Chapter 2

Team 3: Meg, Mitch, Itzayana

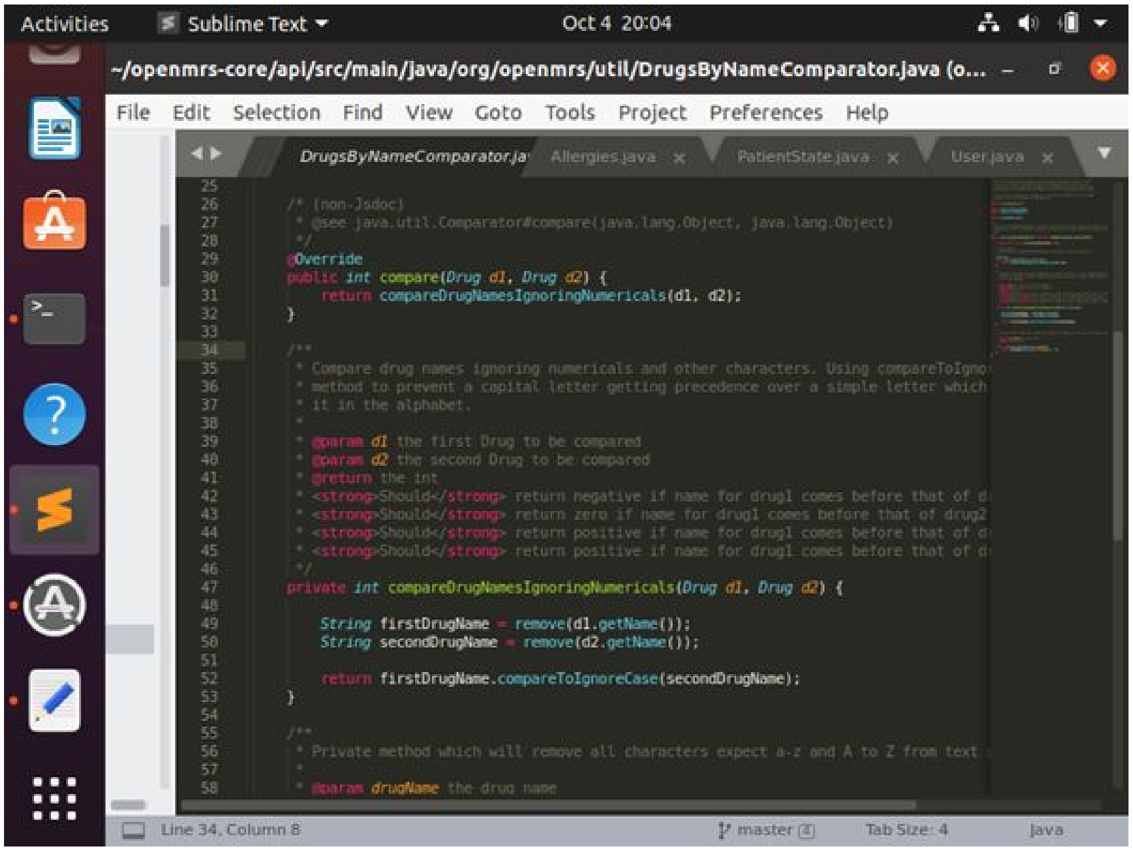
CSCI 362 Project: OpenMrs

Method Selection

OpenMRS had plenty of files for our team to search through to find 5 methods to test. The sheer number of Java files was very overwhelming at first and finding methods fit for testing proved to be difficult. Since OpenMrs is more or less a Database that stores medical information it is mostly made up of getter and setter methods, but as we continued to parse through the different classes, we eventually found several methods that we considered fit for testing. As of now we are looking at more than the required amount of methods since a lot of these classes have several dependencies, and we are not sure what will be manageable and what will not be. For this Deliverable we will be looking at the compare() method from the DrugsByNameComparator class.

Method

The compare() method is DrugsByNameComparitor takes two objects of type Drug and compares their names by using the compare interface. An image of the code is below. This method should take two objects of type Drug, get each one's name, remove any numbers from each name, and then compare them alphabetically. If the first drug should come before the second, then the method should return a positive integer. If the second drugs should come first, then it should return a negative integer. If both drugs have the same name, then it should return a zero. Since this method is more or less just comparing two strings, our team may tweak the code so that compare() takes two strings instead of two Drug objects for ease of testing and to avoid any unnecessary dependencies.



```java
    /* (non-Jsdoc)
     * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     */
    @Override
    public int compare(Drug d1, Drug d2) {
        return compareDrugNamesIgnoringNumericals(d1, d2);
    }

    /**
     * Compare drug names ignoring numericals and other characters. Using compareToIgno
     * method to prevent a capital letter getting precedence over a simple letter which
     * it in the alphabet.
     *
     * @param d1 the first Drug to be compared
     * @param d2 the second Drug to be compared
     * @return the int
     * <strong>Should</strong> return negative if name for drug1 comes before that of d
     * <strong>Should</strong> return zero if name for drug1 comes before that of drug2
     * <strong>Should</strong> return positive if name for drug1 comes before that of d
     * <strong>Should</strong> return positive if name for drug1 comes before that of d
     */
    private int compareDrugNamesIgnoringNumericals(Drug d1, Drug d2) {

        String firstDrugName = remove(d1.getName());
        String secondDrugName = remove(d2.getName());

        return firstDrugName.compareToIgnoreCase(secondDrugName);
    }

    /**
     * Private method which will remove all characters expect a-z and A to Z from text
     *
     * @param drugName the drug name
```

