

Table of Contents

Preface	6
Lab 1: How to create Manual Test Cases?	8
1.1 Objective:	8
1.2 Scope:	8
1.3 Useful Concepts:	8
1.4 Example:	10
1.5 Exercises:	12
1.6 Homework:	12
Lab 2: How to perform static analysis with manual efforts through test cases	14
2.1 Objective	14
2.2 Scope	14
2.3 Useful Concepts:	14
2.4 Lab Exercises	15
2.5 Homework	15
Lab 3: Black Box Testing	17
3.1 Objective:	17
3.2 Scope	17
3.3 Useful Concepts	17
3.4 Examples	18
3.5 Exercises for Lab	19
3.6 Homework	21
Lab 4: Basic White Box Testing How to perform white box testing with CFG and DFG.....	23
4.1 Objective:	23
4.2 Scope:	23
4.3 Useful Concepts	23
4.4 Examples	26
4.5 Exercises for lab:	30



4.6 Homework:	31
Lab 5: Code Inspection and Walkthroughs	33
5.1 Objective	33
5.2 Scope	33
5.3 Useful Concepts	33
5.3.1 Code Inspection:	33
5.3.2 Inspection Session (Ingredients)	34
5.4 Exercises for lab	34
5.5 Homework:	37
Lab 6: JUnit Testing	41
6.1 Objective:	41
6.2 Scope	41
6.3 Useful Concept	41
6.3.1 Installations and Methods:	41
6.4 Exercises for lab	48
6.5 Homework:	53
Lab 7: Advance JUnit and Test Suites, Assertions and Annotations	55
7.1 Objective	55
7.2 Scope	55
7.3 Useful Concept:	55
7.3.1 Annotations:	55
7.3.2 Assertions:	55
7.4 Examples:	56
7.5 Exercises for Lab:	58
7.6 Homework:	60
Lab 8: Selenium Introduction	62
8.1 Objective:	62
8.2 Scope:	62
8.3 Useful Concept:	62



8.4 Examples:	62
8.5 Exercises for lab	66
8.6 Homework:	67
Lab 9: Automation Testing Using Selenium	69
9.1 Objective:	69
9.2 Scope:	69
9.3 Useful Concept:	69
9.4 Examples	70
9.5 Exercises for lab:	72
9.6 Homework:	75
Lab 10: Debugging.....	77
10.1 Objective	77
10.2 Scope:	77
10.3 Useful Concept:	77
10.4 Examples:	81
10. 5 Exercises for lab:	85
10. 6 Homework	90
Lab 11: Mutation Testing	92
11.1 Objective:	92
11.2 Scope	92
11.3 Useful Concept	92
11.4 Examples	92
11.5 Exercises for lab:	94
11.6 Homework	97
Lab 12: Mutation Testing Using PIT.....	99
12.1 Objective:	99
12.2 Scope:	99
12.3 Useful Concept:	99
12.4 Examples:	99



12.5 Exercises for lab:	104
12.6 Homework:	108
Lab 13: GUI Testing	110
13.1 Objective:	110
13.2 Scope:	110
13.3 Useful Concept:	110
13.4 Examples:	111
13.5 Exercises for lab:	115
13.6 Homework:	116
Lab 14: Sikuli.....	118
14.1 Objective:	118
14.2 Scope:	118
14.3 Useful Concept:	118
14.4 Examples:	118
14.5 Exercises for lab:	120
14.6 Homework:	122
Lab 15: Appium	124
15.1 Objective:	124
15.2 Scope:	124
15.3 Useful Concept:	124
15.3.1 Installation:	125
15.4 Examples:	127
15.5 Exercises for lab:	129
15.6 Homework:	131
Lab 16: Document Inspection and Fault Estimation	133
16.1 Objective:	133
16.2 Scope:	133
16.3 Useful Concept:	133
16.4 Exercises for lab:	134



16.5 Homework:	138
-----------------------------	------------



Preface

This lab manual is designed for Software Testing course. It has one pre-requisite course i.e. Software Quality Engineering. Software Quality Engineering provides general view of quality in development, quality, and management perspective while Software Testing course is a subset of Software Quality domain and is specific to code and test. It contains all the industrial practices regarding testing of a code in different platforms and scenarios. Software Testing is a trending course in perspective of IT industry, hence, enhancement in the course is a necessary step. The course starts with manual efforts in testing that lead to automated testing in general and java programs testing in specific. JUnit, Selenium, Appium, PIT, Sikuli and Debugger are the main tools that support the course in lab. It deals with unit testing, system testing, testing web platform, mobile testing, error seeding and testing, inspections, document testing, manual testing, and reporting. The course is designed in a special flow so that dependencies could be handled.

We would like to thank all the faculty members of Computer Science Department, COMSATS University Islamabad, Abbottabad Campus for their helpful comments and suggestions, specifically, Dr Faiz Ali Shah. Ms. Sehr Andleeb has invested valuable hours in the designing and revision of the manual under the guidance of Dr Syed Sajid Hussain. I would like to thank all of them for their contribution and support. I am also thankful to Mr. Tariq Baloch for his valuable suggestions in compilation of the manual.



Lab # 01

How to create Manual Test Cases



Lab 1: How to create Manual Test Cases?

1.1 Objective:

- Learn basic testing techniques
- Learn basic testing template
- Learn Scenario based testing

1.2 Scope:

The student should know the following at the end of this lab:

1. Scenario based test cases development
2. Manual efforts for generating test cases
3. Be able to determine the missing dots in test cases development

1.3 Useful Concepts:

Test Cases:

Simple test case table can be formatted in any desired table with desired headings. The simplest amongst them is the one without which test cases cannot be generated.

- Test Case is a simple pair of

<input, expected outcome>

- Test cases are not that simple. A test case may consist of a sequence of **<input, expected outcome>**



Simple Headings:

Test Case ID	The ID of the test case (Unique ID)
Test Case Summary/ Description	The summary / objective of the test case (Why the test is to be performed)
Test Data/ Input	The test data, or links to the test data, that are to be used while conducting the test. (Input Vector)
Expected Result	The expected result of the test. (Describe the expected result in detail including message/error that should be displayed on the screen)
Actual Result	The actual result of the test; to be filled after executing the test.
Status/ Verdict	If actual result is not as per the expected result, then mark this test as failed . Otherwise, update it as passed .

Scenario

As we learned in object oriented software engineering, a scenario is a path of achieving a goal of the use case. We describe the steps to achieve a goal of the use case. Scenario is positive if goal is achieved and negative otherwise. Scenario can be represented in a brief paragraph or fully dressed steps, it also represents a normal, alternative, or exceptional flow of events.

Identification of Scenarios

First understand the context, the requirements, and the expectations of your users and stakeholders for the use case. You can also use techniques such as brainstorming, mind mapping, or storyboarding to generate and visualize scenarios.

Design test cases from scenarios

- Translate the scenarios into testable steps that verify the expected behavior and outcome of your software.



- Use various methods, such as decision tables, state diagrams, or boundary value analysis, to identify the test conditions, inputs, outputs, and expected results for each scenario.
- Use tools such as test management software, spreadsheets, or flowcharts to document and organize your test cases.

Execution and Evaluation of test cases

1. Follow the test steps and compare the actual results with the expected results.
2. Use various techniques, such as exploratory testing, usability testing, or performance testing
3. Use tools such as test automation software, bug tracking software, or analytics software to run, monitor, and report your test cases.

1.4 Example:

(Test Scenario) Triangle for obtuse angle (angle > 90 is obtuse)



Make a table as follows:



Test Case ID	Test Case Description	Input Data	Expected Outcome	Actual Outcome	Status
TC_01	To draw triangle for angle >90	95	Message: Triangle Drawn Successfully	Message: Triangle Drawn successfully	Pass
TC_02	To not draw triangle for angle <90	89	Message: Angle is not obtuse	Message: Angle is not obtuse	Pass
TC_03	To not draw triangle for angle=90	90	Message: Angle is not obtuse	Message: Angle is not obtuse	Pass
TC_04	To not draw triangle for angle < 90	60	Message: Angle is not drawn	Triangle drawn	Fail
TC_05	To not draw triangle for angle <=0	0	Message: 0 is invalid	Triangle drawn for 90	Fail

Test Case ID	Test Case Description	Input Data	Expected Outcome	Actual Outcome	Status
TC_06	To not draw triangle for angle <0	-20	Message: -20 is invalid	Message: Triangle Drawn successfully	Fail
TC_07	To draw triangle for angle >90	100	Message: Triangle is drawn	Message: Angle is not obtuse	Fail



1.5 Exercises:

- Make test cases for a triangle that can differentiate between isosceles, equilateral, and scalene types based on sides length as input and state if their verdict is true/false. Make assumptions for actual outcome.

1.6 Homework:

- Create Logic to find third angle of a triangle by your own. Write code on paper and make test cases to see the results. On different values, dry run the input for actual value.
- For any software that you have developed or you are using it, Pick up a use case and then:
 - Draw/paste related screens in the order of its use.
 - Create a table of the possible test cases as per the basic template mentioned in example.
 - Save your work in PDF format to cloud storage and submit its path to your evaluation system.
 - Review for at least 2 students and suggest them improvements or mention what you liked about his solution.



Lab # 02

How to Perform Static Analysis with manual efforts through test cases



Lab 2: How to perform static analysis with manual efforts through test cases

2.1 Objective

Aim of this lab is to provide sufficient knowledge about static ways of testing. Scenarios are focused in the lab.

2.2 Scope

At the end of lab students will be able to learn:

Static Testing

Scenarios

Domain Testing

2.3 Useful Concepts:

Points:

- Writing test cases have their own template which must be followed.
- Both positive and negative scenarios should be covered with explicitly mentioning actual outcome against input and mapping to expected outcome.
- Either test case passes or fails. There is no other mediocre choice.

Static Analysis:

- Examination of several documents, namely requirements documents, software models, design documents, and source code
- Static analysis includes code review, inspection, walk-through, algorithm analysis, and proof of correctness
- It does not involve actual execution of the code under development.

Our focus on this lab is algorithm analysis, source code analysis and review.

Static Testing is a software testing technique which is used to check defects in software application without executing the code. Static testing is done to avoid errors at an early stage of development as it is easier to identify the errors and solve the errors. It also helps finding errors that may not be found by Dynamic Testing.

Its counterpart is Dynamic Testing which checks an application when the code is run. Refer to this tutorial for a detailed difference between static and dynamic testing.

The two main types of static testing techniques are



- Manual examinations: Manual examinations include analysis of code done manually, also known as REVIEWS.
- Automated analysis using tools: Automated analysis are basically static analysis which is done using tools.

In this lab we will focus on manual examinations.

2.4 Lab Exercises

ACTIVITY:

Create a program in java to implement Logic to find third angle of a triangle. After that check the triangle type with respect to the angle. Write the program on either paper or compiler but do not execute.

If it's on paper, move it to the peer for static review

If it's on compiler, hand it over to the peer

Peer work:

Check all the program and comment the critical lines if they are correct.

Write test cases for the scenario and check/ dry run the program accordingly

Pass the verdict back to the developer and developer needs to rectify the program.

Re-evaluate the program by peer and selected mistakes would be rechecked by same test cases developed before.

2.5 Homework

Analyze your own code and static review your code



Lab # 03

Black Box Testing Techniques



Lab 3: Black Box Testing

3.1 Objective:

To learn Black Box testing techniques

To learn making partitions for testing (Equivalence Partitioning)

To learn about Boundary Value Analysis

Merge both techniques and their usage criteria

3.2 Scope

Equivalence Partitioning

Boundary Value Analysis

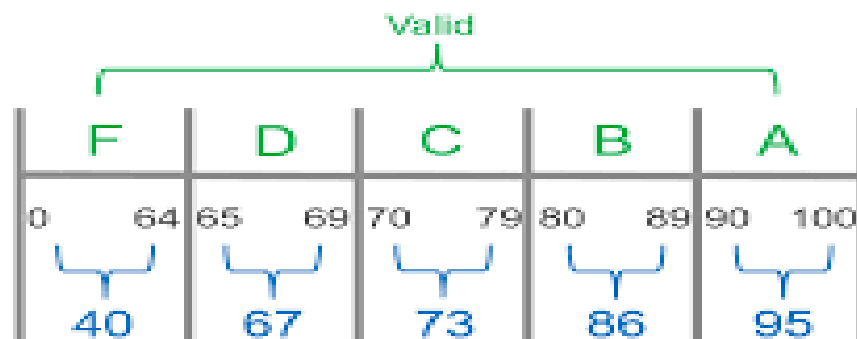
3.3 Useful Concepts

Equivalence Partitioning:

Dividing the test input data into a range of values and selecting one input value from each range is called **Equivalence Partitioning**.

This technique is used to reduce an infinite number of test cases to a finite number, while ensuring that the selected test cases are still effective test cases which will cover all possible scenarios.

Equivalence Partitioning



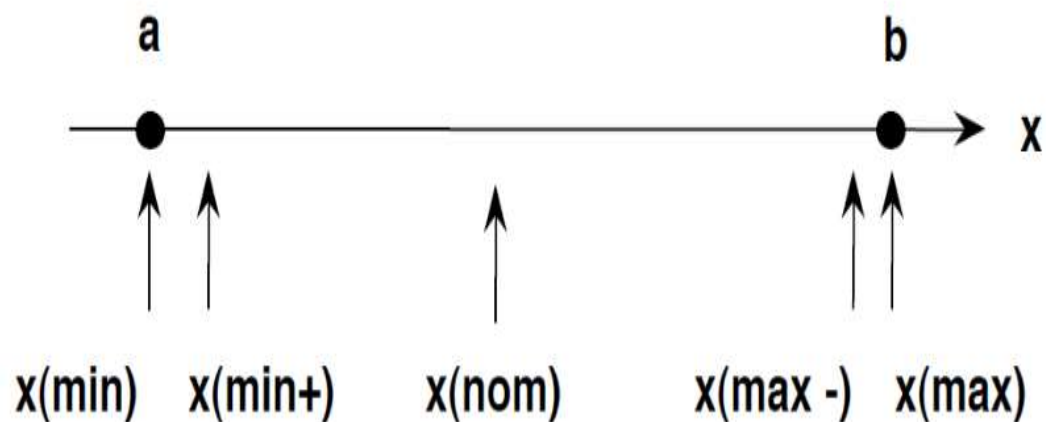
Boundary Value Analysis:

Boundary value analysis is a test case design technique to test boundary value between partitions (both valid boundary partition and invalid boundary partition).



Boundary value analysis is another black box test design technique, and it is used to find the errors at boundaries of input domain rather than finding those errors in the center of input.

- The basic idea in boundary value testing is to select input variable values at their:
- Minimum
- Just above the minimum
- A nominal value
- Just below the maximum
- Maximum



3.4 Examples

1. EP:

If one application is accepting input range from 1 to 100, using equivalence class we can divide inputs into the classes, for example, one for valid input and another for invalid input and design one test case from each class.

In this example test cases are chosen as below:

One is for valid input class i.e., selects any value from input between ranges 1 to 100. So here we are not writing hundreds of test cases for each value. Any one value from this equivalence class should give you the same result.

One is for invalid data below lower limit i.e., any value below 1.



One is for invalid data above upper limit i.e., any value above 100.

2. EP

Purchase amount (in Rs)	Discount (%)
≥ 999	5
≥ 1999	10
≥ 3999	15
≥ 5999	25
≥ 7999	35
≥ 9999	50

3. BVA

For example, an Address text box which allows maximum 500 characters. So, writing test cases for each character once will be very difficult so that will choose boundary value analysis.

3.5 Exercises for Lab

Scenario 1: Select equivalence partitioning based inputs and make test cases after classifying them in valid and invalid compartments.

Mobile Number: (accepts 10 digits)

EQUIVALENCE PARTITIONING		
INVALID	VALID	INVALID

Scenario 2: Select BVA technique and make test cases after classifying them to valid and invalid categories.



Scenario 3:

- ▶ An integer field shall contain values between and including 1 to 15. By applying EP which of the following is a valid collection of equivalence classes for the given scenario.
 - ▶ Less than 1, 1 through 15, more than 15
 - ▶ Negative numbers, 1 through 15, above 15
 - ▶ Less than 1, 1 through 14, more than 15
 - ▶ Less than 0, 1 through 14, 15 and more 3

Scenario 4:

- ▶ In a system designed to work out the tax to be paid:
An employee has £4000 of salary tax free. The next £1500 is taxed at 10% The next £28000 is taxed at 22% Any further amount is taxed at 40% Which of these groups of numbers would fall into the same equivalence class?
 - ▶ £4800; £14000; £28000
 - ▶ £5200; £5500; £28000
 - ▶ £28001; £32000; £35000
 - ▶ £5800; £28000; £32000

Scenario 5:



Purchase discount is 0% for up to 500 US\$, 5% is added for each additional 500 US\$ up to 2000 US\$, and 25% is applied for above 2000 US\$. Which test inputs in US\$ would be selected for valid equivalence partitions?

- (a) 250, 700, 1400, 1800, 4000
- (b) 250, 1400, 3000
- (c) -100, 250, 650, 1300, 1700, 2900
- (d) 200, 720, 1600, 1800, 2100

3.6 Homework

From the above exercises, merge both EP and BVA technique and make test cases accordingly.



