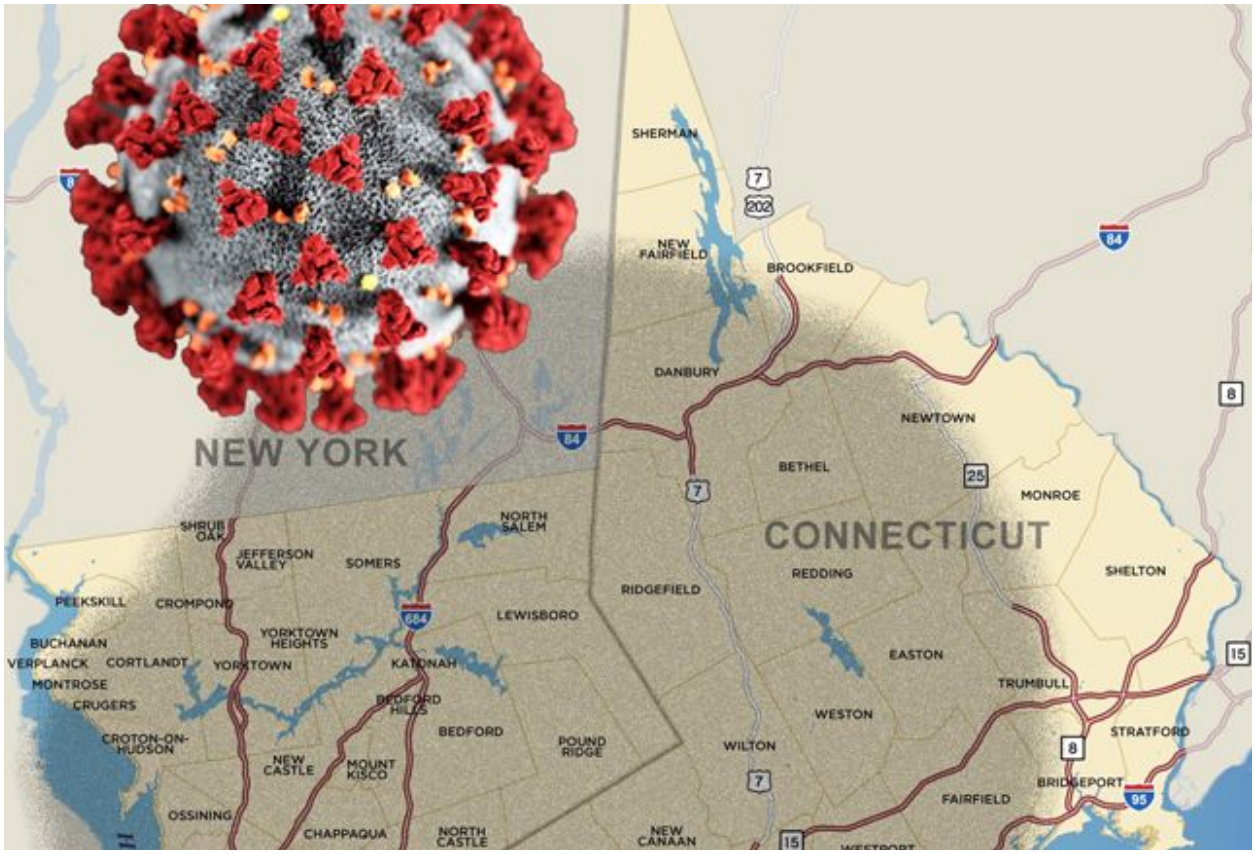
CS506 Software Testing and Quality Assurance

Corona Virus Report

Group 10 Graduate Project



Introduction

This is an open source project that was started by a famous Java Instructor by the YouTube handle “Java Brains.”

You can watch the video here:
https://www.youtube.com/watch?v=8hjNG9GZGnQ&t=1018s

Group 10 team members on this project include John Liu and Nana Ahiabli. We implemented an application and identified four test coverages for this open source project on Coronavirus Reporting.

