants such as constructing a new drug. It also interacts with the Notification component sending it up to date information so that the correct display method will be called.

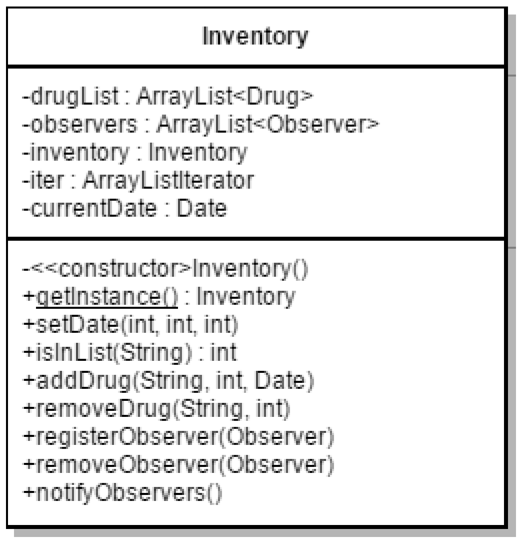
The third component is the Notification which will display the appropriate output based on the current inventory.

4. Detailed design

Class Diagram:



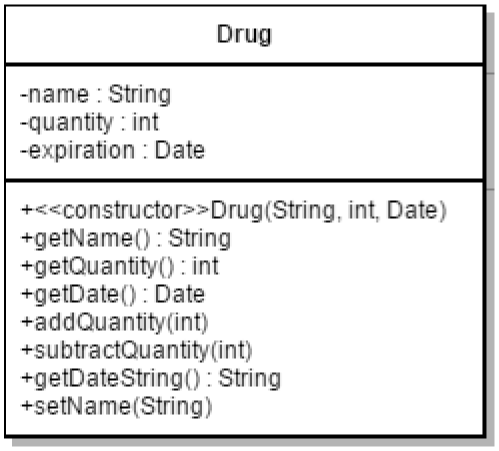
The Pharmacy Inventory Manager manages the output commands and takes in user input for the commands to modify the inventory list and check if stock needs to be replenished. The **singleton design** pattern is used for the Inventory class since only one instance of instance is needed since only one inventory is needed per pharmacy. In addition, the Inventory class is the subject that notifies two display classes, using the **observer pattern**, when the stock is running low(quantity <= 5) and the expiration date close(within 7 days). The Inventory class also uses the **iterator pattern** to traverse an ArrayList of type Drug, the drugs that are in the stock.


Inventory
-drugList : ArrayList<Drug>
-observers : ArrayList<Observer>
-inventory : Inventory
-iter : ArrayListIterator
-currentDate : Date
-<<constructor>Inventory()
+getInstance() : Inventory
+setDate(int, int, int)
+isInList(String) : int
+addDrug(String, int, Date)
+removeDrug(String, int)
+registerObserver(Observer)
+removeObserver(Observer)
+notifyObservers()

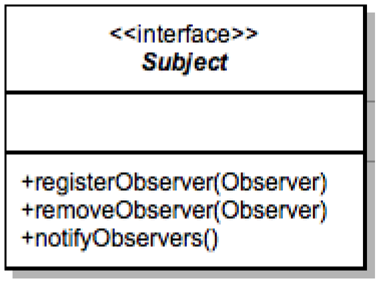
Inventory Class:

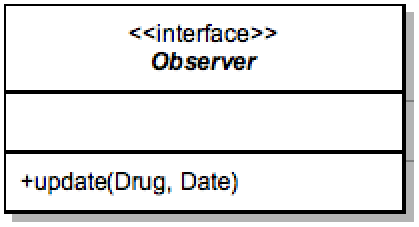
The Inventory class is the primary class for the Pharmacy Inventory Manager. It consists of 5 attributes which are drugList, observers, inventory, iter, and currentDate. The drugList field will contain a list of drugs that are currently available in our system. We will use an arrayList of Drug objects to store the Name, Quantity, and the expiration date of a product. The ArrayList is specifically used because it provides an easy way to create dynamically many elements(elements of type Drug) to the list. The observers method will consist of an arraylist of Observers. The inventory field will hold the actual drug inventory, used for the singleton pattern. The iter is an iterator in which will be used to go through a list of drugs. If we are looking for a particular drug and possibly want to update it we will use the iter to perform the required search. The currentDate field will store the current date. The class also provides methods to access and change these parameters as necessary, as well as all of the methods implemented from the subject interface.