Lab # 04

Basic White Box Testing



Lab 4: Basic White Box Testing How to perform white box testing with CFG and DFG

4.1 Objective:

To learn how to test code by making control flow graph

To learn how to test code by making data flow graph

To learn how to do white box testing and applying its technique in specific

4.2 Scope:

To test the code via Control Flow Graph

To test the code via Data Flow Graph

4.3 Useful Concepts

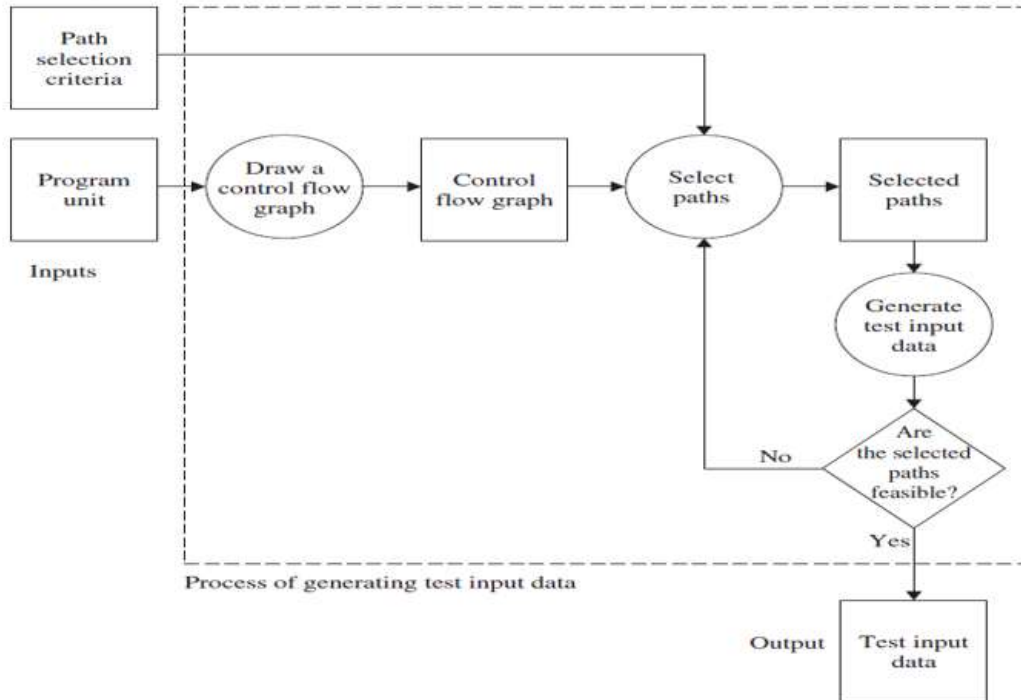
CFG:

The main idea in control flow testing is to appropriately select a few paths in a program unit and observe whether the selected paths produce the expected outcome. By executing a few paths in a program unit, the programmer tries to assess the behavior of the entire program unit.

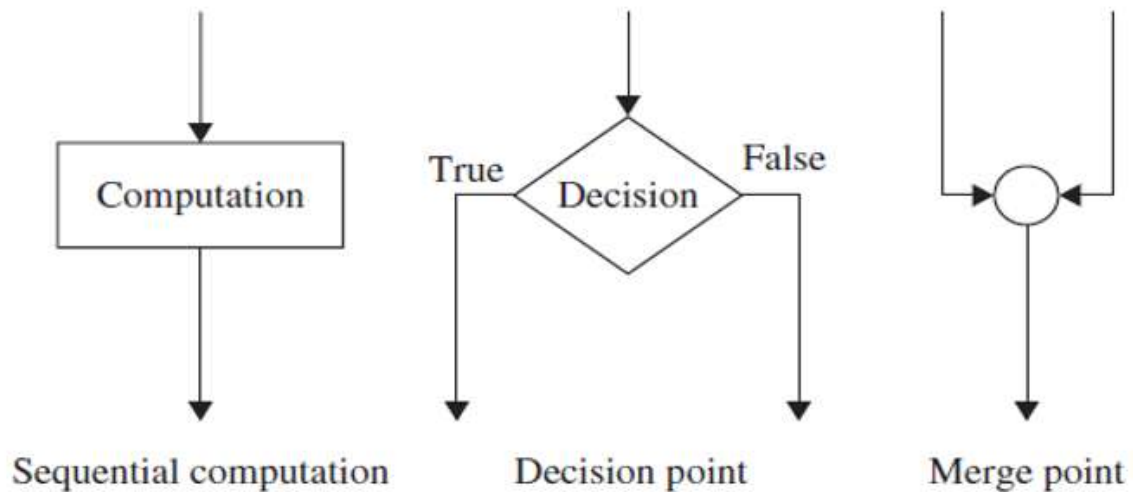
- Select paths such that every statement is executed at least once.
- Select paths such that every conditional statement, for example, an if() statement, evaluates to *true* and *false* at least once on different occasions. A conditional statement may evaluate to true in one path and false in a second path.



Test Input Data for CFG:



Symbols in CFG:

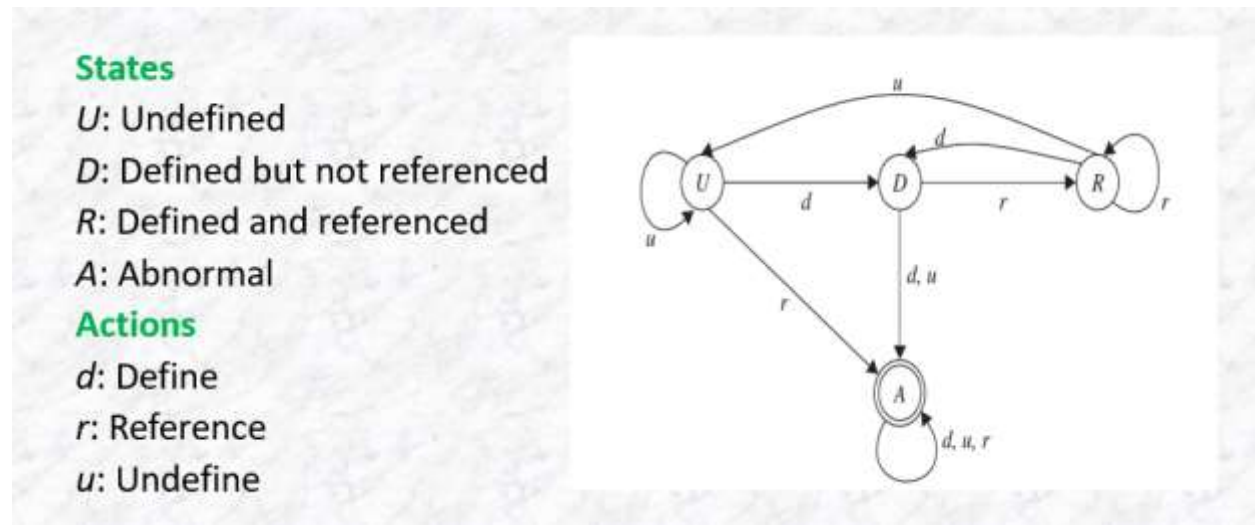


DFG:

The potential program defects are commonly known as data flow anomaly

- *Defined and Then Defined Again (Type 1)*
- *Undefined but Referenced (Type 2)*
- *Defined but Not Referenced (Type 3)*

State Transition Diagram of a Variable:



Two uses of a variable:

- **Computation use (c-use):** This directly affects the computation being performed. In a c-use, a potentially new value of another variable or of the same variable is produced. Referring to the C function VarTypes(), the statement

`*iptr = i + x;`

gives examples of c-use of variables *i* and *x*.

- **Predicate use (p-use):** This refers to the use of a variable in a predicate controlling the flow of execution. Referring to the C function VarTypes(), the statement

`if (*iptr > y) ...`

gives examples of p-use of variables *y* and *iptr*.



4.4 Examples

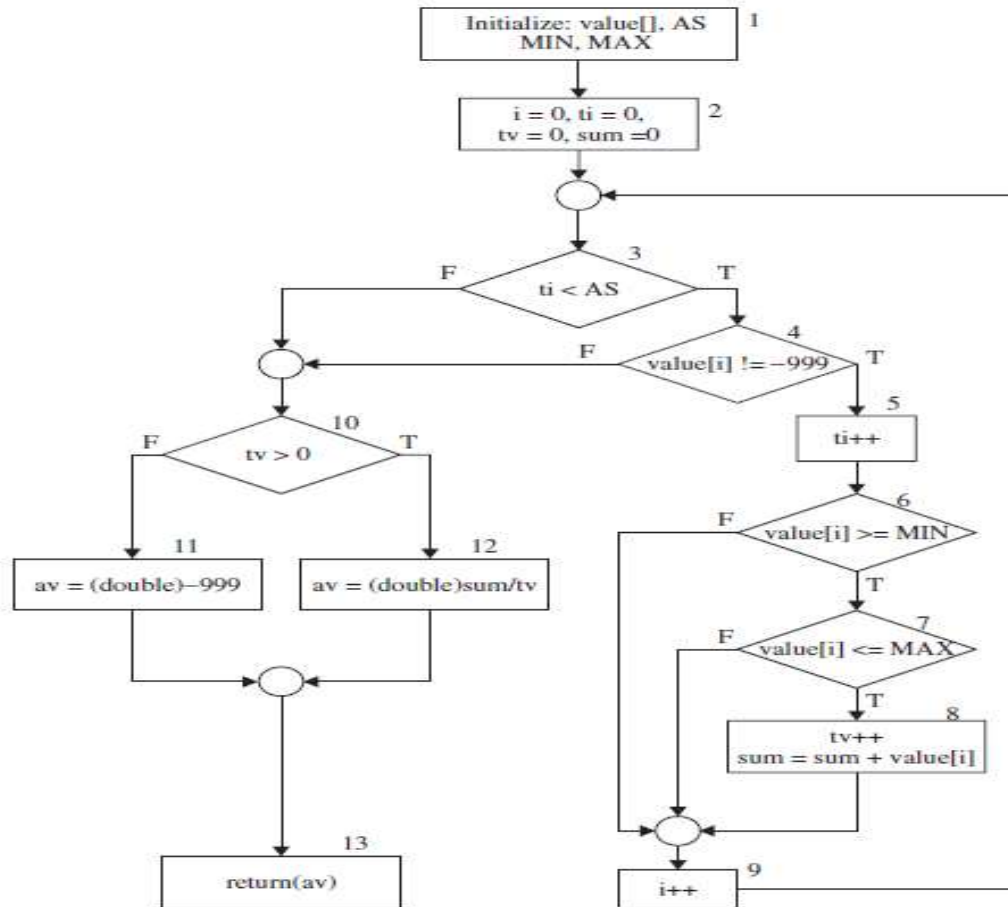
There is a need to draw CFG and DFG of a given code to analyze it in white box way.

For example, the given code is:

```
public static double ReturnAverage(int value[],int AS, int MIN, int MAX)
int i, ti, tv, sum; double av;
i = 0; ti = 0; tv = 0; sum = 0;
while ( ti< AS && value[i] != -999) {
ti++;// total index ... ti=ti+1
if (value[i] >= MIN && value[i] <= MAX) {
tv++; total value
sum = sum + value[i];
} i++; }
if (tv > 0)
av = (double)sum/tv;
else av = (double) -999;
return (av);
}
```

CFG:





Paths:

Path 1 1-2-3(F)-10(T)-12-13

Path 2 1-2-3(F)-10(F)-11-13

Path 3 1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13

Path 4 1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13

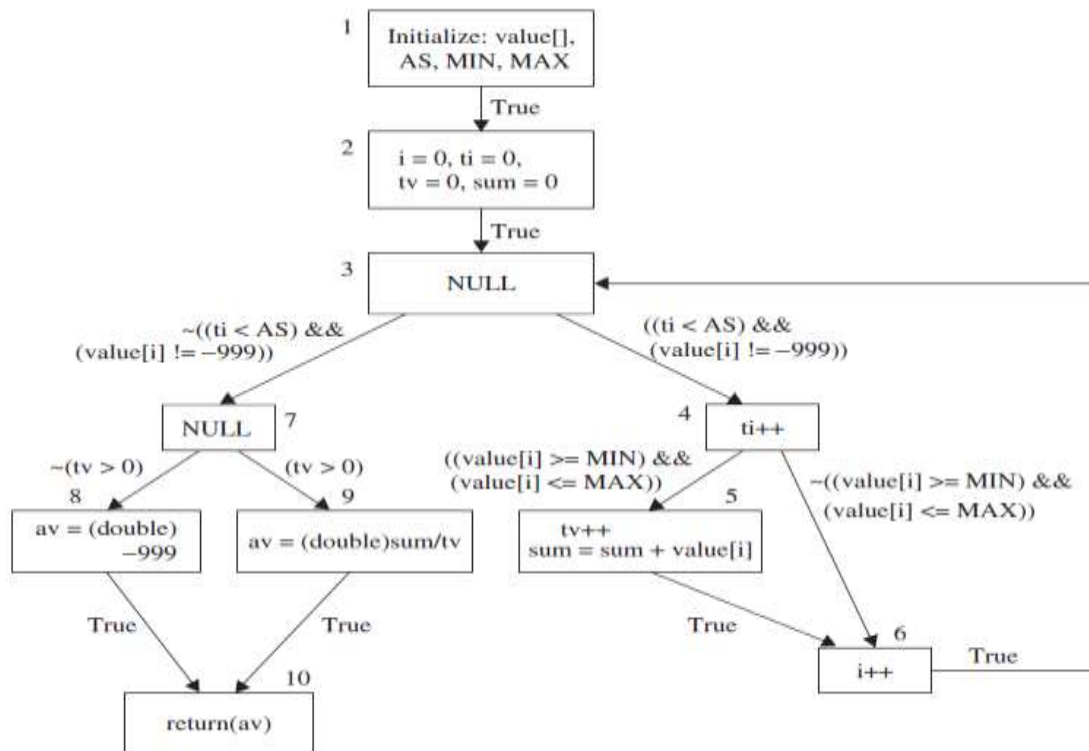
These paths should be covered in a proper way i.e.

- Select *all* paths.
- Select paths to achieve complete *statement* coverage.
- Select paths to achieve complete *branch* coverage.



- Select paths to achieve *predicate coverage*.

DFG of given code:



Paths can be defined as follow:

- *Simple Path*: A simple path is a path in which all nodes, except possibly the first and the last, are distinct.
- Paths **2-3-4-5** and **3-4-6-3** are simple paths



C use of a variable:

Nodes i	def(i)	c-use(i)
1	{value, AS, MIN, MAX}	{}
2	{i, ti, tv, sum}	{}
3	{}	{}
4	{ti}	{ti}
5	{tv, sum}	{tv, i, sum, value}
6	{i}	{i}
7	{}	{}
8	{av}	{}
9	{av}	{sum, tv}
10	{}	{av}

P use of a variable:



Edges (i, j)	predicate(i, j)	p-use(i, j)
(1, 2)	True	{}
(2, 3)	True	{}
(3, 4)	(ti < AS) && (value[i] != - 999)	{i, ti, AS, value}
(4, 5)	(value[i] <= MIN) && (value[i] >= MAX)	{i, MIN, MAX, value}
(4, 6)	~((value[i] <= MIN) && (value[i] >= MAX))	{i, MIN, MAX, value}
(5, 6)	True	{}
(6, 3)	True	{}
(3, 7)	~((ti < AS) && (value[i] != - 999))	{i, ti, AS, value}
(7, 8)	~(tv > 0)	{tv}
(7, 9)	(tv > 0)	{tv}
(8, 10)	True	{}
(9, 10)	True	{}

4.5 Exercises for lab:

Consider the following code and draw its CFG as well as DFG with defining their paths and check the whole code by white box testing:

Code:

```
FILE *fptr1, *fptr2, *fptr3; /* These are global variables. */
int openfiles(){
    int i = 0;
    if(
        ((( fptr1 = fopen("file1", "r")) != NULL) && (i++)&& (0)) || ((( fptr2 = fopen("file2", "r")) != NULL) &&
        (i++)&& (0)) || ((( fptr3 = fopen("file3", "r")) != NULL) && (i++))
    );
    return(i);
}
```



4.6 Homework:

Draw out C use and P use of this code using DFG



Lab # 05

Code Inspection and Walkthroughs



Lab 5: Code Inspection and Walkthroughs

5.1 Objective:

To learn about inspecting code in a team using inspection process (Code Inspection)

To learn about different inspection errors that may arise while writing code

5.2 Scope

- Inspection of code
- Error Checklist
- Errors Identification

5.3 Useful Concepts

5.3.1 Code Inspection:

- A code inspection is a set of procedures and error-detection techniques for group code reading.
- An inspection team usually consists of four people. One of the four people plays the role of moderator. The moderator is expected to be a competent programmer, but he or she is not the author of the program and need not be acquainted with the details of the program
- The second team member is the programmer. The remaining team members usually are the program's designer (if different from the programmer) and a test specialist.

Duties of moderator:

- Distributing materials for, and scheduling the inspection session
- Leading the session
- Recording all errors found
- Ensuring that the errors are subsequently corrected

Inspection Session Activities:

- Two activities occur:
 - 1) The programmer narrates, statement by statement, the logic of the program. During the discourse, other participants should raise questions, and they should be pursued to determine whether errors exist. It is likely that the programmer rather than the other team members will find many of the errors found during this narration.
 - 2) The program is analyzed with respect to a checklist of historically common programming errors (such a **checklist**)



5.3.2 Inspection Session (Ingredients)

- The time and location of the inspection should be planned to avoid all outside interruptions
- The optimal amount of time for the inspection session appears to be from 90 to 120 minutes
- Most inspections proceed at a rate of approximately 150 program statements per hour (For that reason, large programs should be examined in multiple inspections, each inspection dealing with one or several modules or subroutines)
- After the session, the programmer is given a list of the errors found

5.4 Exercises for lab

Students are provided the Checklist and they must identify and explore each type of general errors that may arise during inspection session.

Inspection Checklist of Errors:

Data Reference Errors:

1. Unset variable used?
2. Subscripts within bounds?
3. Non integer subscripts?
4. Dangling references?
5. Correct attributes when aliasing?
6. Record and structure attributes match?
7. Computing addresses of bit strings?
Passing bit-string arguments?
8. Based storage attributes correct?
9. Structure definitions match across procedures?
10. Off-by-one errors in indexing or subscripting operations?
11. Are inheritance requirements met?



Computation Errors:

1. Computations on nonarithmetic variables?
2. Mixed-mode computations?
3. Computations on variables of different lengths?
4. Target size less than size of assigned value?
5. Intermediate result overflow or underflow?
6. Division by zero?
7. Base-2 inaccuracies?
8. Variable's value outside of meaningful range?
9. Operator precedence understood?
10. Integer divisions correct?



Data Declaration Errors:

1. All variables declared?
2. Default attributes understood?
3. Arrays and strings initialized properly?
4. Correct lengths, types, and storage classes assigned?
5. Initialization consistent with storage class?
6. Any variables with similar names?