Source of Software Under Test

Data Sources (refreshed on regular basis by the data owners):
https://covidtracking.com/api/v1/states/daily.csv
https://www.gstatic.com/covid19/mobility/Global_Mobility_Report.csv

Original Source Code from Java Brains:
https://github.com/koushikkothagal/coronavirus-tracker

Expanded Version of the Original Project (our code here - still in development):
https://github.com/stoic-llama/report/tree/johndev

Developer: John Liu
Technical Writers: Nana A., John Liu

Purpose, Services, and Functionalities of Software

Purpose

The original project was launched on Feb 28, 2020 as both an educational material for one video on building a Spring Boot application from scratch, as well as an awareness for Coronavirus and spreading important information to the people.

Services: Providing Latest Metrics For Understanding Covid-19 Impact To Us

This Spring Boot project has since been expanded by this team to use more up-to-date links than the original project. The updated links include Google Mobility statistics to track social distancing and Covid Tracking Project link from a group of citizen activists that track death, ICU, hospitalization etc. statistics.

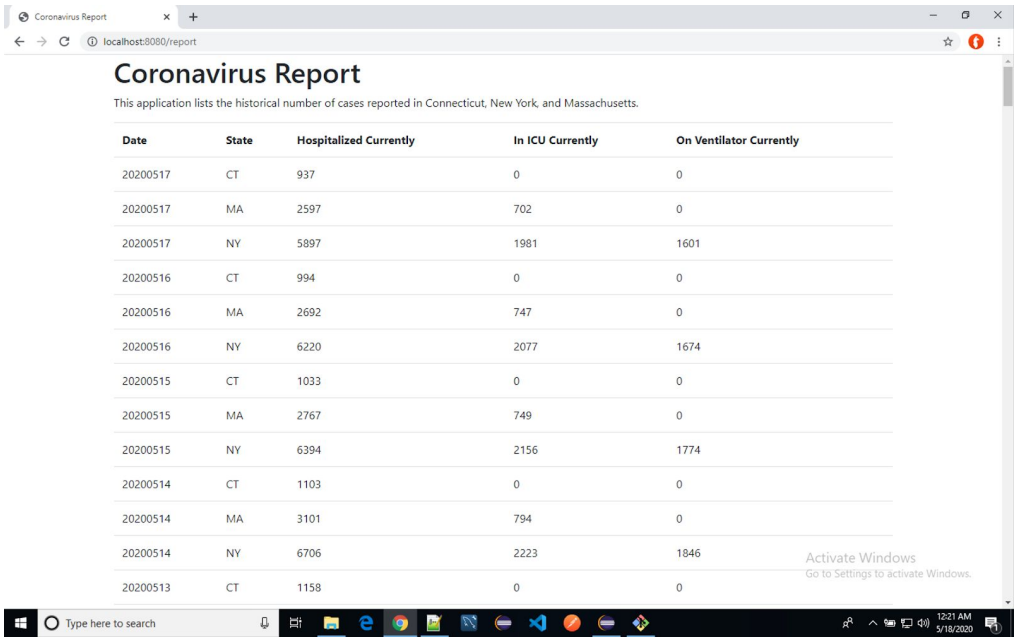
Functionalities

The expanded Spring Boot project is very much evolving and not in the final state. However, as we stand today you can run the application locally on your own machine (it is not hosted on the internet yet), and access three functionalities:

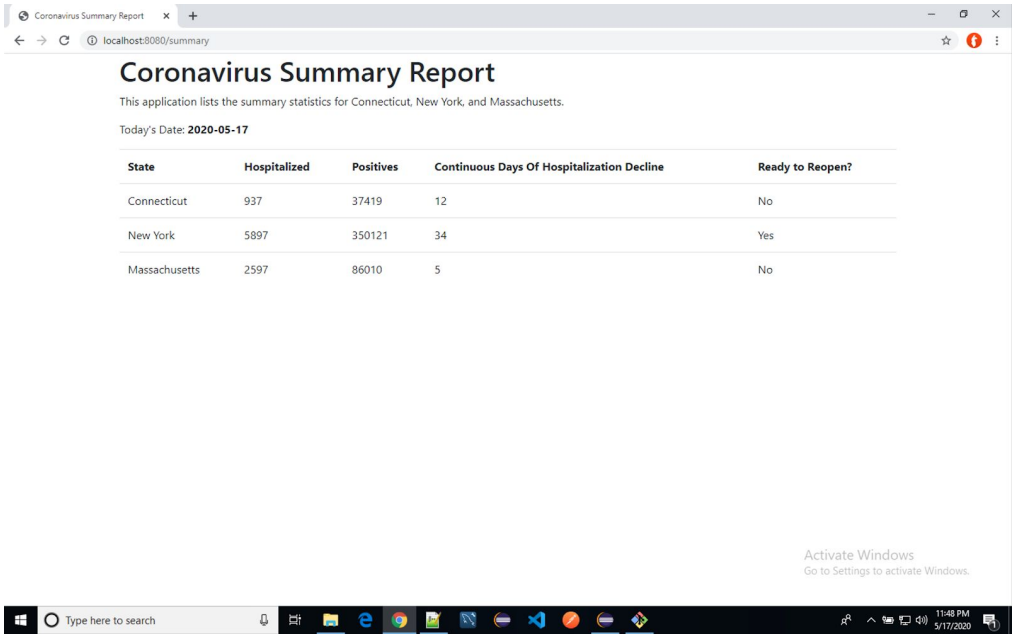
1) Reports (locahost:8080/reports) - this is raw data from Covid Tracking project on the general statistics on deaths, confirmed cases, etc. in the Tri State area - Massachusetts, Connecticut, and New York.
2) Summary (locahost:8080/summary) - this is another chart that presents a summary view of Massachusetts, Connecticut, and New York. Specifically, you will see the cumulative number of confirmed cases, the number of people hospitalized today, and most importantly the days that we have seen a *continuous* decline in hospitalizations. The end of the three data points leads to an educated guess if the state is ready to re-open.
3) State Readiness Report (locahost:8080/statereadiness) - this is a revised effort on Summary to focus on Connecticut specifically. We look at the number of confirmed cases to total teste. This is only at best an idea of how widespread the virus is in Connecticut. Secondly we look at the same metric on Summary on days of continual decline of hospitalizations. Finally we look at a new data source with Google Mobility to see if people have been following official guidance from the governor to stay at home. After all, if the people are not following the orders, then this is another perspective of the issue that we *are not ready* to re-open the state.

Sample Screenshots of Software

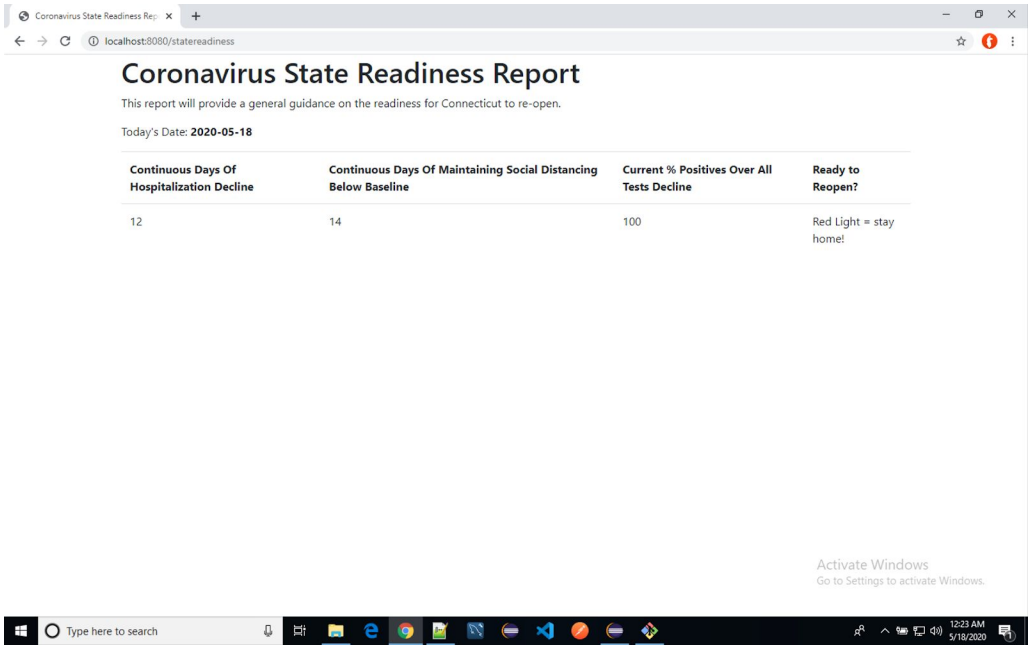
Report View For Tri-State Area (MA, CT, NY)