Drug Class:



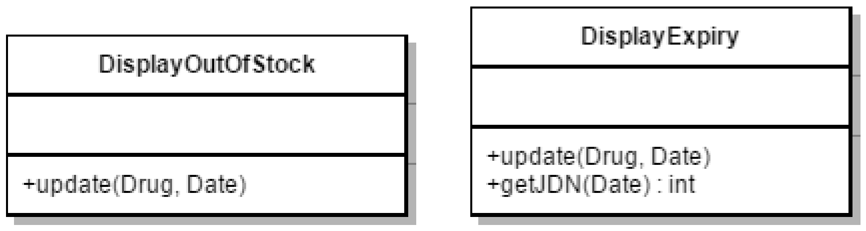
The drug class will keep track of all important information about the stock of a particular drug. This includes the name of the drug, the quantity of the drug, and the expiration date of the drug. The class also has methods that allow clients to access and set the necessary fields. These include a constructor that creates an instance of the drug class with the String name, integer quantity, and Date expiration date given in the parameters. There are also getter methods to access the three fields(name, quantity, and expiration). Finally there is a subtractQuantity(int n) method to remove n from quantity of this drug in stock, and addQuantity(int n) which adds n to the quantity of the drug in stock. subtractQuantity should be called whenever the pharmacy makes a sale, addQuantity should be called only if the pharmacy purchases new stock of the same drug with the same expiration date(checked by client Inventory).





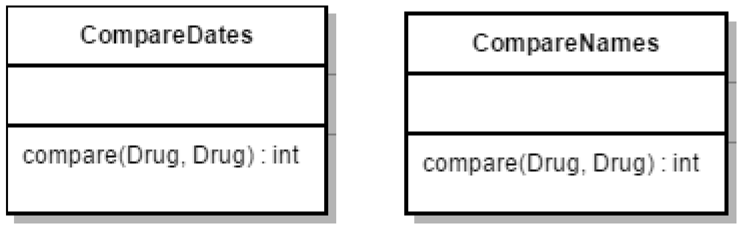
Subject Interface and *Observer* Interface:

Both of these are the standard interfaces for the observer pattern. The Subject interface has methods to register a new observer, remove an observer and notify all observers. The Observer interface has on method update to get updated information from subject. Subject is implemented by our concrete subject, Inventory. Observer is implemented by our concrete observers, DisplayExpiry and DisplayOutOfStock.



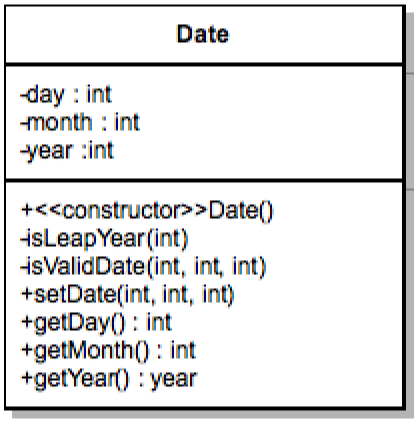
DisplayExpiry Class and DisplayOutOfStock Class:

These are our concrete observer classes. DisplayExpiry will print a warning if any of the drugs in Inventory are going to expire soon. DisplayOutOfStock will print a warning if any of drugs in Inventory are running low on quantity.