Comparison Errors:



1. Comparisons between inconsistent variables?
2. Mixed-mode comparisons?
3. Comparison relationships correct?
4. Boolean expressions correct?
5. Comparison and Boolean expressions mixed?
6. Comparisons of base-2 fractional values?
7. Operator precedence understood?
8. Compiler evaluation of Boolean expressions understood?

Students have to make a report, sample of one of the errors is as follows:

Unset Variable Used?	
Positive	Negative
Int a = 0; Int c=10; Int b = a+c;	Int a; Int b = 1; Int c = a+b;

All the errors under all the categories must be understood in this lab which is the basis of software testing.

5.5 Homework:

Students are provided with simple code and asked to inspect like it is done in the session.

Code 1:

```
public class Circle
{
    private double radius;
```



```

public CircleR(double r)

{

    radius = r;

}

public diameter()

{

    double d = radius * 2;

    return d;

}

}

```

Code 2:

```

public String[] OpenFile() throws IOException {

    Map<String, Double> map = new HashMap();

    FileReader fr = new FileReader("money.txt");

    BufferedReader br = new BufferedReader(fr);

    try{

        while (br.ready()){

```



```
String str = br.readLine();

String[] list = str.split(" ");

System.out.println(list);

}

} catch (IOException e){

    System.err.println("Error - IOException!");

}

}
```



Lab # 06

Junit Testing



Lab 6: JUnit Testing

6.1 Objective:

To implement JUnit Testing (Unit Testing Framework in Java)

To learn making test cases against scenario

To learn automated test cases

To learn about maven repository

6.2 Scope

To learn maven, JUnit, and creation of test cases in an automated way

6.3 Useful Concept

JUnit Maven:

The junit-jupiter-api dependency provides the public API that allows us to write tests and extensions which use JUnit 5. The junit-jupiter-engine dependency allows us to run tests which use JUnit 5.

JUnit:

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit.

Maven Repository:

In Maven terminology, a repository is a directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

A repository in Maven holds build artifacts and dependencies of varying types. There are exactly two types of repositories: local and remote: the local repository is a directory on the computer where Maven runs. It caches remote downloads and contains temporary build artifacts that you have not yet released.

6.3.1 Installations and Methods:

Prerequisites:

-----JAVA Environment Setup-----

Step 1 - Set JAVA Environment



Set the JAVA_HOME environment variable to point to the base directory location where Java is installed on your machine.
For example

JAVA_HOME to C:\Program Files\Java\jdk1.8.0_91

Append Java compiler location to System Path.

Append the string ";C:\Program Files\Java\jdk1.8.0_91\bin" to the end of the system variable, Path.

=> Verify Java Installation using java -version command as explained above.

Step 2 - Verify Java Installation on your Machine
Open Command Console and run "java -version"
c:\> java -version

Output : java version "1.8.0_91"

-----MAVEN SETUP-----

Step 3: Download Maven Archive
<https://maven.apache.org/download.cgi> based on your OS.
as we are using windows so we download Binary zip archive

Step 4 - Extract the Maven Archive

Step 5 - Set Maven Environment Variables
Set the environment variables using system properties.
MAVEN_HOME = C:\Program Files\Maven\apache-maven-3.6.3
M2 = %MAVEN_HOME%\bin

Step 6 - Add Maven bin Directory Location to System Path
Append the string; %M2% to the end of the system variable, Path.



Path =; %M2%.

Step 7 - Verify Maven Installation

Open Command Console c:\> mvn --version

-----Create New Maven Project-----

```
<groupId></groupId>
<artifactId></artifactId>
<version></version>
<scope></scope>
```

GroupId : used to identify project group

ArtifactId: used to identify project.

Version : Current Version of Project.

The <scope> element can take 6 values:

compile, provided, runtime, test, system and import.

This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.

what we have learned till now.

- 1) Java Environment
- 2) Maven Environment Setup
- 3) Created New Project using IntelliJ IDEA.
- 4) defined custom repository for maven dependencies.
- 5) maven home directory path setting.
- 6) download default maven repositories.

To Add New dependency, we need to add in <dependencies> </dependencies>

e.g lets add junit dependency

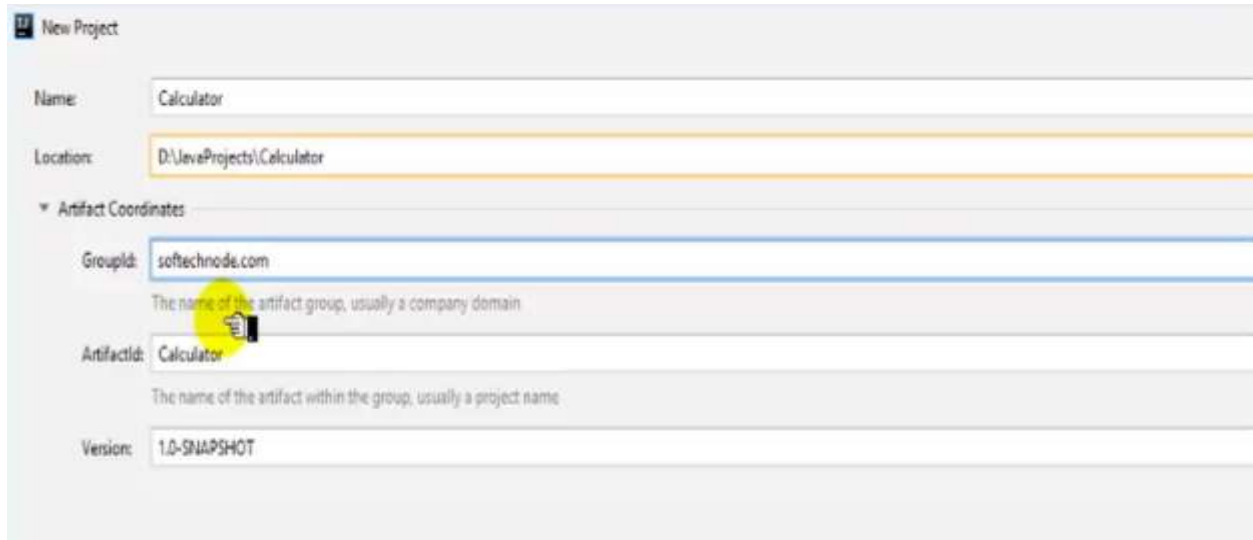
```
<dependencies>
  <dependency>
```



```
<groupId>org.junit</groupId>
<artifactId>junit5-engine</artifactId>
<version>5.0.0-ALPHA</version>
</dependency>
</dependencies>
```

Examples

Create a new project in IntelliJ



New Project

Name: Calculator

Location: D:\JavaProjects\Calculator

Artifact Coordinates

GroupId: softechnode.com
The name of the artifact group, usually a company domain

ArtifactId: Calculator
The name of the artifact within the group, usually a project name

Version: 1.0-SNAPSHOT

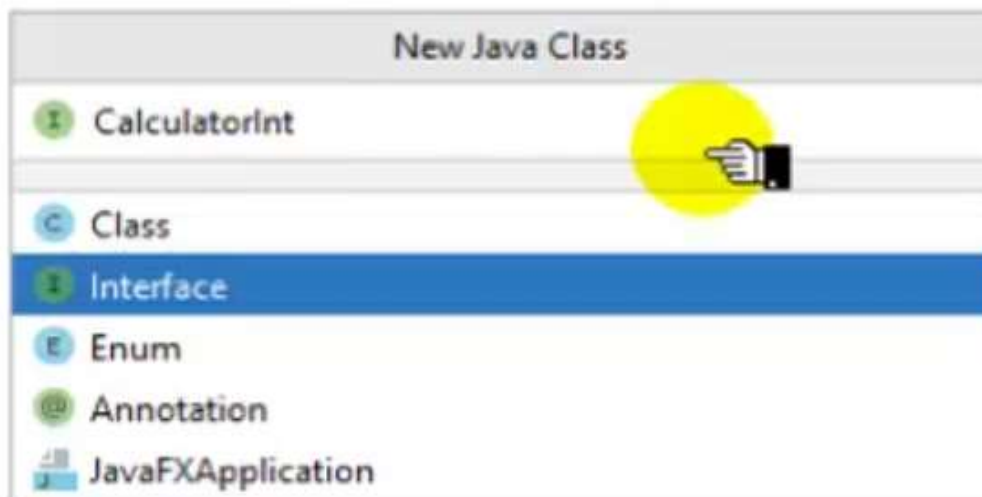
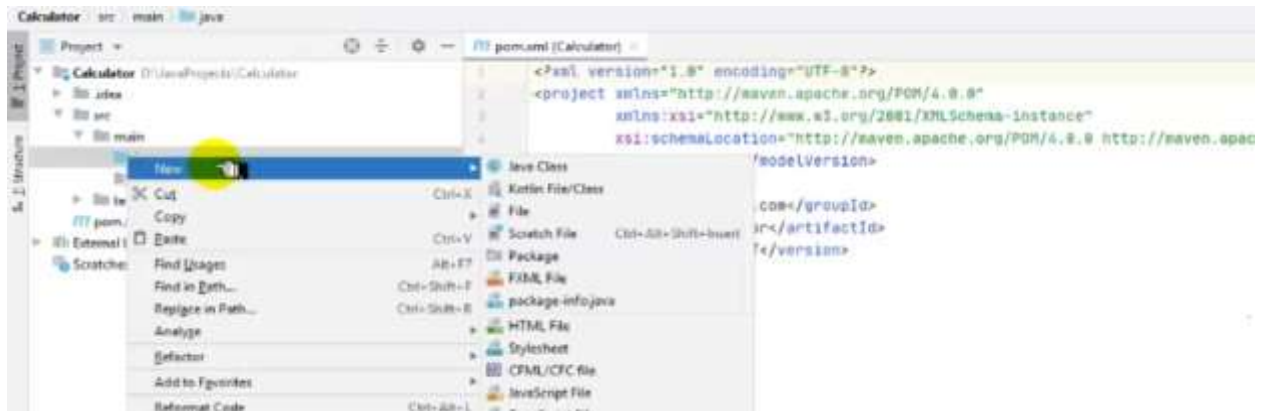
POM file will be created in this way



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>softechnode.com</groupId>
  <artifactId>Calculator</artifactId>
  <version>1.0-SNAPSHOT</version>
</project>
```

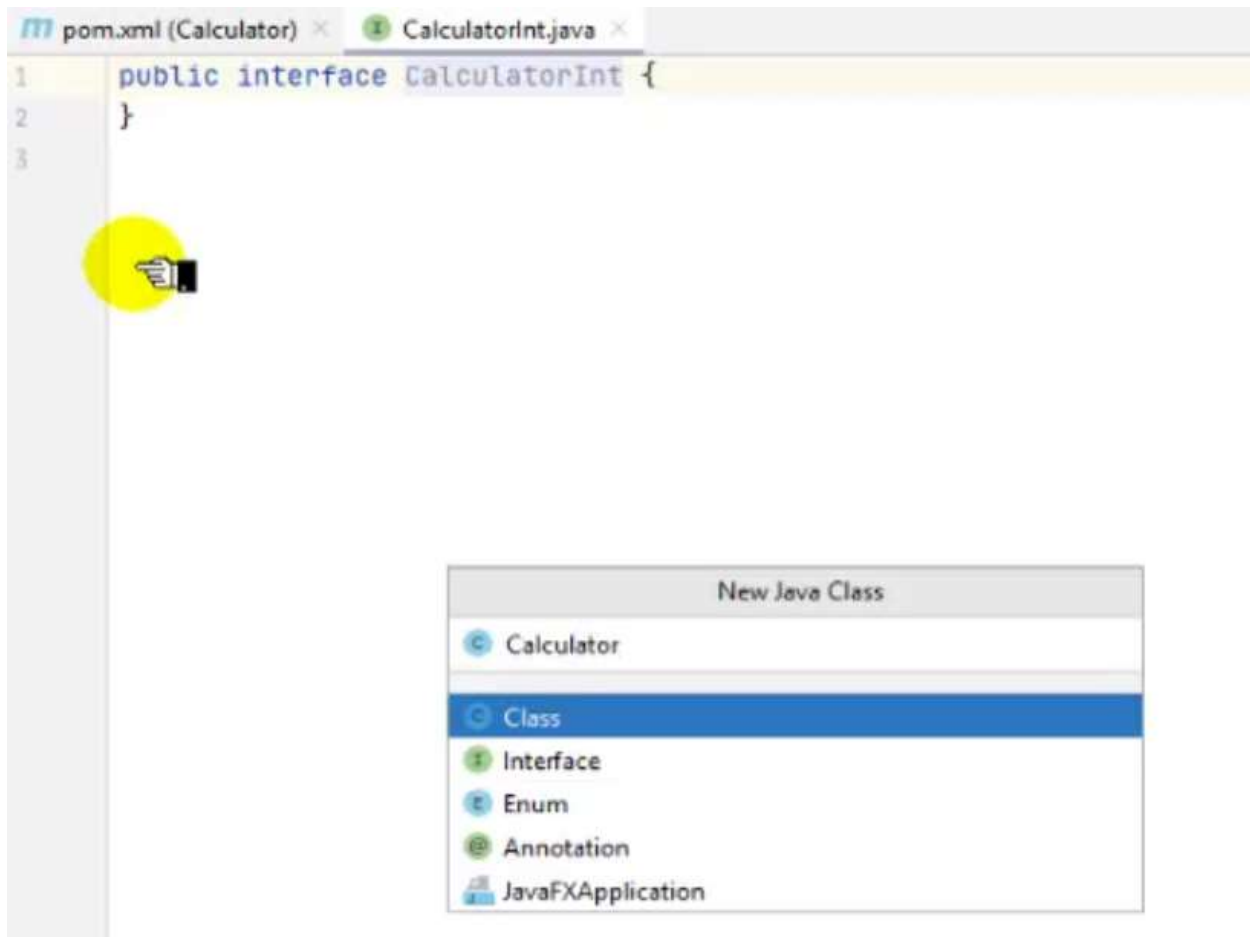
Create a java class as an Interface



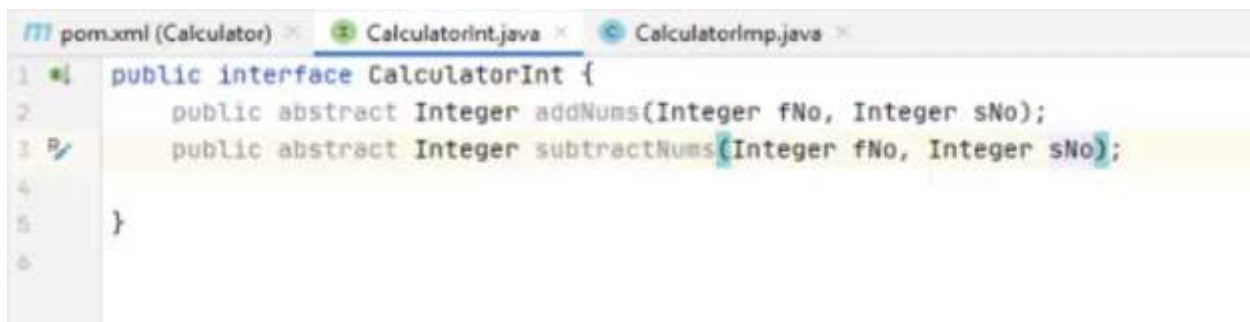


After Interface create java class



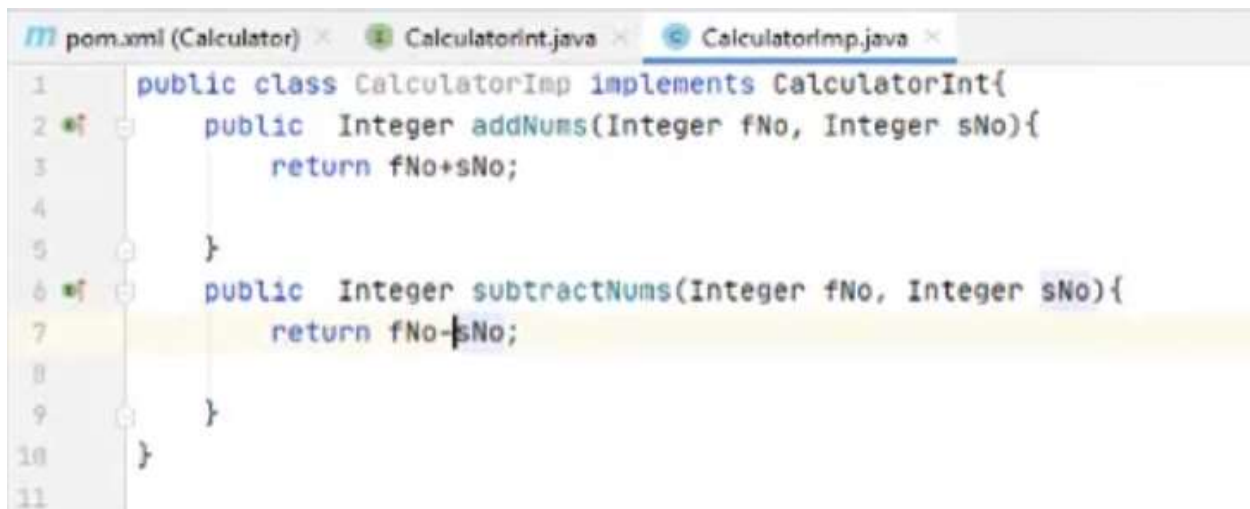


Write two abstract methods in interface



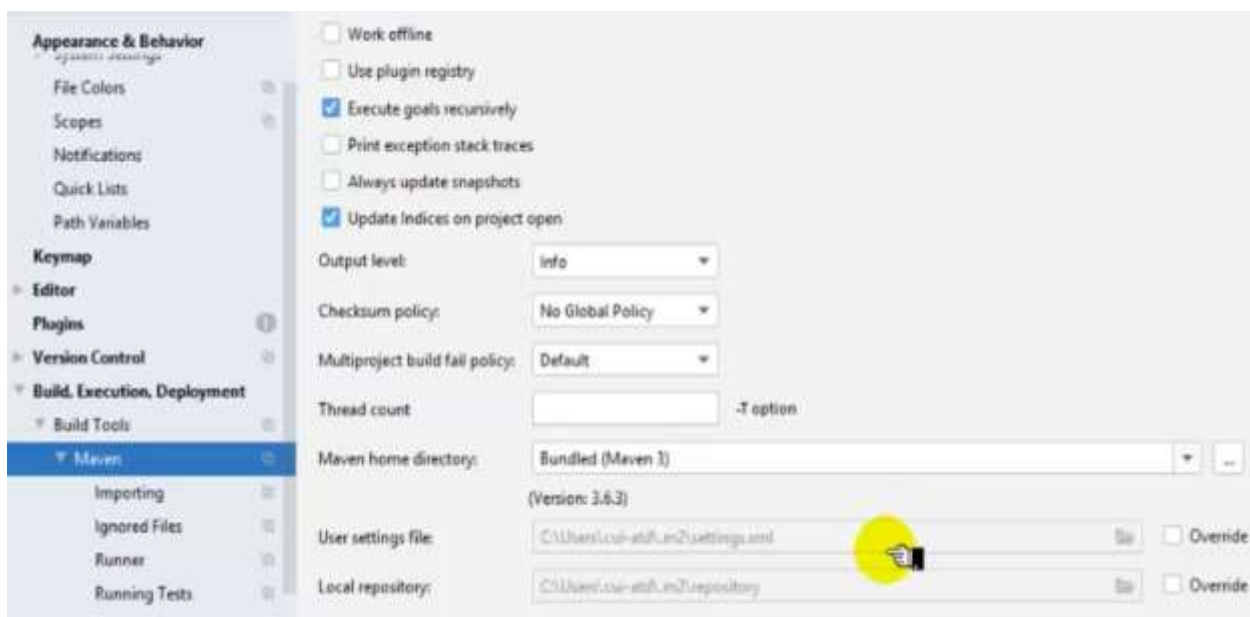
Implement this interface and give definition to the methods