Summary View For Tri-State Area (MA, CT, NY)



State Readiness View For Connecticut



Input Space Partitioning (ISP)

Base Choice Coverage (BCC) was selected for the Input Space Partitioning analysis.

Table A: Determine characteristics						
Method	**Params**	**Returns**	**Values**	**Char ID**	**Characteristic**	**Covered by**
GetLatest Hospitalized()	allStats List<LocationStats>	int	int, null	C1	Non null value for state	
	state String			C2	Non-Empty String for states	
				C3	List contains nonnull values	
getLatest Positive()	allStats List<LocationStats>	int	int, null		Non null value for state	C1
					Non-Empty sString for states	C2
	state String				List contains non null values	C3
getDays SinceDecline()	allStats List<LocationStats>	int	int, null		Non null value for state	C1
	state String				Non-Empty sString for states	C2

					List contains nonnull values	C3
getVerdict()	daysSinceDecline int	String	String, null	C4	daysSinceDecline >= 0	

Table B: Design Partitioning						
Char ID	**Characteristic**	GetLatestHospitalized()	getLatestPositive()	getDaysSinceDecline()	getVerdict()	**Partition (Base Case in Bold)**
C1	Non null value	X	X	X		**True**, False
C2	Non-Empty	X	X	X		**True**, False
C3	List contains non null values	X	X	X		**True**, False
C4	daysSinceDecline >= 14				X	**True**, False

Table C: Define test requirements					
Method	**Char**	**Test Requirements**	**Infeasible TRs**	**Revised TRs**	**# TRs**
getLatestHospitalized()	C1C2C3	**TTT**,TTF, TFT,FTT	FTT, TFT, because a String cannot be both null and not empty or vice versa	FFT, TTT. Both C1 and C2 are either false or true together.	3
getLatestPositive()	C1C2C3	**TTT**,TTF, TFT,FTT	FTT, TFT, because a String cannot be both null and not empty or vice versa	FFT, TTT. Both C1 and C2 are either false or true together.	3
getDaysSinceDecline()	C1C2C3	**TTT**,TTF, TFT,FTT	FTT, TFT, because a String cannot be both null and not empty or vice versa	FFT, TTT. Both C1 and C2 are either false or true together.	3
getVerdict()	C4	**T**,F			2

Table D: Test Case Design				
Method	**Criteria**	**Case**	**Set up Data**	**Steps**
getLatest Hospitalized(), getLatest Positive(), getDays SinceDecline()	C1C2C3	**TTT**	1) state = “CT”; 2) allStats = CoronaVirusDataService.getAllStats(); 3) expectedState = “CT”; 4) expectedStatistics = CoronaVirusDataService.getAllStats();	assertEquals(expectedState, state); assertEquals(expectedStatistics, allStats);
		TTF	1) state = “CT”; 2) List<LocationStats> allStats; // new() command not issued for #2	assertNull(state); assertNull(allStats);
		TFT	1) state = “”; 2) expectedStatistics = CoronaVirusDataService.getAllStats();	assertTrue(state.isEmpty()); assertEquals(expectedStatistics, allStats);
getVerdict()	C4	**T**	1) daysSinceDecline = 15;	assertTrue(daysSinceDecline >= 14);
		F	1) daysSinceDecline = 3;;	assertFalse(daysSinceDecline >= 14);

