OpenMRS

Team6
Jacob Ballou
Alan Arsiniega
Cameron Reuschel

Topics

Topics of Discussion:

Background - What is OpenMRS?

OpenMRS is a patient-based medical record system focusing on giving providers a free customizable electronic medical record system (EMR).

It allows medical organizations to track patients, their treatments and medications, inventory, dosages, and other necessary information.

OpenMRS is a Java-based deployable webapp that utilizes a database and allows for widespread use among an organization

Background - Why We Chose OpenMRS

Reasons why we chose OpenMRS for our testing framework:

1. Programmed in Java, a popular object-oriented language
2. Well organized and documented repository and classes
3. Modular and (fairly) understandable components and dependencies
4. Utilizes Apache Maven to compile
5. Important piece of software for many medical centers around the world

Introduction and Analysis

Our first goals were to get OpenMRS in a working state, and understand how it worked

From there, we would be able to isolate parts of the program and form tests of the software

Initial Challenges:

- Which version of OpenMRS to use (2.20)
- How to compile with Apache Maven
- The structure of the components and the interdependencies
- What components to focus testing on

Introduction and Analysis

We analyzed the different parts of OpenMRS and chose components from there that would fit our testing framework.

We chose to use the OpenMRS API, and test different components within it.

Components chosen:

- DoubleRange.java (part of OpenMRS.util)

Preparation

We took several steps to prepare for programming the script and the accompanying drivers. A plan was devised:

- Draft a template for the test cases
- Prototype a script that can read a file, and locate the dependencies to test the method
- Develop our test cases
- Create drivers to utilize with the test cases
- Refine the script to work efficiently with all test cases
- Modify the script to produce a report in html to the browser
- Check for any bugs within the final testing framework

Assumptions and Initial Challenges

Initial challenges with development

- The complexity of bash with file reading and input parsing
- Getting drivers to compile when outside their intended directory
- Managing output to compare with the oracle
- The complexity of our drivers and test cases

Assumptions made before development

- Bash would not be difficult
- Importing the necessary Java dependencies would be easy
- What our test cases should actually focus on (ie. the scope of our testing)

Requirements Being Tested

Two classes from the OpenMRS API are being tested.

DoubleRange.java is a Java class for an object called DoubleRange, which is a number (double) range used for dosages and other medical ranges.

- DoubleRange(double, double); //inclusive lower, exclusive upper

Result.java is a Java class file for an object called Result, which contains a list of numbers (double) created from an operation from the webapp

- Result(double[]);

Requirements Being Tested

We are testing 4 methods for functionality.

From the DoubleRange.java component of openmrs.util:

- contains() - checks if a number resides within a Double Range
  - 5 test cases
- compare() - compares two Double Range to see which has the largest (positive) range
  - 10 test cases

From the Result.java

- gt() - returns all results greater than a given value
  - 5 test cases
- contains() - checks if a number is within the array of numbers in Result
  - 5 test cases

Building Test Cases/Drivers - Concept

Test Cases:

- Built based on providing one test input per case
- Focused on simplicity; testing one method at a time
- Provide information to allow drivers to be compiled outside OpenMRS

Drivers:

- Implement the object of the component class
- Utilize the objects constructor minimalistically (only providing necessary information for the method)
- Output the result of the test to the console (allows the script to grab it)

Test Case Format

The format of our test cases go as following:

- Name of the test
- Requirement of the method
- Component or package
- Method signature
- Java Driver in TestCaseExecutables
- Classpath variable
- Test case input
- Input Representation

Scripting - Design

The master script of our testing framework is ‘runAllTests.sh’, a bash script. The design goes as follows:

Preparation Phase:

+ Clears the /temp directory
+ Reads all available test cases from the /testCases directory
+ Calls the reportFormat.sh script to create and format the test report

Load Phase:

+ Loads individual test cases and retrieves information to run the test cases
+ Finds the corresponding driver and compiles it

Test Phase:

+ Executes the java driver on the cmd line along with the test case input
+ Retrieves the driver output
+ Compares the output to the corresponding oracle

Scripting - Design

Write Phase:

+ Writes the result of the comparison with the oracle to the report
+ Also adds a row to the table for the individual test case

Repetition Phase:

+ Repeat the Load, Test, and Write phases for all available test cases

Finalization Phase:

+ Exit the loop
+ Complete the testReport.html file
+ Display the test report in the browser

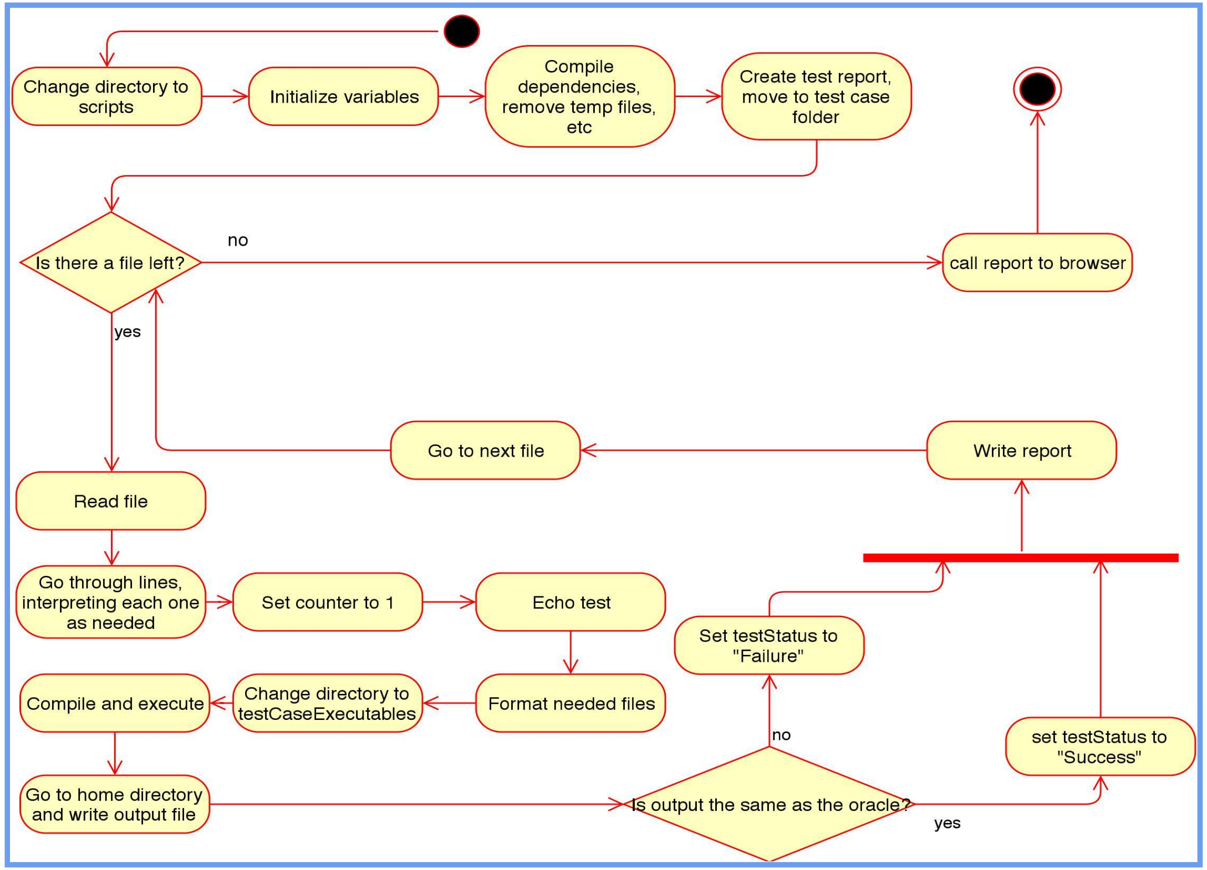
End Script

Scripting - reportFormat.html

This script creates the test report by:

+ Creating the file ‘testReport.html’ in the reports directory
+ Creating the HTML header
+ Formatting the table to display the test cases and their results
+ Formatting the columns of the table
+ Once finished, it returns control to the master script

Scripting Framework



Change directory to scripts
Initialize variables
Compile dependencies, remove temp files, etc
Create test report, move to test case folder
Is there a file left?
no
call report to browser
yes
Go to next file
Write report
Read file
Go through lines, interpreting each one as needed
Set counter to 1
Echo test
Set testStatus to "Failure"
Compile and execute
Change directory to testCaseExecutables
Format needed files
no
set testStatus to "Success"
Go to home directory and write output file
Is output the same as the oracle?
yes