```
1 public class CalculatorImp implements CalculatorInt{
2     public Integer addNums(Integer fNo, Integer sNo){
3         return fNo+sNo;
4     }
5
6     public Integer subtractNums(Integer fNo, Integer sNo){
7         return fNo-sNo;
8     }
9 }
10
11 }
```

Now open the build tool Maven



Provide your own path where the project is stored instead of default one



Output level:	Info	
Checksum policy:	No Global Policy	
Multiproject build fail policy:	Default	
Thread count		-T option
Maven home directory:	Bundled (Maven 3)	
	Bundled (Maven 3)	
	D:/JavaProjects/apache-maven-3.6.3	
User settings file:		Override
Local repository:	C:\Users\cur-stu\m2\repository	Override

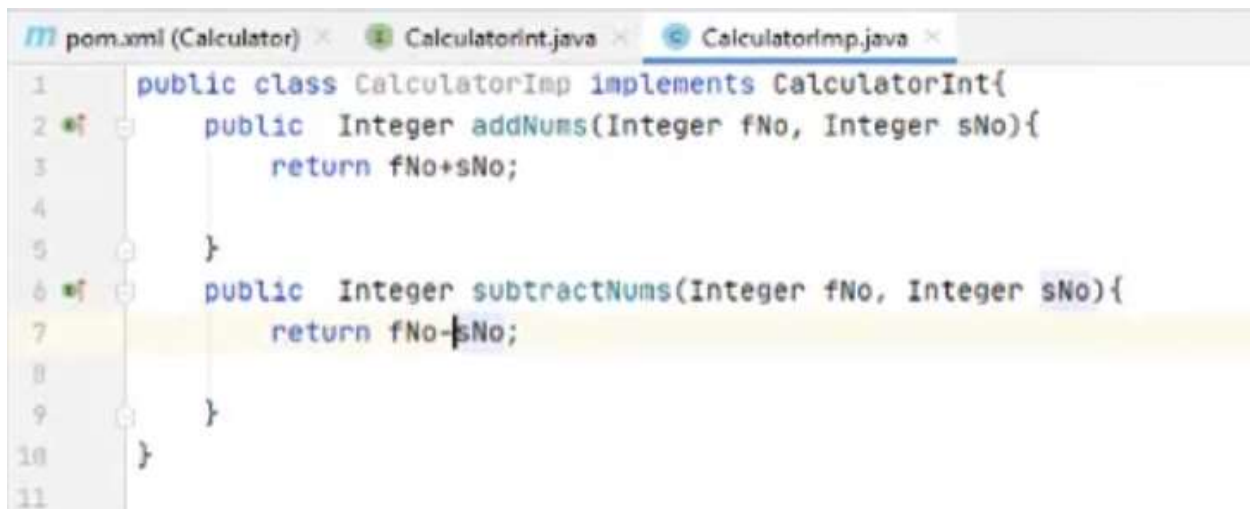
Now in setting file, provide maven config location and change local repos to maven repos

Maven home directory:	D:/JavaProjects/apache-maven-3.6.3	
	(Version: 3.6.3)	
User settings file:	D:\JavaProjects\apache-maven-3.6.3\conf\settings.xml	<input checked="" type="checkbox"/> Override
Local repository:	D:\JavaProjects\maven-repos	<input checked="" type="checkbox"/> Override

6.4 Exercises for lab

Q1. For the given program create test cases:





```
1 public class CalculatorImp implements CalculatorInt{
2     public Integer addNums(Integer fNo, Integer sNo){
3         return fNo+sNo;
4     }
5
6     public Integer subtractNums(Integer fNo, Integer sNo){
7         return fNo-sNo;
8     }
9 }
10
11
```

Test Cases:

@Test

```
public void addTest(){
    Integer firstNo = 5;
    Integer secondNo = 10;
    Integer result = 0;
    result = calculator.add(firstNo,secondNo);
    Integer expected = 15;
    Assertions.assertEquals(expected,result);
}
```

@Test

```
public void subtractTest(){
    Integer firstNo = 8;
    Integer secondNo = 3;
    Integer result = calculator.subtract(firstNo,secondNo);
    Integer expected = 5;
    Assertions.assertEquals(expected,result);
}
```

Q2. Create a program to evaluate triangle based on sides and nominate its type.

Program



Interface:

```
public interface ITriangle {  
  
    public TriangleType classify(Integer a,Integer b,Integer c);  
}
```

Class:

```
package triangle;  
  
public class Triangle implements ITriangle {  
  
    public TriangleType classify(Integer a,Integer b,Integer c){  
  
        Integer triangle = 0;  
  
        if (a <= 0 || b <= 0 || c < 0){  
            return TriangleType.INVALID;  
        }  
  
        if (a == b){  
            triangle = triangle + 1;  
        }  
        if (b == c){  
            triangle = triangle + 2;  
        }  
        if (a == c){  
            triangle = triangle + 3;  
        }  
  
        if (triangle == 0){  
            if ((a + b) < c || (b + c) < a || (a + c) < b){  
                return TriangleType.INVALID;  
            }else{  
                return TriangleType.SCALENE;  
            }  
        }  
        if (triangle > 3){
```



```

        return TriangleType.EQUILATERAL;
    }
    if (triangle == 1 && (a + b) > c){
        return TriangleType.ISOSCELES;
    }
    if (triangle == 2 && (b + c) > a){
        return TriangleType.ISOSCELES;
    }
    if (triangle == 3 && (a + c) > b){
        return TriangleType.ISOSCELES;
    }

    return TriangleType.INVALID;
}

}

```

```
package triangle;
```

```
public enum TriangleType {
    INVALID, SCALENE, ISOSCELES, EQUILATERAL;
}

```

Test Cases:

```
public class TriangleTest {

    ITriangle triangle;

    @BeforeAll
    public void initialize(){

        triangle = new Triangle();
    }

    @BeforeEach
    public void beforeEach(){

```



```

        System.out.println("Before Running Test Case");
    }

    @AfterEach
    public void AfterEach(){
        System.out.println("After Running Test Case");
    }

    @AfterAll
    public void allTestCaseCompleted(){
        System.out.println("All Test Cases Completed..!");
    }

    @Test
    public void classifyInvalid(){

        Assertions.assertEquals(TriangleType.INVALID,triangle.classify(0,0,0));

    }

    @Test
    public void classifyEquilateral(){
        Assertions.assertEquals(TriangleType.EQUILATERAL,triangle.classify(5,5,5));

    }

    @Test
    public void classifyScalene(){
        Assertions.assertEquals(TriangleType.SCALENE,triangle.classify(2,3,4));

    }

    @Test
    public void classifyIsosceles(){
        Assertions.assertEquals(TriangleType.ISOSCELES,triangle.classify(5,5,9));

    }

}

```



6.5 Homework:

- a. Create a program of prime numbers and test whether a number is prime or not
- b. Create a program for hours and minutes, user inputs minutes and program return values in the form of hours: minutes. Create possible test cases for the following scenario



Lab # 07

JUnit Test Suites



Lab 7: Advance JUnit and Test Suites, Assertions and Annotations

7.1 Objective:

To learn JUnit assert method

To learn about annotation used in JUnit

To focus on test suites

7.2 Scope:

After this lab students will be able to learn about assert method, annotations usage and creation of test suites.

7.3 Useful Concept:

7.3.1 Annotations:

There are different annotations that we can use to define callback methods at various stages of test cases execution

- ☐ @BeforeAll
- ☐ @BeforeEach
- ☐ @AfterEach
- ☐ @Ignore
- ☐ @RepeatedTest
- ☐ @DisplayName
- ☐ @ParameterizedTest
- ☐ @ValueSource
- ☐ @Disabled

7.3.2 Assertions:

All the assertions are in the Assert class

- ☐ assertEquals(boolean expected, boolean actual)
- ☐ assertSame(object1, object2)
- ☐ assertNotSame(object1, object2)
- ☐ assertNotNull(Object object)



- ❑ `assertNull(Object object)`
- ❑ `assertTrue(condition)`
- ❑ `assertArrayEquals(expectedArray, resultArray)`

7.4 Examples:

Repeat the same code as per previous class but now with classes of Assertion plus annotations needed.

```
package calculator;
```

```
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.TestInstance;
import org.junit.Test;
```

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class CalculatorTest {
```

```
    Calculator calculator;
```

```
    @BeforeAll
    public void initialize(){
        calculator = new CalculatorImpl();
    }
```

```
    @BeforeEach
    public void beforeEachTest(){
        System.out.println("Before Each Test Case");
    }
```

```
    @AfterEach
    public void afterEachTestCase(){
        System.out.println("After Each Test Case");
    }
```

```
    @Test
```




```

public void addTest(){
    Integer firstNo = 5;
    Integer secondNo = 10;
    Integer result = 0;
    result = calculator.add(firstNo,secondNo);
    Integer expected = 15;
    Assertions.assertEquals(expected,result);
}

@Test
public void subtractTest(){
    Integer firstNo = 8;
    Integer secondNo = 3;
    Integer result = calculator.subtract(firstNo,secondNo);
    Integer expected = 5;
    Assertions.assertEquals(expected,result);
}

@Test
public void multiply(){
    Integer firstNo = 9;
    Integer secondNo = 10;
    Integer result = calculator.multiply(firstNo,secondNo);
    Integer expected = 90;
    Assertions.assertEquals(expected,result);
}

@Test
public void divideTest(){
    Integer firstNo = 10;
    Integer secondNo = 2;
    Integer result = calculator.divide(firstNo,secondNo);
    Integer expected = 5;
    Assertions.assertEquals(expected,result);
}
}

```

The output will be like:



Before Each Test Case
After Each Test Case
Before Each Test Case
After Each Test Case
Before Each Test Case
After Each Test Case
Before Each Test Case
After Each Test Case

7.5 Exercises for Lab:

Triangle Classification testing on the basis of sides

```
package triangle;
```

```
import org.junit.jupiter.api.AfterAll;  
import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.BeforeAll;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.Test;  
import org.junit.jupiter.api.TestInstance;
```

```
public class TriangleTest {  
  
    ITriangle triangle;  
  
    @BeforeAll  
    public void initialize(){  
  
        triangle = new Triangle();  
    }  
  
    @BeforeEach  
    public void beforeEach(){  
        System.out.println("Before Running Test Case");  
    }  
}
```



```

@AfterEach
public void AfterEach(){
    System.out.println("After Running Test Case");
}

@AfterAll
public void allTestCaseCompleted(){
    System.out.println("All Test Cases Completed..!");
}

@Test
public void classifyInvalid(){

    Assertions.assertEquals(TriangleType.INVALID,triangle.classify(0,0,0));

}

@Test
public void classifyEquilateral(){
    Assertions.assertEquals(TriangleType.EQUILATERAL,triangle.classify(5,5,5));

}

@Test
public void classifyScalene(){
    Assertions.assertEquals(TriangleType.SCALENE,triangle.classify(2,3,4));
}

@Test
public void classifyIsosceles(){
    Assertions.assertEquals(TriangleType.ISOSCELES,triangle.classify(5,5,9));
}

}

```

Test Suite Code:



```

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

import calculator.CalculatorTest;
import triangle.TriangleTest;

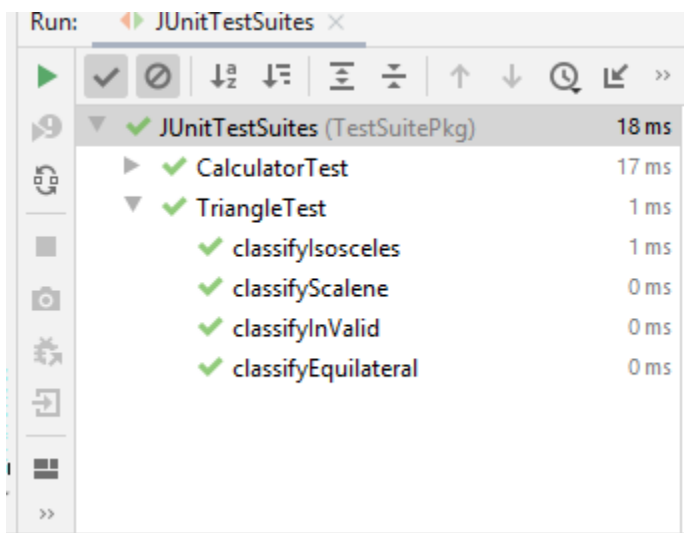
@RunWith(Suite.class)
@Suite.SuiteClasses({TriangleTest.class, CalculatorTest.class})
public class JUnitTestSuite {

}

```

In this case both classes of tests are run together and multiple classes can be executed in this way.

Output will be like:



7.6 Homework:

Make a test case for two different classes, one containing a code whether a number is prime or not, second class can contain whether a number is even or odd. Make a test suite of both test classes.



Lab # 08

Selenium Introduction



Lab 8: Selenium Introduction

8.1 Objective:

Objective of this lab is to learn about automated test cases driven by Selenium Selenium WebDriver and Selenium based Automated Testing

8.2 Scope:

Automated Test Cases for Web pages and Websites
Selenium Web Driver usage

8.3 Useful Concept:

Web Application Testing is a software testing technique, adopted to test applications, that are hosted on the web. It can be divided into different subcategories: Performance, Functionality, Usability, Interface, Compatibility and Security Testing. In this lab we are mainly focusing on functionality testing. The main goal is to get an idea, on how to write unit tests covering different functionalities of the system. This lab has been fully evaluated with IntelliJ and Mozilla Firefox on Windows and Mac, so we strongly recommend using IntelliJ and Mozilla Firefox for this lab

8.4 Examples:

System under test

System under test is a minified Online Store. As an end user (<https://st-online-store.herokuapp.com>) you can buy some items and as admin (<https://st-online-store.herokuapp.com/admin>) you can add and modify items in the store. To access the system as admin, you can create an account by yourself.

The given Store is an unrealistic toy application with very limited functionality. For example: Don't worry about the quantities in the store, there are always infinite supplies.

- o If using filter 'Other' on end user page, system shows items from categories 'Sunglasses' and 'Other'.*
- o After checkout, product total sum on order page (after inserting order details) gets multiplied with the quantity of that item. Meaning, if purchasing only 1 of different items, everything is calculated correctly. Purchasing 2 of the same item -> sum gets doubled.*
- o When editing a product on admin side, category gets always changed to 'Books'.*



Setting up the Web Application with Heroku

For the testing part of this lab we are going to deploy our own web application from given source code with Heroku.

Download "Application Source Code.zip" from courses page and unzip it

Create Heroku account <https://signup.heroku.com/> (Primary Development Language: Ruby)

Download and install JRuby 9.1.8.0

<http://jruby.org/files/downloads/9.1.8.0/index.html> (.exe version)

Open command line and type: `"jruby -S gem install bundler"`

Download and install heroku CLI

<https://devcenter.heroku.com/articles/getting-started-with-jruby#set-u>

Restart command line and go to the folder of your application source code (`"cd C:\Users\...\OnlineStoreApplication"`)

Type into command line:

`"git init"`

`"git add ."`

`"git commit -m 'First Application Commit'"`

`"heroku login"` and then type in your credentials

`"heroku create"` – creates heroku repository

`"git push heroku master"` – deploys committed files to heroku

`"heroku run rake db:migrate"` – migrates the database

`"heroku run rake db:seed"` - seeds the database with initial products

`"heroku open"` and you have an application up and running!