Graph Coverage

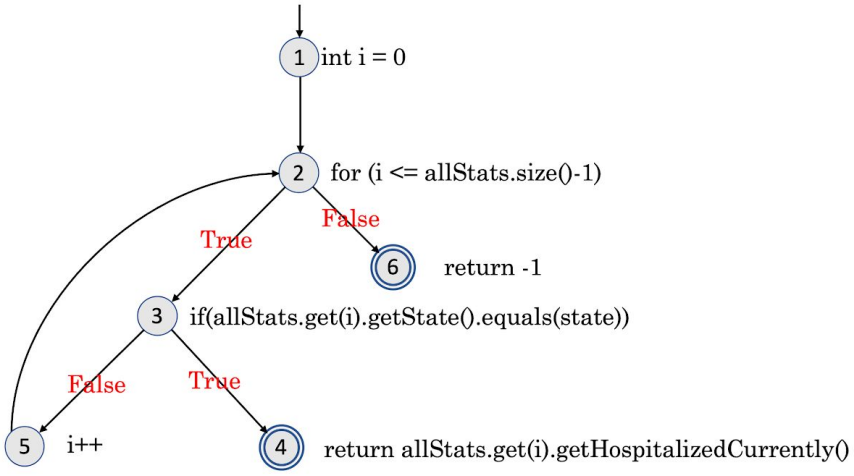
Prime Path Coverage (PPC) was selected for the Graph coverage analysis.

Code Snippet

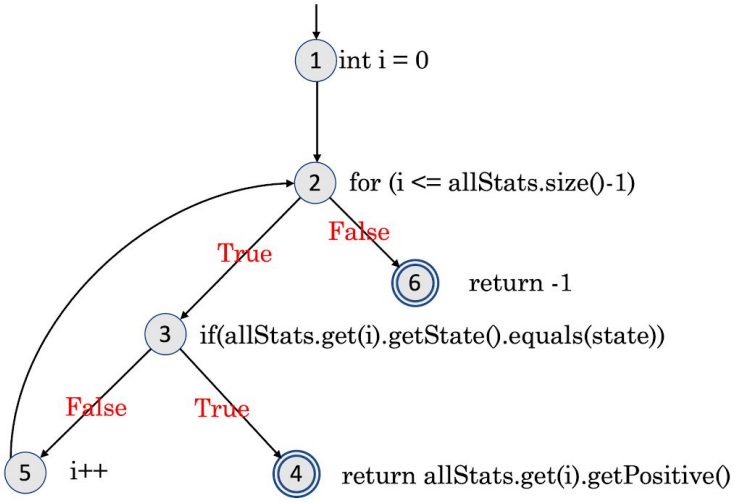
```
public int getLatestHospitalized(List<LocationStats> allStats, String state) {
    for (int i = 0; i <= allStats.size()-1; i++) {
        if(allStats.get(i).getState().equals(state)) {
            return allStats.get(i).getHospitalizedCurrently();
        }
    }

    return -1; // Did not find statistics for this state
}
```

Graphs

getLatestHospitalized(List<LocationStats> allStats, String state)



getLatestPositive(List<LocationStats> allStats, String state)



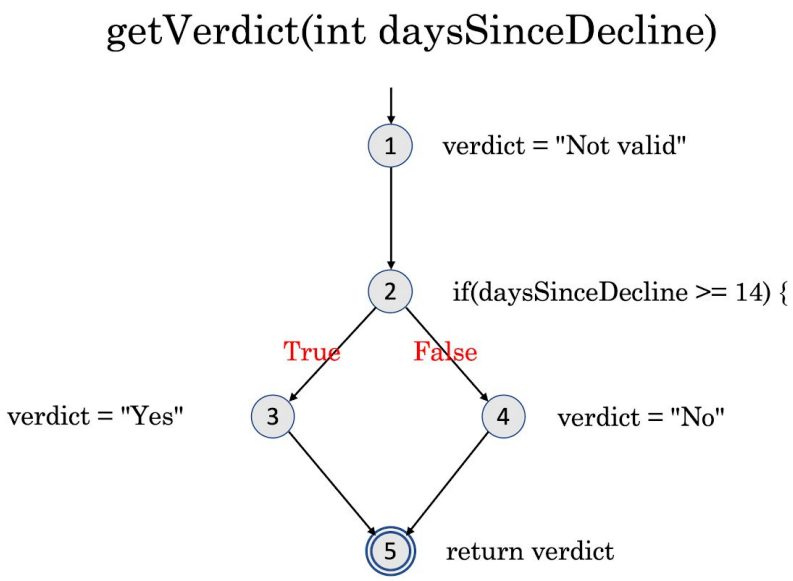
Simple Paths for getLatestHospitalized and getLatestPositive()