Figure 1: Method 1

Test Plan

- **Testing Process**
  - We will create an automated testing framework that will run through the 25 test cases our group has created for OpenMRS. We will be testing five different methods from OpenMRS, each method will have 5 different test cases. After the testing is complete the framework will parse the results, and save them in an HTML file.
- **Requirements Traceability**
  - Each test case will have a unique test ID and unique requirements, which will allow for easy traceability of all the test cases. The requirements for each method are presented in the table below.
- **Items Tested**
  - As stated before, we have selected more methods than needed to ensure we can find a method that doesn't have too many dependencies or that we can tweak enough to keep the logic the same but make it easier to test. Below are the classes and methods we plan to test.

Class	**Method**	**Requirements**
Allergies	Add(Allergy a1)	adds an object of Type Allergy to a list

DateUtil	truncateToSeconds(Date date)	truncates the date to the second
DrugByNameComparator	compare(Drug d1, Drug d2)	Compares the names of two drugs
DrugOrder	hasSameOrderableAs(DrugOrder d1)	Returns true if the drugs have they say orderable
Graph	topographicalSort()	Sorts the graph
OrderUtil	isType(OrderType o1, OrderType o2)	Checks to see if the two orders have the same type
PateintState	/compareTo(PatientState p)	Compares the states of two patients
PersonMergeLogData	computeHashTable()	Creates a hash table of people
UserByNameComparator	compare(User u1, User u1)	Compare the name of two users

- **Testing Schedule**
  - Deliverable 3: Automated Testing Framework. (11/05/20)
    - Extract methods and classes to be tested
    - Create a script that can automatically run several test cases
    - Create 5-10 test cases a week
  - Deliverable 4: Run 25 Test Cases and Pipe them to HTML File (11/17/20)
  - Deliverable 5: Test deliberate errors in the Code (11/24/20)
  - Final Report and Presentation (12/03/20)

- **Test recording procedures**
  - After our driver runs all tests, the output will be sent to an HTML file and displayed in a web browser.

- **Hardware and software requirements**
    - Hardware: Computer
    - Software: VirtualBox, Linux, Terminal, Java, Github, and Git
- **Constraints**
    - Meeting Schedule: as a team we have scheduling constraints. Each team member has a variable work and school schedule, which limits our availability to meet.
- **Unit tests**
    - For testing our methods, we have not yet created an automated testing script, but we have a few preliminary test cases in our Github Repository[1]. The basic information in them can be found in the graph below.
    - <u>TestCase Key</u>
        - TestID
        - Class name
        - Requirements
        - Method name
        - Summary
        - Inputs
        - Expected outputs
        - Driver

    - <u>TestCase 1</u>
        - 01
        - DrugsByNameComparator
        - This class will compare two strings and put them in alphabetical order
        - compare(String d1, String d2)
        - This test will test the compare() when d1 comes before d2 alphabetically
        - Allegra Dayquil
        - 3
        - Driver
    - <u>TestCase 2</u>
        - 02

1

        - DrugsByNameComparator

        - This class will compare two strings and put them in alphabetical order
        - compare(String d1, String d2)
        - This test will test the compare() when d2 comes before d1 alphabetically
        - Dayquil Allegra
        - -3
        - Driver

    - <u>TestCase 3</u>
        - 03
        - DrugsByNameComparator
        - This class will compare two strings and put them in alphabetical order
        - compare(String d1, String d2)
        - This test will test the compare() when d1 and d2 are the same
        - Allegra Allegra
        - 0
        - Driver

    - <u>TestCase 4</u>
        - 04
        - DrugsByNameComparator
        - This class will compare two strings and put them in alphabetical order
        - compare(String d1, String d2)
        - This test will test the compare() when d1 comes before d2 alphabetically, but d2 has a number in it that would make it come first
        - 2Allegra 1Dayquil
        - -3
        - Driver

    - <u>TestCase 5</u>
        - 05
        - DrugsByNameComparator
        - This class will compare two strings and put them in alphabetical order
        - compare(String d1, String d2)
        - This test will test the compare() when d1 is a string and d2 is null
        - Allegra null
        - Null pointer exception
        - Driver

https://github.com/csci-362-02-2020/Team3/tree/master/testCases

Final Thoughts and Evaluation Report

We have been learning a lot as a team finding good testable methods has proven difficult since most of the methods in OpenMRS are setter and getter methods. We also did a team exercise to help learn how to compile a java program from the command line. We are still running into some errors when we try to run OpenMRS, namely with packages and other strange dependencies, hopefully, we can overcome that and continue with our testing.