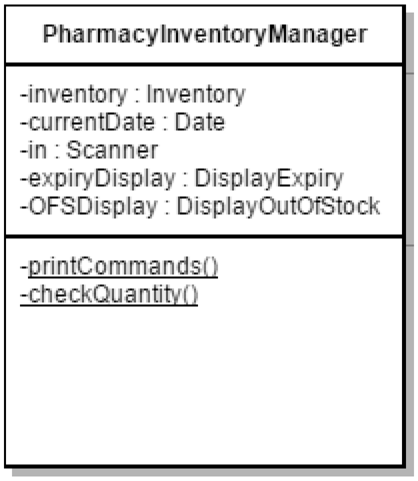
CompareDates and CompareNames Nested Classes of DisplayExpiry:

These are nested classes that compare names and compare dates of two drugs to determine which order of drugs get deleted when there are multiple of the same drug with differing expiry dates.



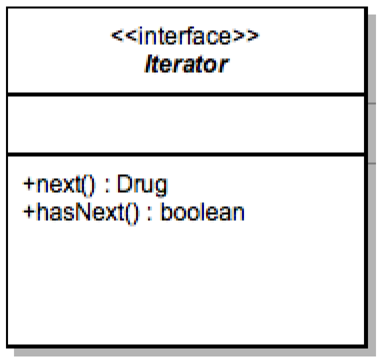
Date Class:

The date class will provide dates throughout the Pharmacy Inventory System. It will determine and also decided whether a given data is a valid date. It will also determine if a given year is a Leap year or not. To return the current date, the date class will utilize getters and setters methods in which would allow the user of the inventory system to check the date of a given drug or simply return the current date.



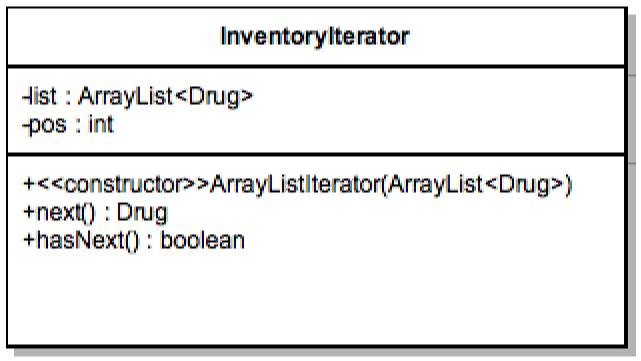
Client Class:

The client class is essentially the face of the Pharmacy Inventory System (PIS). It will provide the tools necessary for users of our software. Upon running the PIS program you will be able to add drugs to the inventory along with basic operations of updating or simply removing a drug from the inventory. The client class will consists of 3 methods, Inventory, currentDate, and StdIn.



<u>*Iterator* Interface:</u>

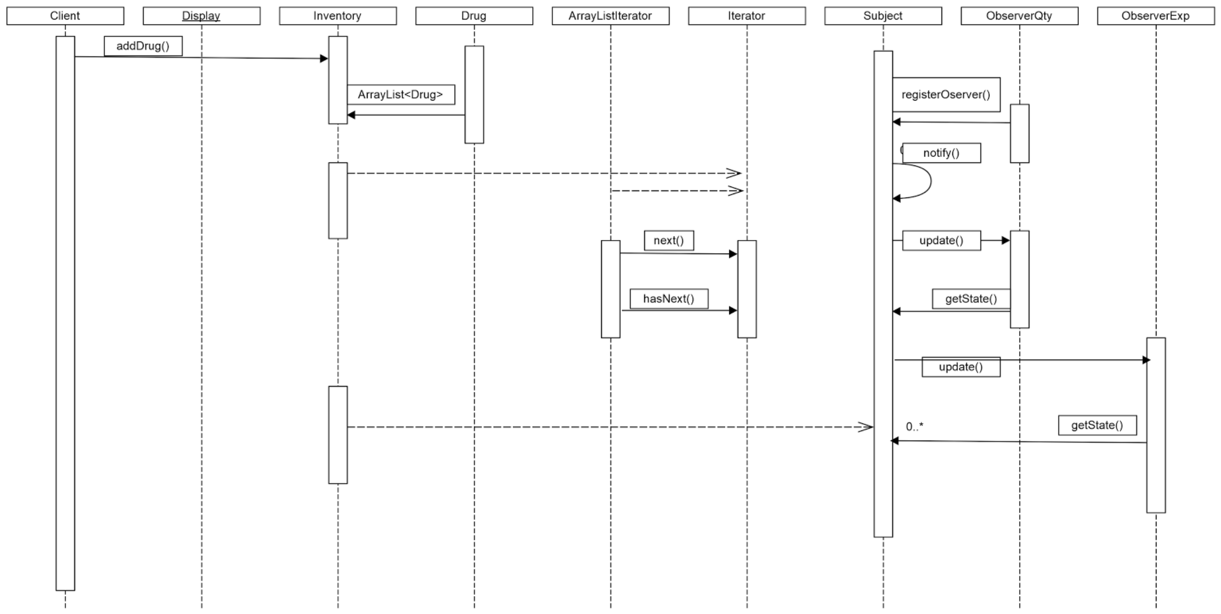
The *Iterator* interface provides a way to iterate through an ArrayList of type Drug(ArrayList<Drug>). Method next() returns the next drug in the ArrayList. Method hasNext() return a boolean indicating whether there is another element after the current element in the ArrayList. The *Iterator* interface is intended to traverse the ArrayList by calling the next() method until the hasNext() method returns false or until the desired Drug element in the ArrayList.