Len 0	Len 1	Len 2	Len 3	Prime Paths
[1]	[1,2]	[1,2,3]	[1,2,3,4]!	[1,2,6]!
[2]	[2,3]	[1,2,6]!	[1,2,3,5]	[1,2,3,4]!
[3]	[2,6]!	[2,3,4]!	[2,3,5,2]*	[1,2,3,5]
[4]!	[3,4]!	[2,3,5]	[3,5,2,3]*	[2,3,5,2]*
[5]	[3,5]	[3,5,2]	[3,5,2,6]!	[3,5,2,3]*
[6]!	[5,2]	[5,2,3]	[5,2,3,4]!	[3,5,2,6]!
		[5,2,6]!	[5,2,3,5]*	[5,2,3,4]!
				[5,2,3,5]*

Test Paths That Provide PPC for getLatestHospitalized and getLatestPositive()

Test path	Test Requirements
[1,2,6]	[1,2,6], [2,6]
[1,2,3,4]	[1,2,3,4], [2,3,4], [3,4]
[1,2,3,5,2,6]	[5,2,6], [3,5,2,6]
[1,2,3,5,2,3,4]	[5,2,3,4]

Graphs



Set of Test Cases for Test Requirements

	Test Paths	Test Case Values	Expected Values
T1	[1,2,3,5]	(20)	“Yes”
T2	[1,2,4,5]	(10)	“No”

Logic Coverage

Restricted Active Clause Coverage (RACC) was selected for the Logic Coverage analysis.

Testing Requirements

	Predicate Expression (predicate = p)	Predicate Clause(s)
	if ((hospitalized >= 14) && (daysKeepingSocialDistance >= 14) && (positivesTrend >= 80)) => p = (a ∧ b ∧ c)	a = hospitalized >= 14 b = daysKeepingSocialDistance >= 14 c = positivesTrend <= 20

This predicate was taken from "StateReadinessDataService.java" class. This is the condition for offering the appropriate recommendation for the state of Connecticut to determine if the state is ready to re-open.

Code Snippet

```
public String getRecommendation(int hospitalized, int daysKeepingSocialDistance, int positivesTrend) {
    String recommendation = "Not sure";

    if ( (hospitalized >= 14) && (daysKeepingSocialDistance >= 14) && (positivesTrend <= 20) ) {
        recommendation = "Green Light";
    } else if ((hospitalized >= 0) && (daysKeepingSocialDistance >= 0) && (positivesTrend <= 50) ) {
        recommendation = "Yellow Light - still need to wait";
    } else {
        recommendation = "Red Light = stay home!";
    }

    return recommendation;
}
```

Truth Table

	a	b	c	a ∧ b ∧ c	Pa	Pb	Pc
1	T	T	T	T	T (1,5)	T (1,3)	T (1,2)
2	T	T	F	F			T (1,2)
3	T	F	T	F		T (1,3)	
4	T	F	F	F			
5	F	T	T	F	T (1,5)		
6	F	T	F	F			
7	F	F	T	F			
8	F	F	F	F			

Analysis

Restricted Active Clause Coverage (RACC) dictates that values of both minor clauses to be identical. For instance, the predicate a (Pa) must be true for the two scenarios when the predicate is T and F. In both scenarios, the minor clauses b and c must be identical.

RACC Test Requirements

Pa = (1,5)

Pb = (1,3)

Pc = (1,2)

Test Cases

Based on the RACC analysis earlier, the six test cases are identified below:.

a = hospitalized >= 14
b = daysKeepingSocialDistance >= 14
c = positivesTrend <= 20

Pa = {TTT, FTT} = { (14, 14, 9), (5, 14, 9) }

Pb = {TTT, TFT} = { (14, 14, 9), (14, 3, 9) }

Pc = {TTT, TTF} = { (14, 14, 9), (14, 14, 80) }

Total Test Cases = (14, 14, 9), (5, 14, 9), (14, 3, 9), (14, 14, 80)

Syntax Based Testing

Production Coverage (PDC) and mutation were selected for the Syntax Based Testing analysis.