For more information: <https://devcenter.heroku.com/articles/getting-started-with-jruby#introduction>

If students get an error like this during heroku command, it means heroku and ruby versions don't match. Fix is to delete folder "C:/Users/Name/AppData/Local/heroku" (exact path depends on students computer)

Error: Cannot find module 'supports-color'

at Function.Module._resolveFilename (module.js:527:15)

at Function.Module._load (module.js:476:23)

at Module.require (module.js:568:17)

at require (internal/module.js:11:18)

at Object.<anonymous> (C:/Users/Name/AppData/Local/heroku/client/6.16.11-

b6217f5/node_modules/chalk/index.js:4:21)

at Module._compile (module.js:624:30)

at Object.Module._extensions..js (module.js:635:10)



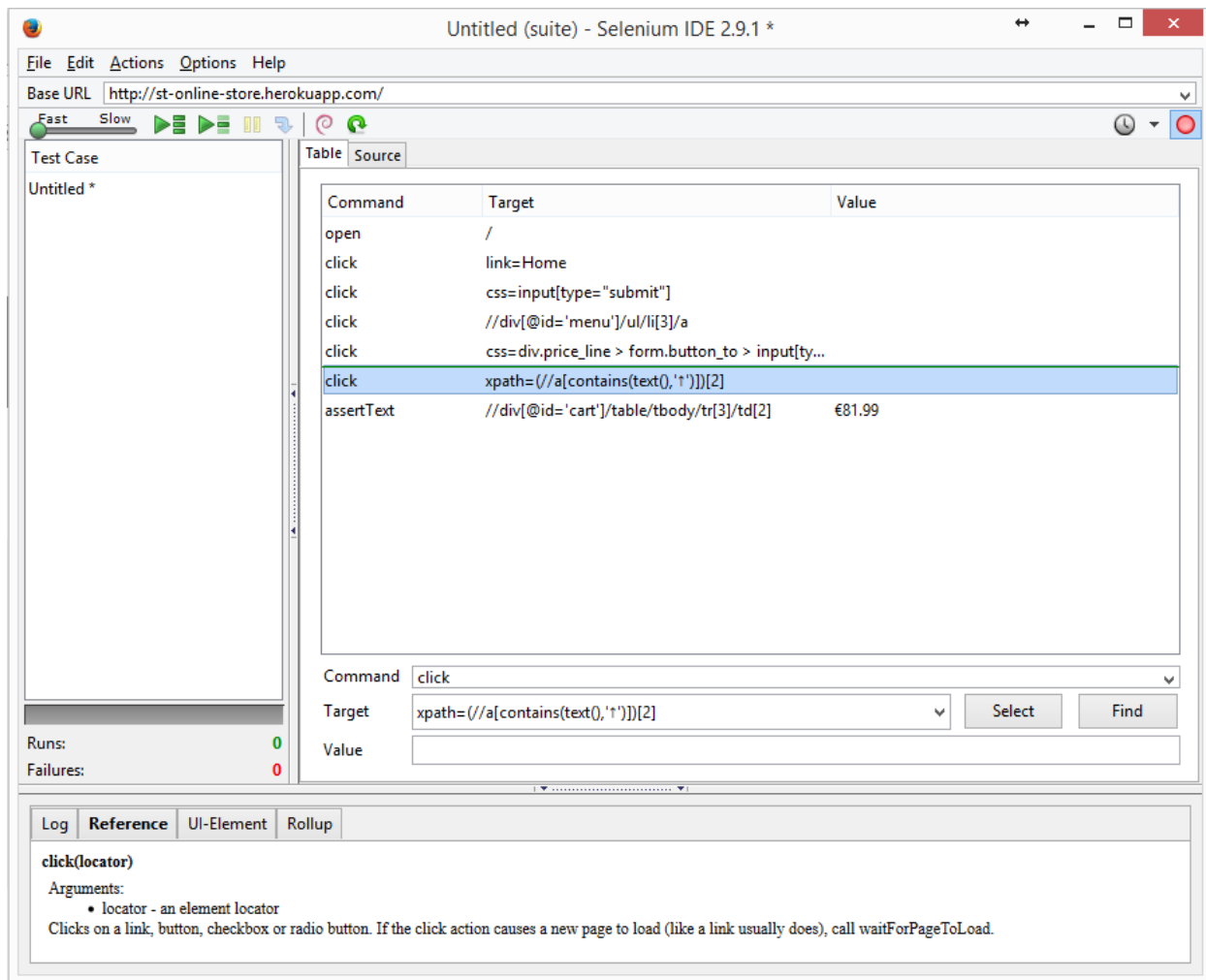
at Module.load (module.js:545:32)
at tryModuleLoad (module.js:508:12)
at Function.Module._load (module.js:500:3)

Selenium is a popular tool for automating web application tests. It's simple to use and the tests can be run against most web browsers.

Selenium IDE

Note: It's not essential to install web-based IDE for this Lab.

When using the IDE, Selenium will record the steps made in the browser and then the test can be rerun. But the way Selenium marks data up in the IDE is not very user friendly. See the picture below.



This test case adds two different products to the cart, increases the quantity of one of them, and then tries to assert that the price is correct. But the exact products, their quantities and prices, remain unclear. And how can you be sure, the price is calculated correctly?

Using the IDE for testing might seem simpler, but using proper unit tests gives more control over input/output and visible data. It's also easier to modify the tests and they are more human readable. Other than that, the IDE can only be used with old versions of Firefox, while WebDriver based tests can be run with multiple browsers and they're also compatible with latest versions.

Link to Selenium documentation: <http://www.seleniumhq.org/docs/>

Selenium WebDriver Tool Setup in IntelliJ

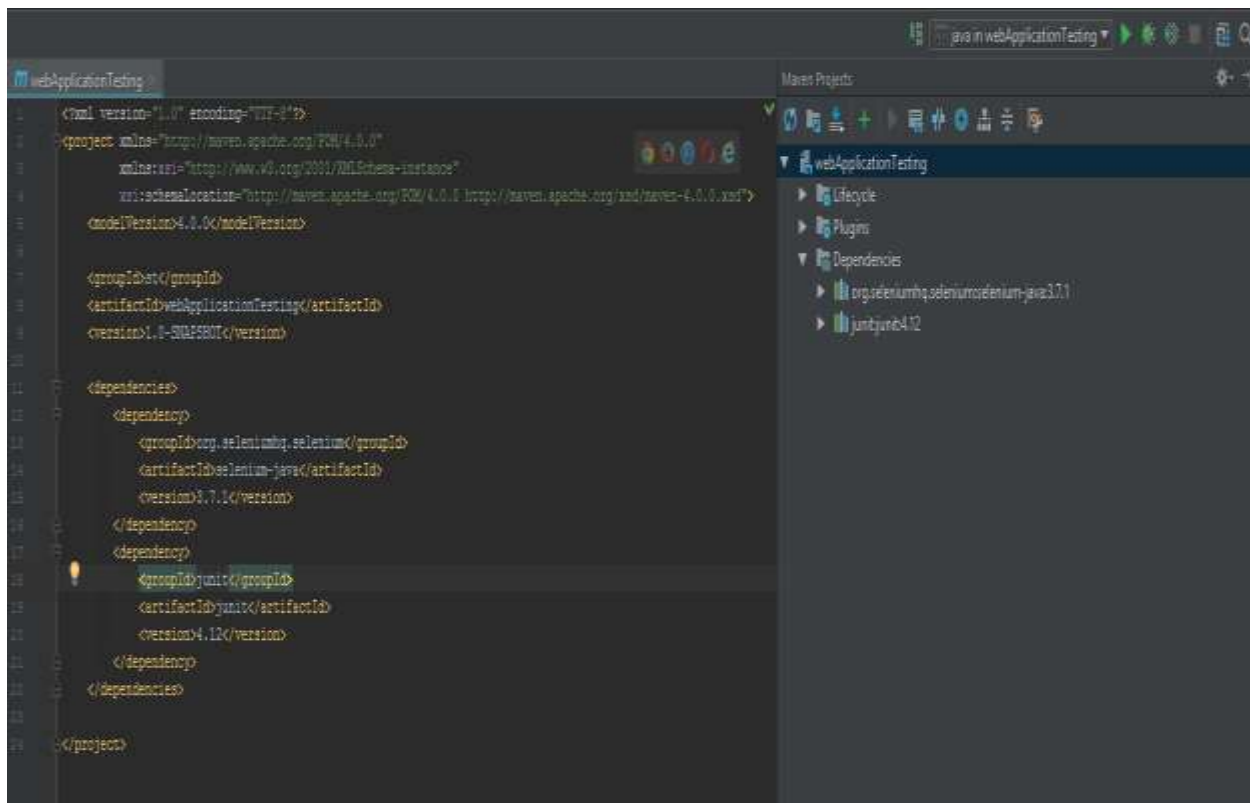
Important! - For the homework you must use Selenium Webdriver and write your own scripts!

Download Maven project "*TestCode.zip*" from courses page and unzip it.

Open project (New -> Project from existing sources -> Click on the "*pom.xml*" file from the downloaded project source)

Refresh the project (View -> Tool windows -> Maven projects -> Click on "*Reimport All Maven Projects*"). You should see something similar to picture below





Drivers for other browsers can be found here:

<http://www.seleniumhq.org/download/>

- Also add the path of your driver to your class setUp() method in “Testhelper” class.

Add your website’s URL to “Testhelper” class.

8.5 Exercises for lab

Example test code

Example test code is located in the given project

In given project the class “TestHelper” has setUp()/tearDown() methods, so if your test class extends this class you don’t have to write them in all test classes.

The setUp() and tearDown() methods get called before/after every test case. SetUp() initializes the driver, that the test is going to use and tearDown() makes sure the driver gets closed, even if the test case fails.

In-class exercise



Try to get the given code up and running, so that the first basic test `titleExistsTest()` passes.

Fill in `loginLogoutTest()` and `login mehtod` in `TestHelper`, so that `theloginLogoutTest()` passes correctly. (First you have to make an admin account.)

Now write a test case, where you make sure, that one can't log in with a false password.

8.6 Homework:

Write unit tests covering all the functionalities listed below (both Admin and End-user functionality). Make sure to name the test cases properly so that during grading it is obvious for the TA's what functionality is supposed to be tested by a certain test case.

- As admin you can:

1. Register an account
2. Login to the system
3. Logout from the system
4. Delete an account
5. Add products
6. Edit products
7. Delete products

- As end user you can:

1. Add products to the cart
2. Increase/decrease the quantity of products in the cart
3. Delete items one by one from the cart
4. Delete the entire cart
5. Search products by name on the home page and filter them by categories
6. Purchase items



Lab # 09

Automation Testing Using Selenium



Lab 9: Automation Testing Using Selenium

9.1 Objective:

Automated Testing using Selenium and firefox plugin/ extension

Gecko Driver for firefox

9.2 Scope:

Testing of websites and webpages using Selenium web Driver

Automated procedures to remotely access websites for testing.

9.3 Useful Concept:

Selenium-WebDriver

- A piece of program
 - Control the browser by programming
 - More flexible and powerful
 - Selenium-WebDriver supports multiple browsers in multiple platforms
 - Google Chrome 12.0.712.0+
 - Internet Explorer 6+
 - Firefox 3.0+
 - Opera 11.5+
 - Android – 2.3+ for phones and tablets
 - iOS 3+ for phones
 - iOS 3.2+ for tablets
-
- WebDriver is designed to providing a simpler and uniformed programming interface
 - Same WebDriver script runs for different platforms
 - Support multiple programming language:
 - Java, C#, Python, Ruby, PHP, Perl...
 - It's efficient



- WebDriver leverages each browser's native support for automation.

What Selenium Can Do:

- A solution for the automated testing
 - Simulate user actions
 - Functional testing
 - Create regression tests to verify functionality and user acceptance.
 - Browser compatibility testing
 - The same script can run on any Selenium platform
 - Load testing
 - Stress testing

How to use Selenium Web Driver:

- (1) Go to a page
- (2) Locate an element
- (3) Do something with that element

.....

- (i) Locate an element
- (i+1) Do something with that element
- (i+2) Verify / Assert the result

9.4 Examples

Verify the page title:

```
public static void main( String[] args )
{
    // Create a new instance of the Firefox driver
    WebDriver driver = new FirefoxDriver();
    // (1) Go to a page
    driver.get("http://www.google.com");
```



```

// (2) Locate an element
WebElement element = driver.findElement(By.name("q"));

// (3-1) Enter something to search for
element.sendKeys("Purdue Univeristy");

// (3-2) Now submit the form. WebDriver will find the form for us from the element
element.submit();

// (3-3) Wait up to 10 seconds for a condition
WebDriverWait waiting = new WebDriverWait(driver, 10);
waiting.until( ExpectedConditions.presenceOfElementLocated( By.id("pnnext") ) );

// (4) Check the title of the page
if( driver.getTitle().equals("purdue univeristy - Google Search") )
    System.out.println("PASS");
else
    System.err.println("FAIL");

//Close the browser
driver.quit();
}

```

To Locate an Element:

- **By id**
 - HTML: <div id="coolestWidgetEvah">...</div>
 - WebDriver:

```
driver.findElement( By.id("coolestWidgetEvah") );
```

- **By name**
 - HTML: <input name="cheese" type="text"/>
 - WebDriver: driver.findElement(By.name("cheese"));



- By Xpath

- HTML

```
<html>
```

```
    <input type="text" name="example" />
```

```
    <input type="text" name="other" />
```

```
</html>
```

- WebDriver: `driver.findElements(By.xpath("//input"));`
 - There are plug-ins for firefox/chrome to automatically display the Xpath

9.5 Exercises for lab:

Facebook Login:

```
package selenium;
```

```
import org.openqa.selenium.By;
```

```
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.firefox.FirefoxDriver;
```

```
public class FacebookLogin {
```

```
    public static void main(String[] args) {
        try {
```

```
            System.setProperty("webdriver.gecko.driver", "geckodriver.exe");
```

```
            WebDriver driver = new FirefoxDriver();
```

```
            Thread.sleep(2000);
```

```
            driver.manage().window().maximize();
```

```
            driver.get("https://www.facebook.com");
```

```
            Thread.sleep(3000);
```

```
            driver.findElement(By.id("email")).sendKeys("yourusername@gmail.com");
```

```
            driver.findElement(By.id("pass")).sendKeys("password");
```

```
            driver.findElement(By.name("login")).click();
```

```
//            driver.quit();
```




```

    } catch (Exception exception) {
    }
}
}

```

Maven File POM.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>SeleniumExamples</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>3.141.59</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>

```



```
</plugins>
</build>
</project>
```

Getting Title of webpage:

```
package com.softech.selenium;
```

```
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.jupiter.api.BeforeEach;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
```

```
public class SeleniumTest {
```

```
    private static FirefoxDriver webDriver;
```

```
    @BeforeClass
```

```
    public static void initialize() {
```

```
        System.setProperty("webdriver.gecko.driver",
"D:\\JavaProjects\\WebDrivers\\FireFox\\geckodriver.exe");
        webDriver = new FirefoxDriver();
```

```
    }
```

```
    @Test
```

```
    public void startWebDriver() {
```

```
//    System.setProperty("webdriver.gecko.driver",
"D:\\JavaProjects\\WebDrivers\\FireFox\\geckodriver.exe");
        webDriver.navigate().to("https://www.seleniumsimplified.com/");
        Assert.assertTrue("Title should start differently", webDriver.getTitle().startsWith("Selenium
Simplified"));
        // Assert.assertTrue("Title should start differently", webDriver.getTitle().endsWith("Made
Simple"));
```

```
    }
```



```

@Test
public void start() {
//    System.setProperty("webdriver.gecko.driver",
"D:\\JavaProjects\\WebDrivers\\FireFox\\geckodriver.exe");
    webDriver.navigate().to("https://www.seleniumsimplified.com/");
    Assert.assertTrue("Title should start differently", webDriver.getTitle().endsWith("Made
Simple"));
    // Assert.assertTrue("Title should start differently", webDriver.getTitle().endsWith("Made
Simple"));

}
}
@AfterClass
public static void closeBrowser(){

    webDriver.close();