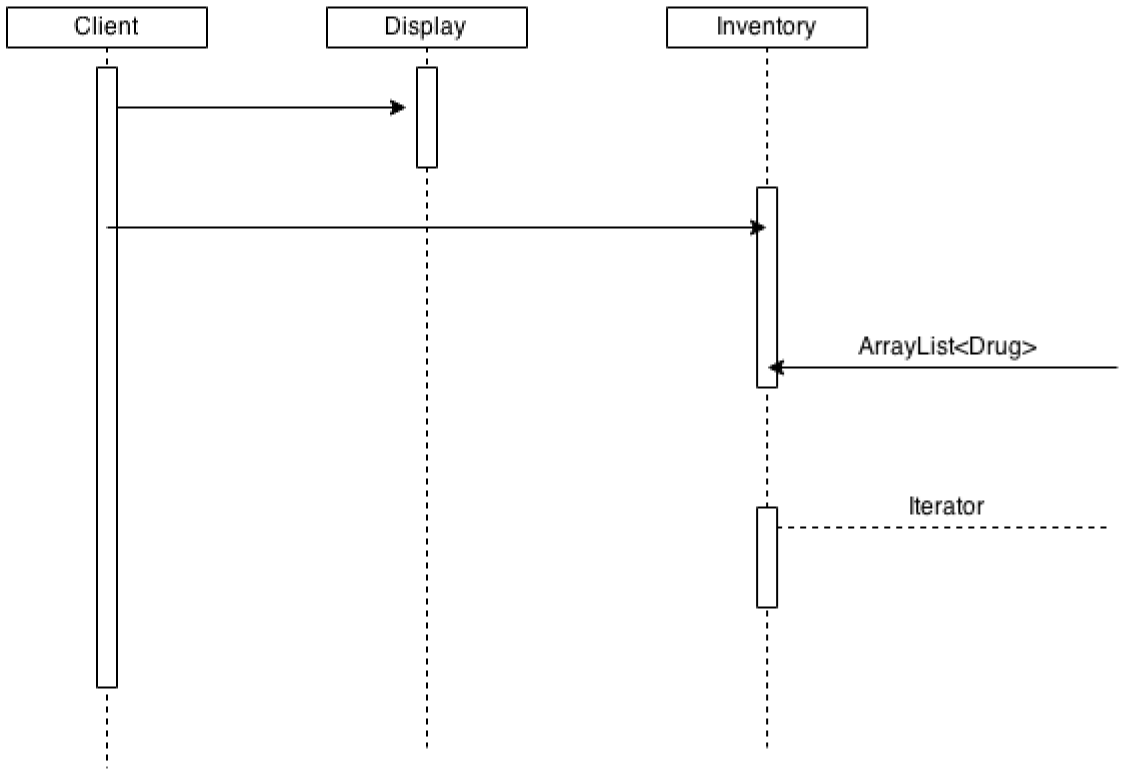
<u>Inventory Iterator Class:</u>

The InventoryIterator class provides a way to iterate through an ArrayList of type Drug. The pos field provides a way to keep track of the current position while traversing the ArrayList.