DoubleRange class

DoubleRange - The Ends

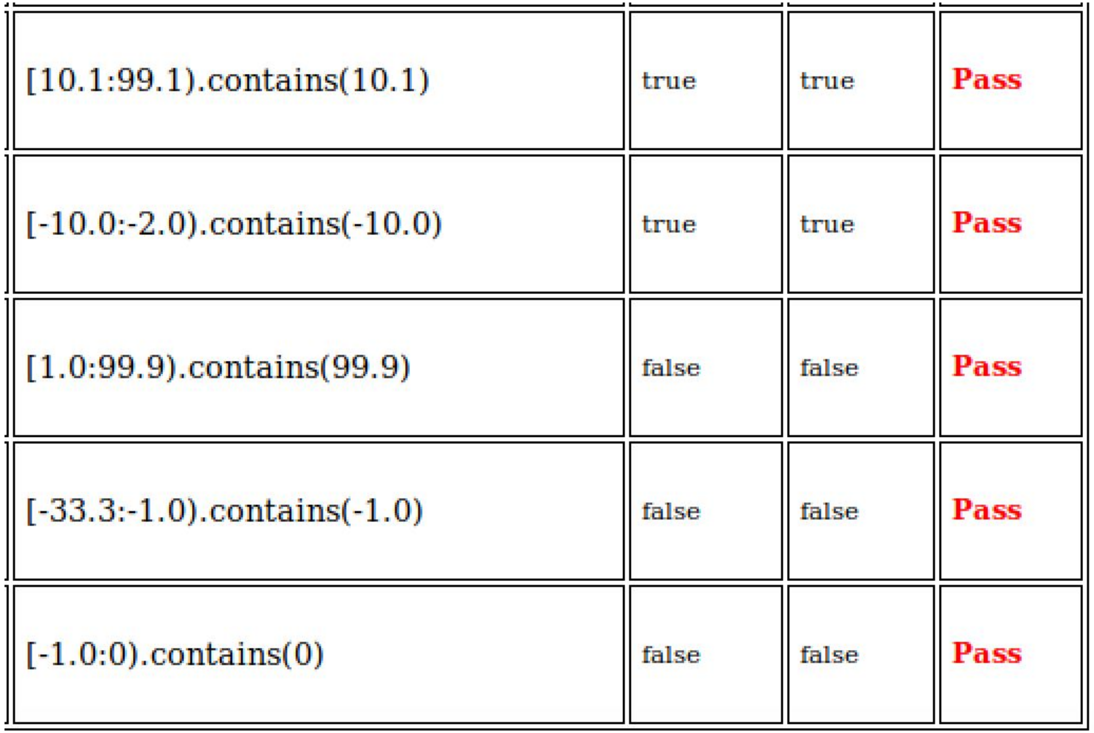
[- Low end is *closed* or *inclusive*

) - High end is *open* or *exclusive*

[double : double) - DoubleRange object

Fault Injection

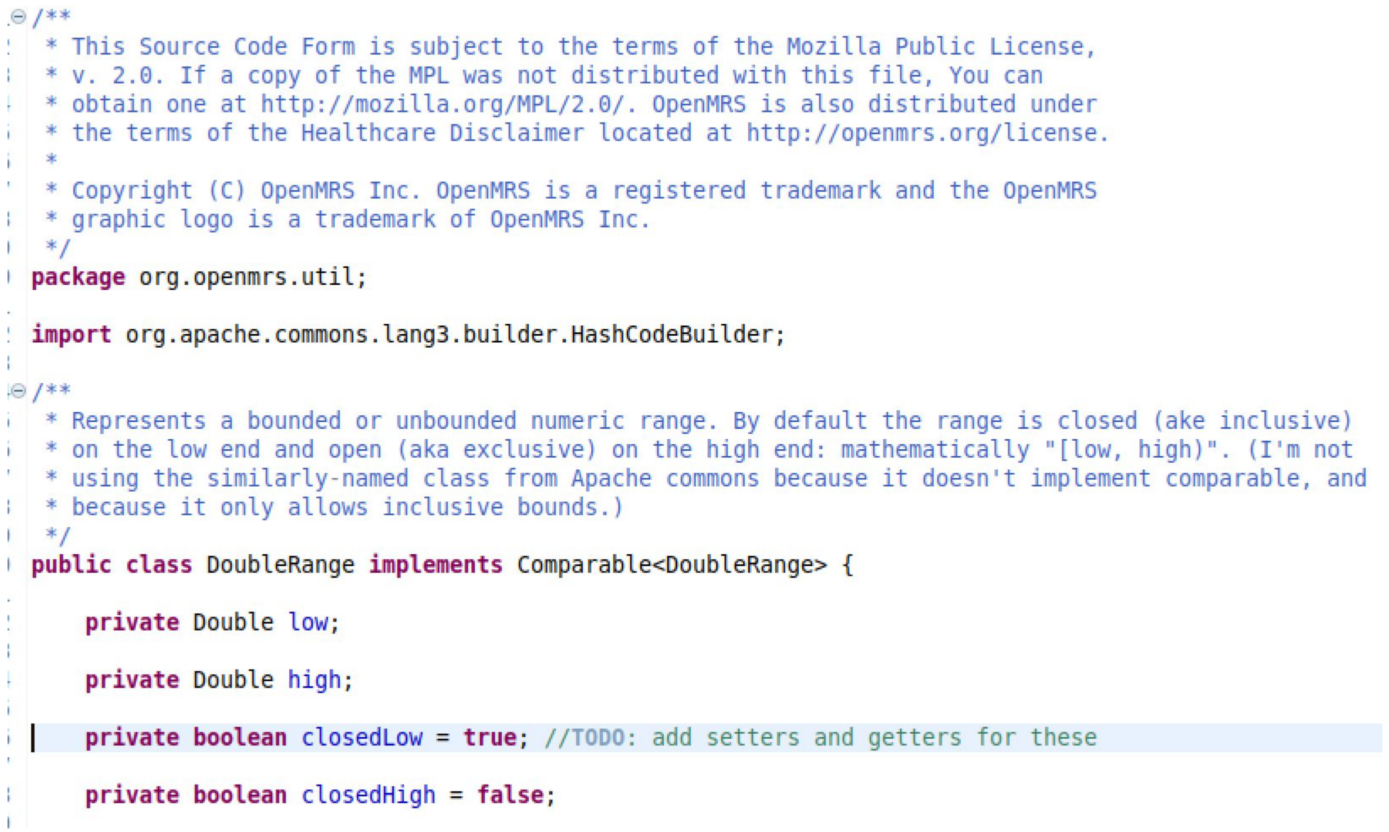
Tests 11 - 15



[10.1:99.1).contains(10.1)	true	true	Pass
[-10.0:-2.0).contains(-10.0)	true	true	Pass
[1.0:99.9).contains(99.9)	false	false	Pass
[-33.3:-1.0).contains(-1.0)	false	false	Pass
[-1.0:0).contains(0)	false	false	Pass

Show Testing Framework

Fault Injection



```java
/**
 * This Source Code Form is subject to the terms of the Mozilla Public License,
 * v. 2.0. If a copy of the MPL was not distributed with this file, You can
 * obtain one at http://mozilla.org/MPL/2.0/. OpenMRS is also distributed under
 * the terms of the Healthcare Disclaimer located at http://openmrs.org/license.
 *
 * Copyright (C) OpenMRS Inc. OpenMRS is a registered trademark and the OpenMRS
 * graphic logo is a trademark of OpenMRS Inc.
 */
package org.openmrs.util;

import org.apache.commons.lang3.builder.HashCodeBuilder;

/**
 * Represents a bounded or unbounded numeric range. By default the range is closed (ake inclusive)
 * on the low end and open (aka exclusive) on the high end: mathematically "[low, high)". (I'm not
 * using the similarly-named class from Apache commons because it doesn't implement comparable, and
 * because it only allows inclusive bounds.)
 */
public class DoubleRange implements Comparable<DoubleRange> {

	private Double low;

	private Double high;

	private boolean closedLow = true; //TODO: add setters and getters for these

	private boolean closedHigh = false;
```

Lessons Learned

- Talk to other classmates early and often! Can save you time or provide valuable insights.

- Get to know your teammates learning styles. Sometimes slack gets misinterpreted and probably better to meet in person or web meeting.

- Always Think Ahead. Testing faster becomes critical later.

Questions