}

}

```

9.6 Homework:

Visit eBay website and buy an item and add the item to the cart in a sequence and checkout using Selenium Driver.



Lab # 10

Debugging



Lab 10: Debugging

10.1 Objective:

The purpose of this lab is to learn how to debug using Eclipse and/or IntelliJ built-in debugger tools and how to combine that with several debugging heuristics. As there are many ways to approach problems, this lab aims to give an overview of some of them in combination with debugger tools.

10.2 Scope:

How to debug the code?

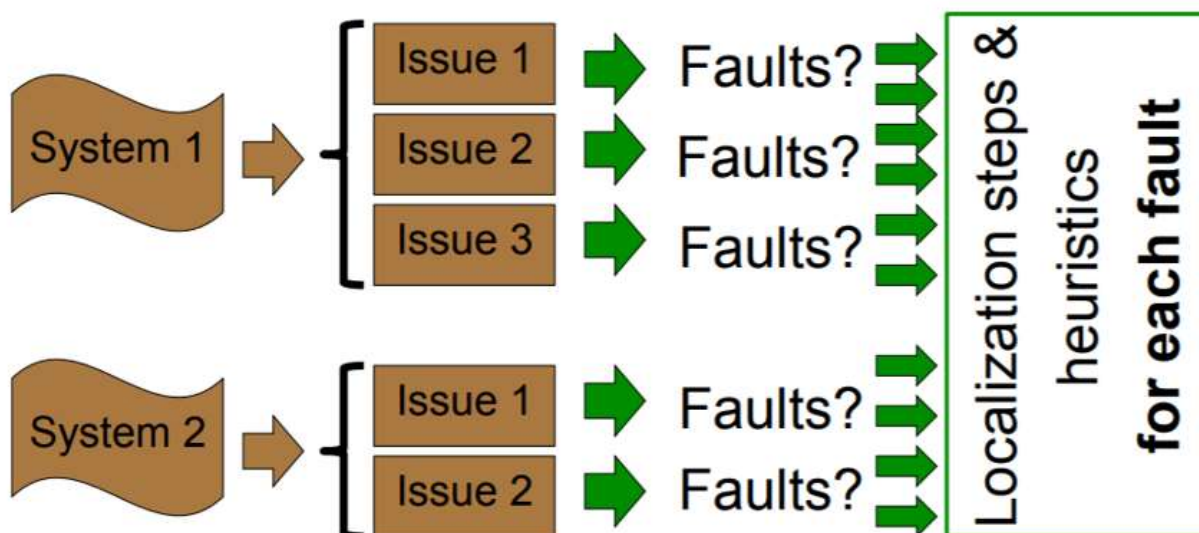
How to set breakpoints?

How to use watchpoints?

How to step into and step over the code?

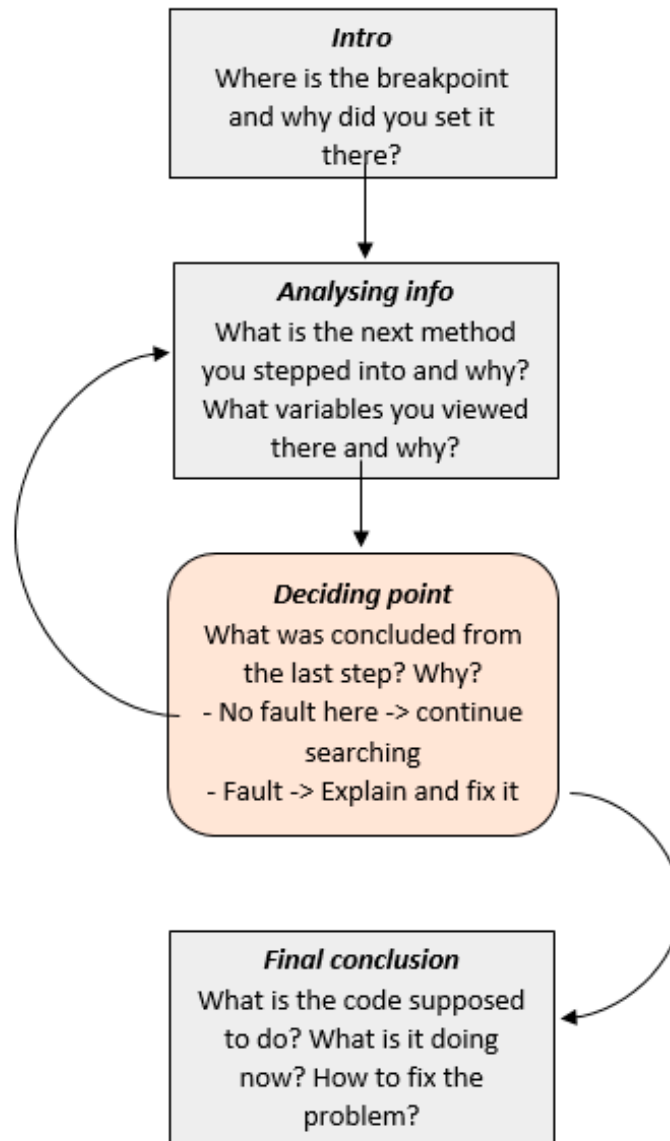
10.3 Useful Concept:

Debug Process



Debugging Process:





Eclipse and IntelliJ buttons and meanings:

- **Step Into** – Step into whatever function call is on the line you are at. If there is no function call there, program will continue to the next line.
- **Step Over** – Step over the line you are in and onto the next one

Drop to Frame (Eclipse) – Drop Frame (IntelliJ). In Eclipse you go back to the start of the function you are in. In IntelliJ you go back to the call of the function you are in.

- **Step Return** – Step to the returned value of the function you are in (or the line after this function)



was called).

- **Resume** – continue running the program as normal until the next breakpoint is found.
- **Variables tab** – this is your most useful tool, there you can see all the current values in the function you're in – integers, strings, arrays etc. In both IntelliJ and Eclipse, you can see an array better if you go deeper in the variables tab. In both open the little arrow of the array in the variables tab, in Eclipse click on the array (such as "heapList"), then you can see the array at the bottom of the variables tab. In IntelliJ you need to right-click on the list that opened, choose "View as" and "toString".

Note: In IntelliJ you can see the variables in the editor on the lines you have already passed. In Eclipse you see them only in the variables tab

List of Debugging Heuristics:

No.	Remark	Why is it useful
1	Whenever you reach a new function call, step into it.	If you have reached a function call and don't know the source of the failure yet, it is likely that the problem is deeper in the code. This means you need to check whether the body for this function works as it should.
2	Whenever you reach a new function call that takes a parameter, check if the parameter given is logical.	The cause for a failure may just be that all the functions work properly, just that one of them is called with incorrect parameter value(s).
3	Check the documentation of the code for hints.	The comments in the code from the author can be very useful in determining what the code is supposed to do and how.
4	Change the input code from main method	Reducing input can make it easier to follow the run of the code or even cause an error that makes it easier to localise the problem. Changing it into something that you know the correct output for can also make it easier to make sure the code does what it's supposed to.
5	If the data is in a structure that is difficult to visualize, construct it on a paper while debugging	If the data is a tree, a matrix or some other structure that is difficult to keep visualizing all the time, it can be useful to draw out (with pen and paper) the state you have in your debugger while debugging (using the variables view). That makes it easier to be sure that you did not miss anything from the variables and states you were viewing.
6	Change the value of a variable in the debugger. ("Change value" command)	This can give you a more varied overview of the ways the code acts with different values without having to restart the debugger from the start with a new original call parameter.
7	Run only part of the program at a time, if possible.	If you run the full program you will have a lot of code to debug through, which does not save you much time when



		compared to exhaustive solving. Therefore find a way to only run some of the program and check that before continuing on with the rest.
8	If you already know something is going to work correctly in a loop, don't step through every iteration again.	You can skip for loops to a specific run of the cycle. This allows you to skip the runs of the loop that you already know will be correct and immediately get the run where you think a problem might be. To do this, set a breakpoint on the for loop line (You can do this even when the debugger is already running). In IntelliJ, right click on the breakpoint, write the iteration number in the Condition field (e.g. <code>i==3</code>), and then run the debugger. In Eclipse, right click on the breakpoint, choose Breakpoint properties, tick Conditional and write the condition in the field below (e.g. <code>i==3</code>).



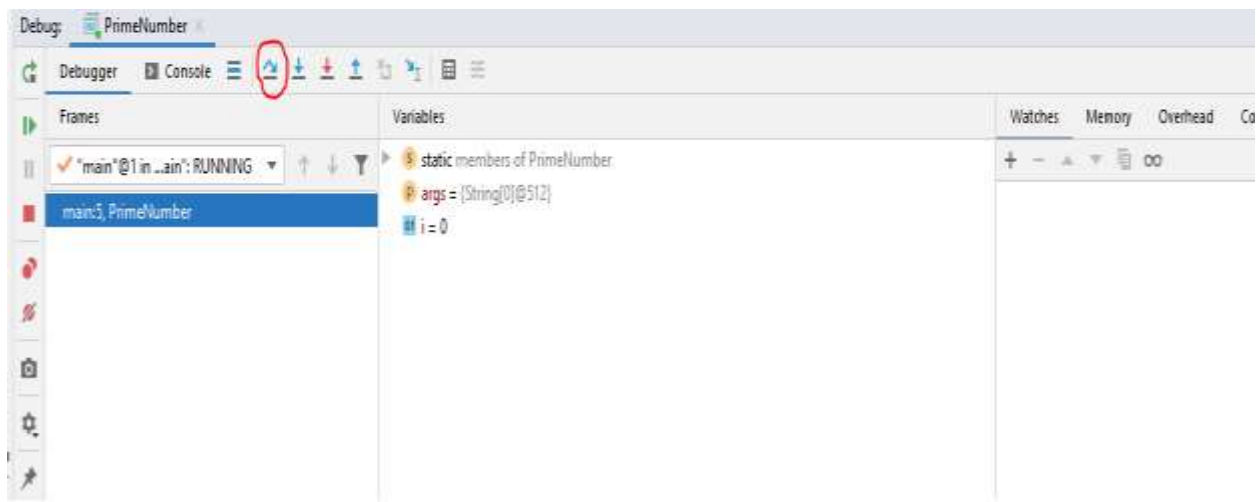
10.4 Examples:

Breakpoints

```
1  ► public class PrimeNumber {
2  ►   public static void main(String[] args) {
3      int i = 0;
4
5  ●   String primeNumbers = "";
6
7  ●   for (i = 1; i <= 100; i++) {
8      if (isPrime(i)) {
9  ●   primeNumbers = primeNumbers + i + ", ";
10     }
11     }
12  ●   System.out.println("Prime numbers from 1 to 100 are :");
13     System.out.println(primeNumbers);
14     }
15
16     static int res;
17
18     public static boolean isPrime(int i) {
19  ●   int counter = 0;
20     for (int num = i; num >= 1; num--) {
21         if (i % num == 0) {
22             counter = counter + 1;
23         }
24     }
25     if (counter == 2) {
26         return true;
27     }
28     return false;
29     }
30 }
```



Step Over:



Step Into:

The screenshot shows an IDE with a Java project. The code is as follows:

```
16 static int res;  
17  
18 public static boolean isPrime(int i) { i: 1  
19     int counter = 0; counter: 0  
20     for (int num = i; num >= 1; num--) { num: 1  
21         if (i % num == 0) { i: 1 num: 1  
22             counter = counter + 1;  
23         }  
24     }  
25     if (counter == 2) {  
26         return true;  
27     }  
28     return false;  
29 }  
30 }
```

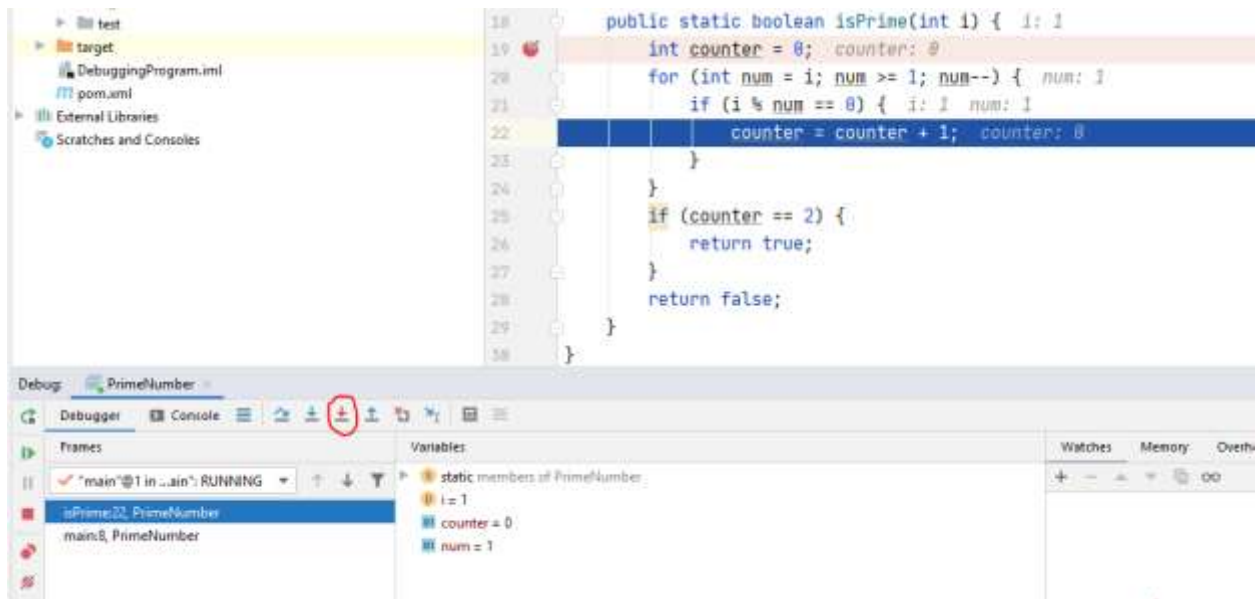
The debugger toolbar shows the 'Step Into' button (a blue arrow pointing down) circled in red. The 'Frames' pane shows the current frame is 'isPrime21, PrimeNumber'. The 'Variables' pane shows the following variables:

- static members of PrimeNumber:
 - i = 1
 - counter = 0
 - num = 1

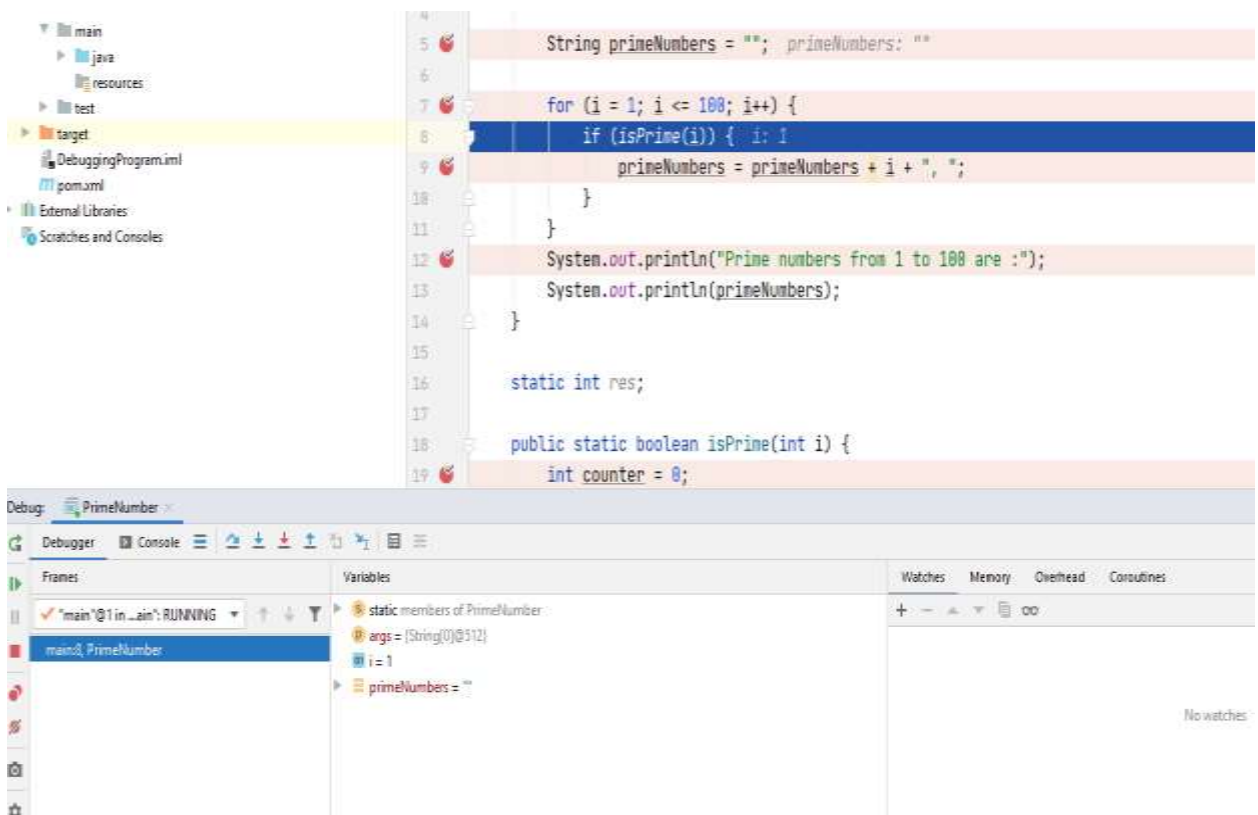
The 'Watches' pane is empty, showing 'No watches'.

Force Step Into:





Force Step Over:



10.5 Exercises for lab:

A **complete bug resolution** includes:

- the faulty code line
- the correction of the faulty code line
- the resolution process table with your work process and reasoning

Debug the program and according to the test cases table, make a report out of debugging.

// Java program for implementation of Heap Sort

```
public class HeapSort {
    public void sort(int arr[])
    {
        int n = arr.length;

        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

        // One by one extract an element from heap
        for (int i = n - 1; i > 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    void heapify(int arr[], int n, int i)
    {
        int largest = i; // Initialize largest as root
```



```

int l = 2 * i + 1; // left = 2*i + 1
int r = 2 * i + 2; // right = 2*i + 2

// If left child is larger than root
if (l < n && arr[l] > arr[largest])
    largest = l;

// If right child is larger than largest so far
if (r < n && arr[r] > arr[largest])
    largest = r;

// If largest is not root
if (largest != i) {
    int swap = arr[i];
    arr[i] = arr[largest];
    arr[largest] = swap;

    // Recursively heapify the affected sub-tree
    heapify(arr, n, largest);
}
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver code
public static void main(String args[])
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = arr.length;