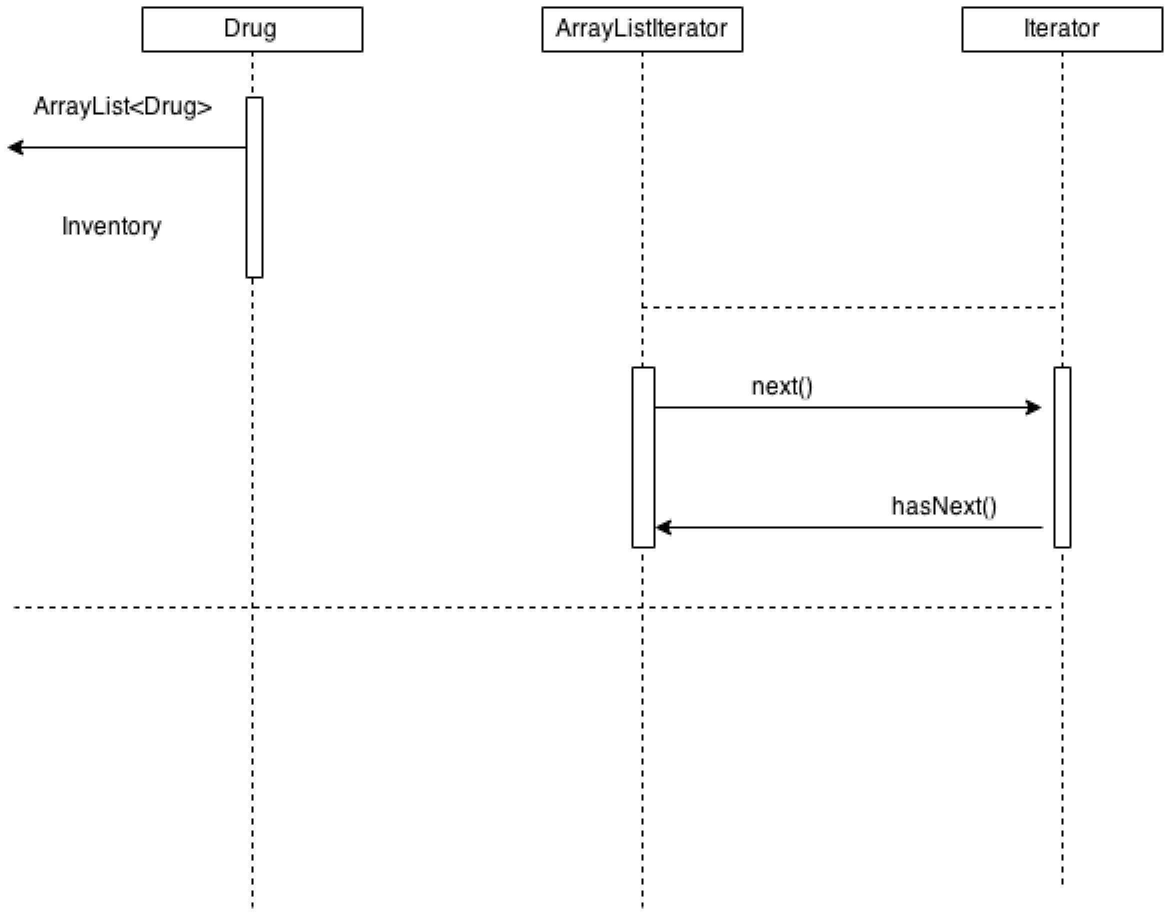
The list field creates a field for the list for it to hold the ArrayList<Drug> object passed through the constructor. The Iterator methods, next() and hasNext(), are implemented here.