PDC Testing Requirements

Code Snippet

```
49    private List<MobilityStats> filterListByCountryState (List<MobilityStats> allMobileStats, String countryCode, String country, String state) {
50        List<MobilityStats> filteredMobileStats = new LinkedList<>();
51
52        String filter = countryCode+country+state+"No data";  // we are not looking at county level detail now, so exclude sub-region2
53
54        String tempCountryCode = "";
55        String tempCountry = "";
56        String tempState = "";
57        String tempCounty = "";
58        String temp = "";
59
60        for (MobilityStats m : allMobileStats) {
61            tempCountryCode = m.getCountry_region_code();
62            tempCountry = m.getCountry_region();
63            tempState = m.getSub_region_1();
64            tempCounty = m.getSub_region_2();
65            temp = tempCountryCode + tempCountry + tempState + tempCounty;
66
67            if (filter.equals(temp)) {
68                filteredMobileStats.add(m);
69                System.out.println(m.toString());
70            }
71        } // end for loop
72
73        return filteredMobileStats;
74    }
```

Grammar = **private** List<MobilityStats> filterListByCountryState (

List<MobilityStats> allMobileStats, String countryCode, String country, String state)

expr ::= L S S S

L ::= “List<LocationStats>”

S ::= “US” | “United States” | “Connecticut” | “Massachusetts” | “New York”

Production is any terminal in the grammar that can be rewritten or reduced further. A terminal is a symbol in the grammar that cannot be rewritten or reduced further. Here we have one production (left side of the grammar): expr ::= L S S S.

PDC Test Cases

We will assign one test case for the production.

expr ::= L S S S

List<LocationStats>, US, United States, Connecticut

Mutation Testing Requirements

In the PDC Test Case above, we designed a Sunny Day scenario valid input for the method filterListByCountryState(). Now we will use mutation to experiment with terminals that are not part of the original grammar.

Mutation Terminal Operator: Replace S with L

expr ::= L L S S

Mutation Test Case

Mutant (Invalid Input):

expr ::= **L L S S**

L ::= “List<LocationStats>”

L ::= “List<LocationStats>”

S ::= “United States”

S ::= “New York”

Mutant Result: List<LocationStats>, List<LocationStats>, United States, New York

Appendix

Option 1: Testing in practice demonstration

This option focuses on applying different types of coverage criteria covered in the class to a real application by designing tests cases. Your task will be to select an open source project(s) that contains code/requirements that are suitable for you to develop coverage criteria and test cases for the following:

- Input Space Partitioning
    - BCC
- Graph Coverage
    - PrimePathCoverage
- Logic Coverage
    - RACC
- Syntax Based Testing
    - ProductionCoverage
    - A demonstration of mutation coverage appropriate for your choice

The software under test should be simple enough for hand analysis and complex enough to demonstrate your competence in test design and generation. You are not expected in any way to provide comprehensive testing of any project. The expectation is that you pick some aspect of the open source project that provides a non-trivial example for that specific coverage. You may use the same example for all of the coverage criteria methods or use different examples. For the test cases, you are required to demonstrate how these tests satisfy the chosen testing coverage criteria.

Directions

You need to write a report describing chosen coverage criteria, how they were applied and associated test cases. You are not required to use any specific template. In reality, the format of software documentation depends largely on companies or organizations. Thus,

you may use any format to document your software as long as the document includes the following information.

- Cite the source of your software under test
    - If you use an open source software, clearly specify the owners/authors' names or cite the reference
    - If you are the sole owner of the software or collaborated with other developers, list all developers' names
- Briefly describe what the software under test does such as its purposes, services, and functionalities
- Describe how you applied each of the coverage methods
    - For each coverage criteria:
        - How you broke down the code/requirements into an appropriate abstraction
        - Details of the analysis on that abstraction (either typed up or pictures of analysis of abstraction and resulting **test requirements**)
        - **Test cases** to satisfy the abstract test requirements (inputs and rough description of test oracle (does not need to be code))
            - Be sure to analyze and justify infeasible requirements and infeasible paths.