    HeapSort ob = new HeapSort();
    ob.sort(arr);

```



```

        System.out.println("Sorted array is");
        printArray(arr);
    }
}

```

Solution:

Heap

Table 1 - Heap bug no 1 – issue 1 – (To be completed by the TA with the students in the lab – not to be graded or included in lab report)

Method	Variables	<i>Intro and Analysing info</i>	<i>Deciding point and Final conclusion</i>	Heuristics used
Main	Heap heap = new Heap(heapList);	Since the elements are all there in the first list, this is the first point where something is wrong.	This was a good entry point to the main program class as this is the highest level and allows me to go further into the program.	1,2,3,5
heapify()	Heap - [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0]; lastIndex – 12; parentIndex – 6	Need to check if the indexes that are used to heapify the list are calculated correctly.	Since I know that in heapsort the parent index should be 5 here, but it is 6, there must be something wrong in calculating this index.	
elemParent-Index (int index)	Heap - [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0]; index – 12	I found from last method, this one returns a wrong index so I need to see what is happening inside it.	As I know that in heapsort parent indexes are calculated with $(currentIndex-1)/2$ rounded down. I can see that is not the case here. Change <i>return index/2</i> to <i>return (index-1)/2</i>	

Issue 1 bugs:



This is an example of how the students should also list the faulty lines with the corrections:

- Line 71, wrong: `return index/2`, correct: `return (index-1)/2`

The rest of the list:

- Lines 35-36, wrong: `if (valueList.size() > (2*index)-1){`
`return (2*(index)-1);`
correct: `if (valueList.size() > (2*index)+1){`
`return (2*(index)+1);`
- Line 89, wrong: `valueList.set(index,rightChild);`, correct: `valueList.set(index,leftChild);`
- Line 96, wrong: `empty`, correct: `BubbleDown(rightIndex);`

Table 2 - Heap bug no 2 – issue 1

Method	Variables	Intro and Analysing info	Deciding point and Final conclusion	Heuristics used
heapify	heap – [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0]; lastIndex-12, parentIndex-5	Since this function calls another function several times, I need to make sure it gives the correct values as parameters. As I already checked that <code>elemParentIndex</code> method is correct, I should check what this method does further on, that means checking the for loop.	Looking at the current state of the heap and the variable values I already deduced that parent index was correct and match heapsort logic (parent index is calculated with $i-1/2$, where i is child index, left child with $2 * j + 1$ and right child with $2 * j + 2$, where j is parent index)	1, 2, 3, 4
BubbleDown	Heap - [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0]; index – 5, currentElem – 8, leftChild - ?	This is the first method call that is found when continuing on from last bug and same entry point. Should check all the variables here as well as all the methods this one calls.	I should step into <code>leftElem(index)</code> function and check the calculations	



leftElem and leftIndex	Heap - [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0]; index – 5, leftChild - 4, leftIndex returned - 9	These methods use heapsort structure and index calculations; I need to make sure they are correct before I can continue.	Since leftElem does not calculate anything on its own, I need to check leftIndex. I know that index of left child in heapsort should be calculated with $(2 * \text{CurrentIndex} + 1)$ and I can see that it is calculated with $(2 * \text{currentIndex} - 1)$ I can conclude there is a bug here. Correct lines should be: <i>if (heap.size() > (2 * index) + 1){ return (2 * (index) + 1);</i> leftIndex returned should be 11, left child should be 1	
------------------------	--	--	---	--

Table 3 - Heap bug 3 – issue 1

Method	Variables	<i>Intro and Analysing info</i>	<i>Deciding point and Final conclusion</i>	Heuristics used
BubbleDown	Heap - [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0]	Need to check if the if-else branches are used correctly according to the heap given (with the indexes and values calculated)	Does not step into any branches – but the ones checked were checked correctly. All the element values and indexes are calculated correctly at this point (fixed in last 2 bugs)	2, 3, 5, 6, 8
BubbleDown	Heap - [1, 2, 5, 7, 6, 8, 11, 10, 3, 4, 9, 1, 0] index 4, currentElem = 6, leftChild = 4, rightChild = 9,	Program reached part of an if-else branch that has not been executed until now. This means I need to make sure this part is done correctly.	From the if-else checks this part is reached when the current element is larger than the left child but smaller than the right child. The program should swap these, which it does. It continues with the right child to make sure the	



	leftIndex = 9, rightIndex = 10		small element reaches the lowest part in the tree it can.	
BubbleDown	Heap - [1, 2, 5, 7, 9, 8, 11, 10, 3, 4, 6, 1, 0], index = 10, currentElem 6, leftChild = -1, rightChild = -1, leftIndex = -1, rightIndex = -1	Recursive call after the changes made in last method call (previous row in this table) should also be checked to make sure the element reaches the final destination.	The element was already in the correct position. Continue with sorting.	
BubbleDown	Heap - [1, 2, 5, 7, 9, 8, 11, 10, 3, 4, 6, 1, 0]; index = 3 currentElem = 7 leftChild = 10 rightChild = 3 leftIndex = 7 rightIndex = 8	Program reached part of an if-else branch that has not been executed until now. This means I need to make sure this part is done correctly.	In this part of the if-else, the current element is smaller than the left child but larger than the right one. This means it should be swapped with the left child. The program also changes the right child which means there is a bug here. <i>heap.set(index,rightChild);</i> should be <i>heap.set(index,leftChild);</i>	

10. 6 Homework

Learn Genetic algorithm and debug its code by seeding faults and repeat the session of class with report.



Lab # 11

Mutation Testing

Page 91 | 138



Lab 11: Mutation Testing

11.1 Objective:

The purpose of this lab is to explain why mutation testing is important and should be used as an addition to other software testing methods and introduce one of the mutation testing's tools PIT.

11.2 Scope:

After this lab, students will be able to learn:

1. Mutation testing
2. Tool as PIT
3. Mutants (Killed + Survived)

11.3 Useful Concept

Mutation Testing:

Mutation testing is a way of testing where bugs (mutations) are seeded into your program and then the tests are run. If the tests fail then the mutations are killed. If not then the mutations are alive. The quality of your tests can be estimated from the percentage of mutations killed.

Analysis Tool

The tool used in this lab is PIT mutation testing system which has a plugin for Eclipse. It applies a configurable set of mutation operators (or mutators) to the byte code generated by compiling your code.

11.4 Examples

Task 1: Playing Code Defenders online (~30 minutes)

To get the general idea of how the Pitest plugin works you will first play a game online.

Go to <http://code-defenders.org/> and create a user (takes less than a minute).

The Teacher will create a battleground where half of the students play as defenders and the other half as attackers.

To join the game, go to games -> Open games, scroll down to battlegrounds.

The Teacher will tell students the game ID and you can join as a defender or an attacker.

One of the desk mates should choose the defender role and the other one the attacker role. If you prefer to do your homework, alone choose the defender role.

Attacker: You will start the game by seeding a bug in the original code. A bug



can be one small change at a time (e.g. changing a variable name, changing one operator).

For example, the conditional boundary operator is changed:

```
52     public void goDown() {  
53         if (currentFloor > 0)  
54             currentFloor--;  
55     }  
  
52     public void goDown() {  
53         if (currentFloor < 0)  
54             currentFloor--;  
55     }
```

Defender: Your task is to write unit tests to kill

the mutants generated by the attackers. When there are no mutants alive, write a test that will kill a possible future mutant. Write at least 1, but not more than 2 assertions, no loops, no new methods, no calls to System.

An example of a test:

```
import org.junit.*;  
import static org.junit.Assert.*;  
  
public class TestElevator {  
    @Test(timeout = 4000)  
    public void test() throws Throwable {  
        Elevator e = new Elevator(10, 2);  
        e.addRiders(1);  
        assertEquals(1, e.getNumRiders());  
    }  
}
```

To see the changes made by other students simply refresh the page.



11.5 Exercises for lab:

Equivalent mutants

It is possible to create a mutant which is identical in functionality to the code, so no test can pass and fail on the mutated class.

For example, the following functions are identical in behavior, they are equivalent:

```
39     public void addRiders(int numEntering) {
40         if (numRiders + numEntering <= capacity) {
41             numRiders = numRiders + numEntering;
42         } else {
43             numRiders = capacity;
44         }
45     }

39     public void addRiders(int numEntering) {
40         if (numRiders + numEntering > capacity) {
41             numRiders = capacity;
42         } else {
43             numRiders = numRiders + numEntering;
44         }
45     }
```

If a defender believes that an attacker's mutant is equivalent, they can click the "Claim Equivalent" button on the mutant. After this, the attacker will see that their mutant was marked as equivalent. If the mutant is equivalent, they should accept it as equivalent.

If a defender believes that an attacker's mutant is equivalent, they can click the "Claim Equivalent" button on the mutant. After this, the attacker will see that their mutant was marked as equivalent. If the mutant is equivalent, they should accept it as equivalent.

For IntelliJ:

Set up IntelliJ IDE and install PIT plugin from

<https://plugins.jetbrains.com/plugin/7119-pit-mutation-testing-idea-plugin>

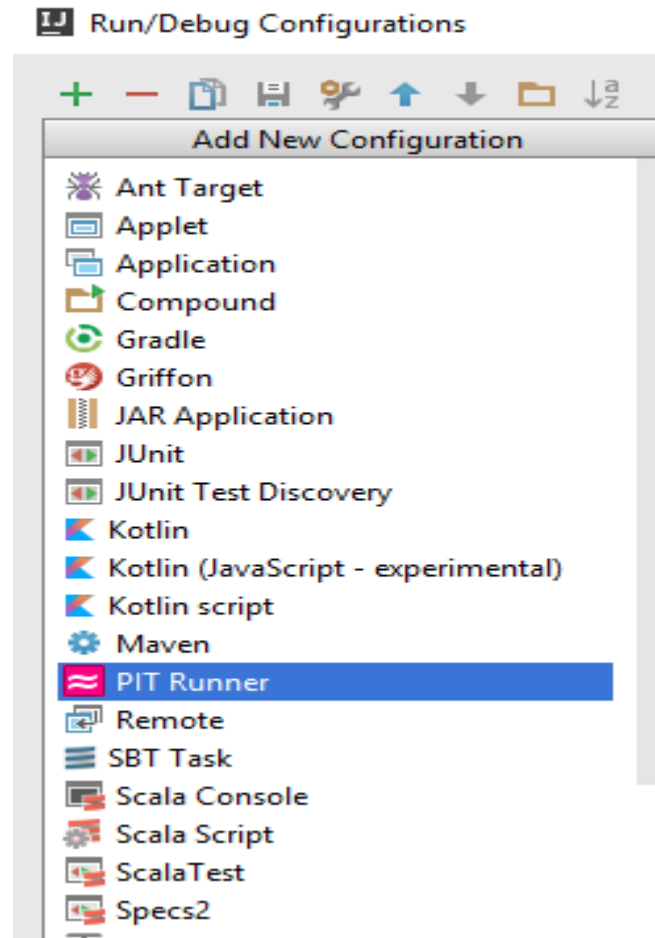
File -> Settings -> Plugins -> PIT mutation testing Idea plugin and import MinBinaryHeap from the course wiki page:



<https://courses.cs.ut.ee/2017/SWT2017/spring/Main/LabsPracticeSessions>.

Next, move tests to the same folder as main class (src/main). Add a new Run Configuration

Run -> Edit Configurations -> Add new configuration (green plus button)



Source directory should be the same as where the classes are. Click Apply and Ok.

PS! If you for some reason get an exception when running tests

(java.lang.NoClassDefFoundError: org/hamcrest/SelfDescribing) then add Hamcrest library to the classpath as shown here (underlined lines below):



```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <module type="JAVA_MODULE" version="4">
3    <component name="NewModuleRootManager" inherit-compiler-output="false">
4      <output url="file://$MODULE_DIR$/bin" />
5      <exclude-output />
6      <content url="file://$MODULE_DIR$">
7        <sourceFolder url="file://$MODULE_DIR$/src" isTestSource="false" />
8      </content>
9      <orderEntry type="sourceFolder" forTests="false" />
10     <orderEntry type="inheritedJdk" />
11     <orderEntry type="module-library">
12       <library name="junit4">
13         <CLASSES>
14           <root url="jar://$APPLICATION_HOME_DIR$/lib/junit-4.12.jar!/" />
15           <root url="jar://$APPLICATION_HOME_DIR$/lib/hamcrest-core-1.3.jar!/" />
16           <root url="jar://$APPLICATION_HOME_DIR$/lib/hamcrest-library-1.3.jar!/" />
17         </CLASSES>
18         <JAVADOC />
19         <SOURCES />
20       </library>
21     </orderEntry>
22   </component>
23 </module>

```

For more information: <http://stackoverflow.com/questions/14539072/java-langnoclassdeffoundererror-org-hamcrest-selfdescribing/22975179#22975179>

Task 3: Amending the test suite

You are given a test suite with 71% mutation coverage and 94% line coverage. Your task is to write more tests to kill most of the remaining mutants and look for bugs (here we mean the bugs that were in the code before the mutation testing started).

PIT tool can be launched:

Run As -> PIT Mutation Test (**Eclipse**) or

Run PIT (or whatever you named in the set up for **IntelliJ**).

Result should be displayed in the bottom side of IDE.

For **Eclipse** in the **PIT Mutations** tab, coverage report is in **PIT Summary** tab

-> MinBinayHeap.java

For **IntelliJ**, click Open report in browser for mutation coverage report.

Light green shows line coverage; dark green shows mutation coverage. Light pink shows lack of line coverage; dark pink shows lack of mutation coverage.

The goal is to kill mutants and fix the bugs. Once you find a bug, fix it and



continue to look for more bugs.

For this lab you should submit a ZIP folder containing the following:

1. PDF report including:

- A list of found bugs (with brief statement what is wrong)
- A list of the added test cases.
- PIT mutation coverage and line coverage statistics
- Bonus: A list of equivalent mutants with explanations (if found)

2. Fixed code and amended test suite

11.6 Homework

Create any program having recursion and loops in it. Test the program, then add the mutants and retest the program. Make a comparison report.



Lab # 12

PIT (Mutation Testing)



Lab 12: Mutation Testing Using PIT

12.1 Objective:

Objective of this lab is to learn about mutation using PIT. Reports are developed using IntelliJ and PIT in maven.

12.2 Scope:

- After this lab students will be able to learn mutation testing using PIT tool and IntelliJ compiler.
- Students will also be able to learn about maven-based projects and create mutation-based programs.

12.3 Useful Concept:

Mutation testing is a way of testing where bugs (mutations) are seeded into your program and then the tests are run. If the tests fail, then the mutations are killed. If not, then the mutations are alive. The quality of your tests can be gauged from the percentage of mutations killed.

The purpose of this lab is to explain why mutation testing is important and should be used as an addition to other software testing methods and introduce one of the mutation testing tools PIT.

Analysis Tool:

The tool used in this lab is PIT mutation testing system which has a plugin for Eclipse. It applies a configurable set of mutation operators (or mutators) to the byte code generated by compiling your code.

12.4 Examples:

Example 1 Code:



The screenshot shows an IDE with a project named 'PITestingProject'. The project structure on the left includes 'src/main/java/mutate' containing 'Example2', 'PITExample', 'Thermist', and 'ToBeTested'. The 'test' directory contains 'Example2Test', 'PITExampleTest', 'TestMutate', and 'TestThermist'. The main code editor displays the following Java code:

```

1 package mutate;
2
3 public class Example2 {
4     public String getSomething(int someParameter) {
5         if (someParameter > 0) {
6             return "foo";
7         } else {
8             return "bar";
9         }
10    }
11 }
12

```

Example 1 Test:

The screenshot shows the same IDE with the project structure. The 'test' directory is expanded, showing 'Example2Test'. The code editor displays the following Java code for 'Example2Test.java':

```

1 package mutate.test;
2
3 import mutate.Example2;
4 import org.junit.Test;
5
6 import static junit.framework.TestCase.assertEquals;
7
8 public class Example2Test {
9     Example2 example2 = new Example2();
10
11     @Test
12
13     public void testOne() { assertEquals( expected: "foo", example2.getSomething( someParameter: 1)); }
14
15     @Test
16     public void testMinusOne() { assertEquals( expected: "bar", example2.getSomething( someParameter: -1)); }
17
18 }
19

```



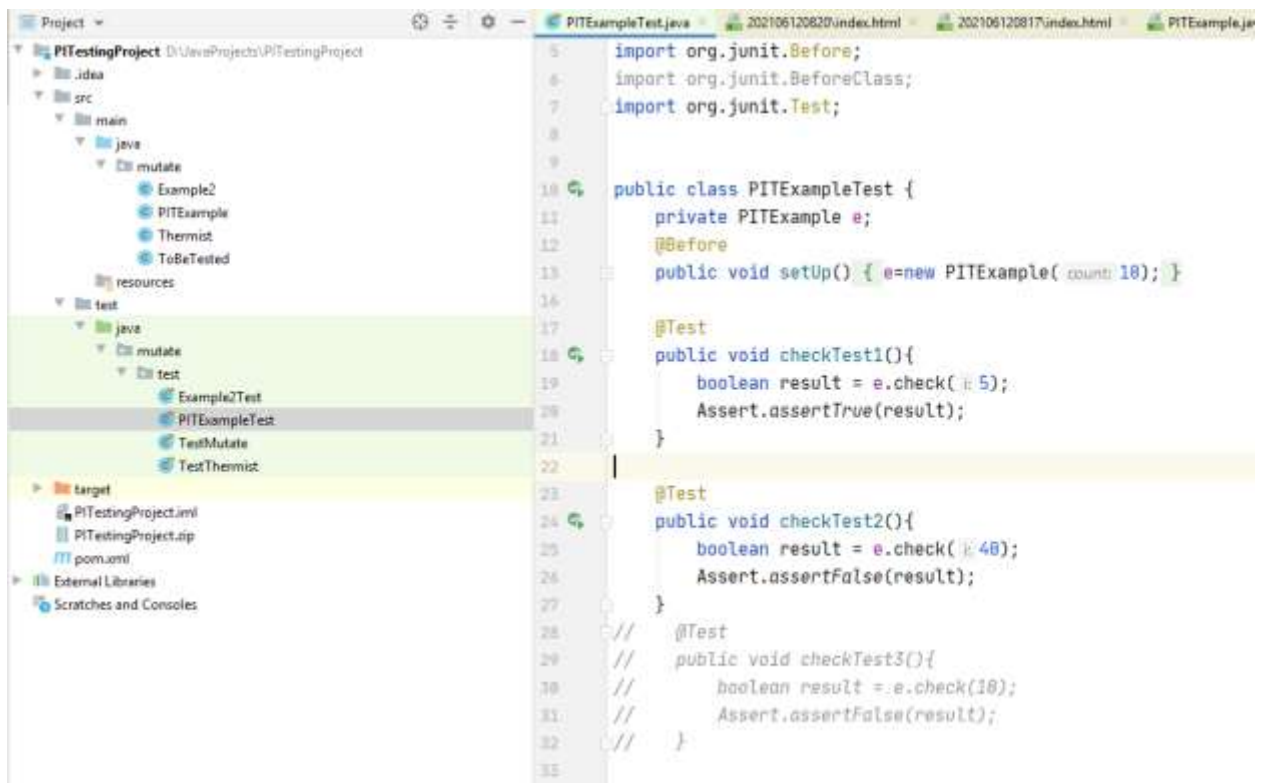
Example 2 Code:



The screenshot shows an IDE with a project named 'PITestingProject'. The project structure on the left includes 'src/main/java/mutate' containing 'Example2', 'PITExample', 'Thermist', and 'ToBeTested'. The 'test' directory contains 'Example2Test', 'PITExampleTest', 'TestMutate', and 'TestThermist'. The main editor displays the source code for 'PITExample.java' in the 'mutate' package. The code defines a 'PITExample' class with a private 'count' attribute, a constructor, and a 'check' method.