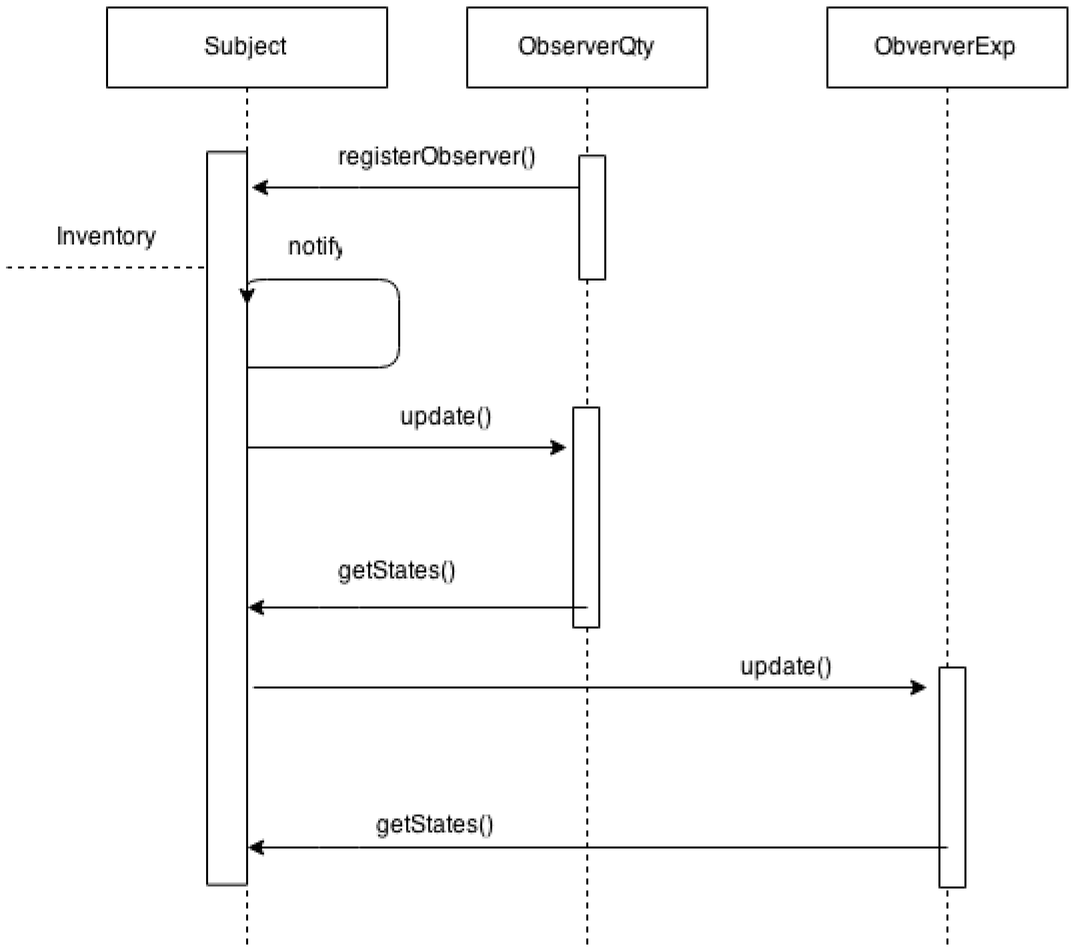
The above diagram shows roughly the step-by-step sequence of the program. Below are sections of this diagram explained individually.



The Client creates an instance of the classes Inventory, Display, and Date. Display accesses the Inventory class and gets the necessary information in order to display them.
Inventory implements the interfaces Iterator and Subject. Inventory’s responsibility is to create an arraylist of Drug objects and calls on the Subject interface in order to identify changes.



ArrayListIteratory implements the interface Iterator which iterates through the list of Drug objects whenever called by the Observer. ArrayListIteratory uses the methods next() and hasNext() in order to iterate the array list.



The Inventory class implements the interface Subject which notifies the concrete observers ObserveQty and ObserveExp. The two observers call the update() method whenever there is either a shortage of a type of drug or a drug is about to expire. The updated information returned by the observers is printed via the Display class.

5. Implementation notes

The interface right now utilizes a command line interface for output and input. We plan on adding a graphical user interface for ease of use and aesthetic purposes. Of course, a graphical user interface may complicate things so we have designed our current program to use a command line interface for the meantime.

<u>What everybody did:</u>

Everyone:

- Test code
- Make small changes and updates to code overall
- Work on final design write up
- Keep other members updated with new changes
- Part 1 hardware specifications
- Coming up general design of the entire program

Thomas Misiaszek:

- Part 1
- Part 4 -UML Class Diagram
- Part 2 - input and program output example and command sequence
- CompareDates and CompareNames explanations in design
- sample code

Abraheem:

- Part 1: Added the project explanation example from the PDF.
- Included my hardware and software information
- Part 4: Sequence diagram:
- Designed the sequence diagrams
- Wrote detailed explanations of each section of the sequence diagrams

- Wrote majority of main and removeDrug() in Inventory class
- Wrote Comparator methods for sorting
- displayInventory() formatting

Daniel:

- Architecture design/component diagrams
- Part 1 design pattern section
- Part 3
- Commented Date,DrugSubjectArrayList Iterator classes
- Wrote removeExpiredDrug and getDay methods

Marcell:

- Design Pattern setup and initial phase design
- Inventory class, method, and field explanation
- Date class, method, and field explanation
- Client field class, method, and field explanation

Joseph:

- Part 2 - description, inputs, outputs
- Subject interface, method, and field explanation
- Observer interface, method, and field explanation
- DisplayOutOfStock class, method, and field explanation
- DisplayExpiry class, method, and field explanation
- Drug class, method, and field explanation
- Wrote DisplayExpiry class
- Commented Inventory class
- Made changes to RemoveDrug method in Inventory, Date class constructor, and the main method.

<u>Code Snippets: Main</u>

```
package pharmacyinventorymanager;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;
import java.util.StringTokenizer;

/**
 * The pharmacy inventory manager that keeps track of an inventory of drugs.
 * Takes in user input to modify the stock.
 * @author Tom & Abraheem
 */
public class PharmacyInventoryManager
{
        /**
         * Prints help menu containing all valid commands
         *
         */
    private static void printCommands()
    {
        System.out.println("Please enter a command:");
        System.out.println("1 - Add drug");
        System.out.println("2 - Subtract drug");
        System.out.println("3 - Display drug inventory");
        System.out.println("4 - Verify stock");
        System.out.println("5 - Change current date");
        System.out.println("6 - Help");
        System.out.println("7 - Input from file");
        System.out.println("x - quit");
    }

    /**
     * Checks quantity to make sure it's not less than 1
     *
     * @param qty                   Drug quantity from user input
     */
    private static void checkQuantity(int qty){
            if(qty < 1){
                    System.out.println("*** Error: Quantity must be larger than 0 ***");
            }
    }

    /**
     * Main method used by client.
     * Displays menu and takes user input.
     */
    public static void main(String args[])
    {
        Inventory inventory = Inventory.getInstance();
        DisplayExpiry expiryDisplay = new DisplayExpiry();
        DisplayOutOfStock OFSDisplay = new DisplayOutOfStock();
        //Date currentDate =  new Date();
        int day, month, year;
        String name, dateString, fileName;
        int quantity;
        Date date;
        StringTokenizer token;

        inventory.registerObserver(expiryDisplay);
        inventory.registerObserver(OFSDisplay);

        Scanner in = new Scanner(System.in);
        System.out.println("Welcome to the Pharmacy Inventory Manager!");
        printCommands();

        char input;
        String line;
        while(true)
        {
           System.out.print("Command: ");
           line = in.next();
           input = line.charAt(0);

           switch (input) {
           case '1': // Add drug
              date = new Date();

              System.out.println("Enter name of drug: ");
              in.nextLine(); // Flush out the newline character
              name = in.nextLine(); // Take name even if it contains more than one word like
(Tylenol PM)

              do{ // Prompt for quantity until input is correct
                    System.out.println("Enter amount of drug to add: ");
                    quantity = in.nextInt();
                    checkQuantity(quantity); // Make sure quantity is a valid number
              } while(quantity < 1);

              do{ // Prompt for date until input is correct
                       System.out.println("Enter expiry date(dd/mm/yyyy): ");
                       dateString = in.next(); // The entire date string

                       token = new StringTokenizer(dateString); // Splits dateString into tokens
                       day = Integer.parseInt(token.nextToken("/"));
```

```
                month = Integer.parseInt(token.nextToken("/"));
                year = Integer.parseInt(token.nextToken("/"));

            } while( ! date.setDate(day, month, year) ); // Checks date. If valid, set date, else
re-prompt

            inventory.addDrug(name, quantity, date); // Creates a drug object
            inventory.notifyObservers();

            break;
          case '2': // Subtract drug (sell drug)
            System.out.print("Enter name of drug: ");
            in.nextLine(); // Flush out the newline character
            name = in.nextLine(); // Take name even if it contains more than one word
            //System.out.println("Name: " + name); // DEBUG
            System.out.print("Enter amount of drug to subtract: ");
            quantity = in.nextInt();

            inventory.removeDrug(name, quantity);
            inventory.notifyObservers();

            break;
          case '3': // Display inventory
            inventory.displayInventory();

            break;
          case '4': // Check for low quantity or expiring drugs (This is the observer part)
            inventory.notifyObservers();

            break;
          case '5': // Change current system date
            System.out.println("Enter current date(dd/mm/yyyy): ");
            dateString = in.next(); // The entire date string
            token = new StringTokenizer(dateString, "/"); // Splits dateString into tokens
            inventory.setDate(   Integer.parseInt(token.nextToken()),
                                 Integer.parseInt(token.nextToken()),
                                 Integer.parseInt(token.nextToken()));

            break;
          case '6': // Print help menu
            printCommands();

            break;
          case '7': // Take input from a file
              System.out.print("Enter file name: ");
```

```
        fileName = in.next(); // Gets file name from stdin

        try {
            FileReader fileReader = new FileReader(fileName); // Create file reader
            Scanner buff = new Scanner(fileReader); // Create scanner to traverse the
    file
            // While not end of file, read and add all contents of file to inventory
                while( buff.hasNext() ){ // While file is not empty
                        date = new Date();
                        name = buff.nextLine();
                        quantity = buff.nextInt();

                        dateString = buff.next();
             token = new StringTokenizer(dateString);
             day = Integer.parseInt(token.nextToken("/"));
             month = Integer.parseInt(token.nextToken("/"));
             year = Integer.parseInt(token.nextToken("/"));
             date.setDate(day, month, year);

             inventory.addDrug(name, quantity, date);
             if( ! buff.hasNext() ){ // Exits the while loop before last flush. It would
crash otherwise.
                break;
             }
             buff.nextLine(); // Flush out the newline character
                    }
                    buff.close();
                }
                catch (FileNotFoundException ex) {
                    System.out.println("Unable to open file '" + fileName +
"'");
                }

        break;
    case 'x': // Exit program
        in.close();
        System.out.println("Exiting Program...");
     System.exit(0);

    default: // Invalid user input
     System.out.println("Invalid command!");
     printCommands();
     break;
    } // SWITCH
  } // WHILE
```

```
    } // MAIN
} // CLASS
```

Design issues:
The current design does not implement taking input from a file. Without taking input from a file, it becomes difficult to debug because the user must enter the drug data individually. However, this feature will be added to a later version of this design.