```
1 package mutate;
2
3 public class PITExample {
4     private int count;
5
6     public PITExample(int count) { this.count=count; }
7     public boolean check(int i){
8
9         return i<=count;// i=10, count=10; false
10    }
11 }
12
13
14
```

Example 2 Test:



The screenshot shows the same IDE with the 'PITExampleTest.java' file open. The project structure on the left is the same as in the previous screenshot. The main editor displays the test code for 'PITExampleTest'. The code imports 'org.junit.Before', 'org.junit.BeforeClass', and 'org.junit.Test'. It defines a 'PITExampleTest' class with a private 'e' attribute, a 'setUp' method, and three test methods: 'checkTest1', 'checkTest2', and 'checkTest3' (commented out).

```
1 import org.junit.Before;
2 import org.junit.BeforeClass;
3 import org.junit.Test;
4
5
6 public class PITExampleTest {
7     private PITExample e;
8     @Before
9     public void setUp() { e=new PITExample( count: 10); }
10
11     @Test
12     public void checkTest1(){
13         boolean result = e.check(10);
14         Assert.assertTrue(result);
15     }
16
17     @Test
18     public void checkTest2(){
19         boolean result = e.check(10);
20         Assert.assertFalse(result);
21     }
22
23     // @Test
24     // public void checkTest3(){
25     //     boolean result = e.check(10);
26     //     Assert.assertFalse(result);
27     // }
28
29
30
31
32
33
```



Required POM File:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>PITestingProject</artifactId>
  <version>1.0-SNAPSHOT</version>

  <build>
    <plugins>

      <plugin>
        <groupId>org.pitest</groupId>
        <artifactId>pitest-maven</artifactId>
        <version>1.4.2</version>
        <configuration>

          <targetClasses>
            <param>mutate.*</param>
          </targetClasses>
          <targetTests>
            <param>mutate.test.*</param>
          </targetTests>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
    </dependency>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
```



```

    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
  </dependency>
</dependencies>

<properties>
  <java.version>1.8</java.version>
</properties>
</project>

```

Mutation Automated Report for Code 1:

Example2.java

```

1 package mutate;
2
3 public class Example2 {
4     public String getSomething(int someParameter) {
5         if (someParameter > 0) {
6             return "foo";
7         } else {
8             return "bar";
9         }
10    }
11 }

```

Mutations

```

5 1. changed conditional boundary + SURVIVED
6 2. negated conditional + KILLED
7 1. mutated return of Object value for mutate/Example2::getSomething to ( if (x != null) null else throw new RuntimeException ) + KILLED
8 1. mutated return of Object value for mutate/Example2::getSomething to ( if (x != null) null else throw new RuntimeException ) + KILLED

```

Active mutators

- INCREMENTS_MUTATOR
- VOID_METHOD_CALL_MUTATOR
- RETURN_VALS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- INVERT_NEGS_MUTATOR
- CONDITIONALS_BOUNDARY_MUTATOR

Tests examined

- mutate.test.Example2Test.testOne(mutate.test.Example2Test) (1 ms)
- mutate.test.Example2Test.testMinusOne(mutate.test.Example2Test) (19 ms)

Report generated by [PIT](#) 1.4.2

Mutation Code 2:



PITExample.java

```
1 package mutate;
2
3 public class PITExample {
4     private int count;
5
6     public PITExample(int count){
7         this.count=count;
8     }
9     public boolean check(int i){
10
11         return i<count;// i=10, count=10; false
12     }
13 }
```

Mutations

1. changed conditional boundary → KILLED
11 2. negated conditional → KILLED
3. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED

Active mutators

- INCREMENTS_MUTATOR
- VOID_METHOD_CALL_MUTATOR
- RETURN_VALS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- INVERT_NEGS_MUTATOR
- CONDITIONALS_BOUNDARY_MUTATOR

Tests examined

- mutate.test.PITExampleTest.checkTest3(mutate.test.PITExampleTest) (0 ms)
- mutate.test.PITExampleTest.checkTest2(mutate.test.PITExampleTest) (1 ms)
- mutate.test.PITExampleTest.checkTest1(mutate.test.PITExampleTest) (1 ms)

12.5 Exercises for lab:

Code is provided and seed a mutant and generate PIT Report:



Code 1:

```
package mutate;

public class Thermist {
    int a = 90;

    public int checkTemp(int temp) {

        if (temp > a) {
            temp++;
        }

        return temp;
    }
}
```

Mutation code:

Page 105 | 138



```

package mutate.test;

import mutate.Thermist;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;

public class TestThermist {
    Thermist t = new Thermist();
    @Test
    public void checkTempTest(){
        Assert.assertEquals( expected: 101, t.checkTemp(100));
    }
}

```

Code 2:



```

package mutate;

public class ToBeTested {
|
    public int display() {
        int a = 9;
        int b = 8;
        int c = 7;
        int d = a + b + c;
        System.out.printf("d="+d);
        return d; //24
    }
}

```

Mutation Code:

Page 107 | 138



```

package mutate.test;

import mutate.ToBeTested;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;

public class TestMutate {

    ToBeTested toBeTested = new ToBeTested();

    @Test //annotation
    public void displayTest() {
        int a = 9;
        int b = 8;
        int c = 7;
        int d = a + b + c; //24
        int res = toBeTested.display();
        Assert.assertEquals(d, res);
    }
}

```

12.6 Homework:

Develop a code for prime number, Temperature conversion scales and average calculator. Seed mutants and derive mutation PIT report graphically.



Lab # 13

GUI Testing

Page 109 | 138



Lab 13: GUI Testing

13.1 Objective:

The aim of this lab is to provide practical experience about GUI Testing

Tool used in the lab is Sikuli

13.2 Scope:

1. After performing testing of GUI in this lab, students will be able to learn:
2. Sikuli
3. GUI Testing

13.3 Useful Concept:

The purpose of this lab is to learn how Visual GUI Testing helps testers automate the testing of GUI interfaces. Visual GUI Testing (VGT) tools are usually used for acceptance testing but can also be used for regression testing. The tool that we will be working with in this Lab is “Sikuli”.

Most automated testing techniques approach testing from low-level abstraction. GUI intensive systems however require high-level tests. To battle the problem of automating high-level tests techniques such as “Record and Play” and “VGT” were developed.

Record and Play

A tool that uses this technique, records the coordinates of GUI-components, with which the tester manually interacts. Then, the recording of the test can be played again to simulate the user’s interaction with the system. This technique provides a much better solution than manual testing, because the test needs to be done manually only once. However, this technique has a major flaw. “Record and Play” tools are usually sensitive to GUI layout, which means that it is nearly impossible to run the tests consistently on different screens, resolutions, machines etc.

Visual GUI testing is a technique that uses image recognition to interact with what is shown on the screen. In the process of VGT, the tester creates a script with valid instructions. GUI interaction, that was previously done by specifying the exact coordinates of the pixels of the element, are now done by taking a screenshot of the GUI element in question. VGT tools interact with the computer’s screen to search for matches of these screenshots. This makes automating high-level tests much more reliable and useful.

Sikuli

Sikuli is an open source VGT tool developed by researchers at MIT, USA. Sikuli identifies and interacts with elements of GUI through the principle of image matching. Sikuli consists of Jython based API (mix of Python and Java), which gives the tester an opportunity to write readable test scripts. Integrated development environment Sikuli IDE is a fully functioning IDE for creating visual GUI tests.



13.4 Examples:

Installation

To run Sikuli IDE you must have a valid Java installation of at least Java 11 (Sikuli works with Java 8 but the applications need Java 11). To check your Java version enter the following on the command line:

```
java -version
```

Download here:

<https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html>

Before running Sikuli IDE make sure to download Jython from this link:

<https://repo1.maven.org/maven2/org/python/jython-standalone/2.7.1/jython-standalone-2.7.1.jar>

Download Sikuli IDE version 2.0.0 (tested for this lab) from here

<https://launchpad.net/sikuli/+milestone/2.0.0>

or download the latest stable build from here:

<https://raiman.github.io/SikuliX1/downloads.html>

Then place both files in the same folder of your choice. Doubleclick on sikulix jar to run it or run from

command line

```
java -jar path-to/sikulix.jar
```

The Jython file will be automatically removed after running Sikuli for the first time. If everything is

set up correctly in the bottom right corner of Sikuli IDE you should see that it uses Jython.



(jython) | R: 1 | C: 1

If you don't see (jython) in the IDE, you should delete the IDE, download it again and redo the instructions.

Maintain pom.xml file as:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```



```

<groupId>org.example</groupId>
<artifactId>SikuliGuiTesting</artifactId>
<version>1.0-SNAPSHOT</version>

<dependencies>
  <dependency>
    <groupId>org.sikuli</groupId>
    <artifactId>sikuli-api</artifactId>
    <version>1.2.0</version>
  </dependency>
  <dependency>
    <groupId>com.sikulix</groupId>
    <artifactId>sikulixapi</artifactId>
    <version>1.1.2</version>
    <exclusions>
      <exclusion>
        <groupId>com.github.vidstige</groupId>
        <artifactId>jadb</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <!-- <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-chrome-driver</artifactId>
    <version>2.50.0</version>
  </dependency>-->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
  </dependency>

  <!-- <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.0.0-alpha-6</version>
  </dependency>-->
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>

```




```

        <artifactId>selenium-java</artifactId>
        <version>3.141.59</version>
    </dependency>
</dependencies>

    <properties>
        <java.version>1.8</java.version>
    </properties>
</project>

```

To open the folder with similar visual property, you can use following code to open music folder in window explorer.

```
import org.sikuli.script.FindFailed;
```

```
import org.sikuli.script.ImagePath;
```

```
import org.sikuli.script.Pattern;
```

```
import org.sikuli.script.Screen;
```

```
import java.awt.event.KeyEvent;
```

```
import java.util.Random;
```

```
public class Example {
```

```
    public static final Pattern MUSIC = new Pattern(Example.class.getResource("music.PNG"));
```

```
    public static final Pattern NEW_FOLDER = new
    Pattern(Example.class.getResource("new_folder.PNG"));
```



```

public static void main(String[] args) throws FindFailed {

    Screen s = new Screen();

    s.keyDown(KeyEvent.VK_WINDOWS);

    s.keyDown(KeyEvent.VK_E);

    s.keyUp(KeyEvent.VK_WINDOWS);

    s.keyUp(KeyEvent.VK_E);


    s.wait(MUSIC.similar((float) 0.90), 2).click();

    s.wait(NEW_FOLDER.similar((float) 0.90), 2).click();


    Random rand = new Random();

    s.type("Sikuli Automation " + rand.nextInt(10));

    s.keyDown(KeyEvent.VK_ENTER);

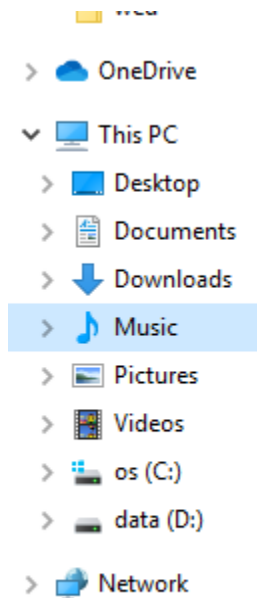
    s.keyUp(KeyEvent.VK_ENTER);

}
}

```



Output can be like:



13.5 Exercises for lab:

To make Login to facebook:

```
import org.openqa.selenium.By;
```

```
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.chrome.ChromeDriver;
```

```
import org.openqa.selenium.firefox.FirefoxDriver;
```

```
public class Facebook {
```

```
    public static void main(String[] args) {
```



```

try {

    System.setProperty("webdriver.gecko.driver", "geckodriver.exe");

    WebDriver driver = new FirefoxDriver();

    Thread.sleep(2000);

    driver.manage().window().maximize();

    driver.get("https://www.facebook.com");//signup

    Thread.sleep(3000);

    driver.findElement(By.id("email")).sendKeys("abc@gmail.com");

    driver.findElement(By.id("pass")).sendKeys("mypassword");

    driver.findElement(By.name("login")).click();

} catch (Exception exception) {

    exception.printStackTrace();

}

}

```

13.6 Homework:

Make a new folder, copy pictures from any folder and paste in the newly created folder. Test the environment in GUI testing.



Lab # 14

Sikuli



Lab 14: Sikuli

14.1 Objective:

To familiarize the students with Sikuli and GUI testing.

14.2 Scope:

At the end of this lab students will be able to learn more about:

1. Visual Testing
2. Automated Testing
3. GUI Testing

14.3 Useful Concept:

“Automate anything you see” using the Sikuli Graphical User Interface (GUI) automation tool.

Sikuli is a tool to automate Graphical User Interfaces (GUI) using the “Visual Image Match” method. In Sikuli, all the web elements should be taken as an image and stored inside the project. Sikuli will trigger GUI interactions based on the image visual match, the image which we have passed as the parameter along with all methods.

Sikuli can be very much useful to automate flash objects (which do not have ID or name). It can be useful in the situation, where we have a stable GUI (i.e. GUI components not changing).

Even Window based applications can also be automated using Sikuli. Sikuli provides very friendly Sikuli-script.jar, which can be easily used together with Selenium WebDriver. We can even automate Adobe Video/Audio player, Flash Games on the website using Sikuli. With simple API, it makes coding easier.

14.4 Examples:

Step #1) Open a YouTube video link and Capture play and pause element images using the screen capture tool.

Pause button (**Note:** filename is pause.png)

Play button (**Note:** filename is play.png)

Copy these images inside the project.

Step #2) Create a package inside the Sikuli java project created and within that create a class named “Youtube”.

Step #3) Type the following code inside that class.

```
package com.test;
```



```

import org.sikuli.script.FindFailed;
import org.sikuli.script.Screen;

public class Youtube {

    public static void main(String[] args) throws FindFailed, InterruptedException {
        // TODO Auto-generated method stub

        Screen s=new Screen();
        s.find("pause.png"); //identify pause button
        s.click("pause.png"); //click pause button
        System.out.println("pause button clicked");

        s.find("play.png"); //identify play button
        s.click("play.png"); //click play button
    }

}

```

Step #4) Right-click on the class select Run As -> Java Application.
Open Notepad And Type Some Text

Step #1) Capture the notepad icon on the desktop on the screen.

notepad_icon.png

notepad.png

Step #2) Copy these images inside your project.

Step #3) Create a class named “NotepadExample” inside your project and type the following code.

package com.test;

```

import org.sikuli.script.FindFailed;
import org.sikuli.script.Screen;

public class NotepadExample {

    public static void main(String[] args) throws FindFailed {

```



```
// TODO Auto-generated method stub
```

```
Screen s=new Screen();
s.click("notepad_icon.png");
s.find("notepad.png");
s.type("notepad.png","This is Nice Sikuli Tutorial!!!!");
}

}
```

Step #4) Open the screen to be tested before executing the code.

Execute this file by Right click Run As -> Java Application.

14.5 Exercises for lab:

Google Search Example:

```
import org.openqa.selenium.WebDriver;
//import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.sikuli.script.*;

public class GoogleSearchExample {
    public static final Pattern GOOGLE = new
Pattern("D:\\SikuliGuiTesting\\SikuliGuiTesting\\src\\main\\resources\\google.JPG");

    public static void main(String[] args) {
        try {
            //      System.setProperty("webdriver.chrome.driver", "chromedriver.exe");
            System.setProperty("webdriver.gecko.driver", "geckodriver.exe");
            WebDriver driver = new FirefoxDriver();
            //      WebDriver driver = new ChromeDriver();

            driver.get("http://www.google.com");
            driver.manage().window().maximize();
            Thread.sleep(2000);

            Screen screen = new Screen();

            //      System.out.println(ImagePath.getBundlePath());
        }
    }
}
```




```

        screen.wait(GOOGLE.similar((float) 0.90), 10);

//    screen.wait(imageSearch, 20);
        screen.click(GOOGLE);
        screen.type("Introduction to Sikuli");
        Thread.sleep(2000);
        screen.type(Key.ENTER);
        Thread.sleep(2000);

        // Close the browser
//    driver.quit();
    } catch (Exception exception) {
        System.out.println("File not Found!");
        exception.printStackTrace();
    }
}
}

```

Drag and Drop of an image/folder:

```
package com.test;
```

```
import org.sikuli.script.FindFailed;
import org.sikuli.script.Screen;
```

```
public class DragAndDrop {
```

```

    public static void main(String[] args) throws FindFailed, InterruptedException {
        // TODO Auto-generated method stub
        Screen s=new Screen();
        s.find("source.png");
        System.out.println("Source image found");
        s.find("target.png");
        System.out.println("target image found");
        s.dragDrop("source.png", "target.png");
    }
}

```



14.6 Homework:

Go to the following link:

["http://www.thecolor.com/Coloring/a-puppy-with-a-kitten.aspx"](http://www.thecolor.com/Coloring/a-puppy-with-a-kitten.aspx)

And color the image through GUI Sikuli based visual testing.



Lab # 15

Appium



Lab 15: Appium

15.1 Objective:

Objective of this lab is to learn testing at mobile platforms. For the said purpose, Appium tool will be used.

15.2 Scope:

Familiarization of the following learning aspects are covered in this lab:

Appium

Mobile Testing

.apk file testing

GUI testing of mobile applications

15.3 Useful Concept:

Appium is a cross-platform testing framework that is flexible, enabling testers to write test scripts against multiple platforms such as iOS, Windows, and Android using the same API. That means QAs can use the same code for iOS as for Android, time and effort. Similar to [Selenium](#), Appium allows QAs to write test scripts in different programming languages which include Java, JavaScript, PHP, Ruby, Python, and C#.



15.3.1 Installation:

Step 1) Go to <https://github.com/appium/appium-desktop/releases/tag/v1.18.0-1>

Step 2) For Windows, select the Appium-windows-1.18.0-1.exe file and download. The file is around 240MB will take time to download based on your internet speed.

Step 3) Download calculator apk file from following link
<https://apkpure.com/calculator/com.google.android.calculator>

Step 4) Add ANDROID_HOME variable in environment variables and set value of platform-tools path e.g: C:\Users\~\AppData\Local\Android\Sdk\platform-tools

Step 5) used cd to change current directory and go to the folder containing calculator.apk using command "adb install calculator.apk"

Step 6) Install Appium and then Start application.

After installation, you can open inspector session in appium.

You can use xpath's to get buttons and other regions of calculator.apk gui.



id

com.google.android.calculator:id/digit_5

Get Timing

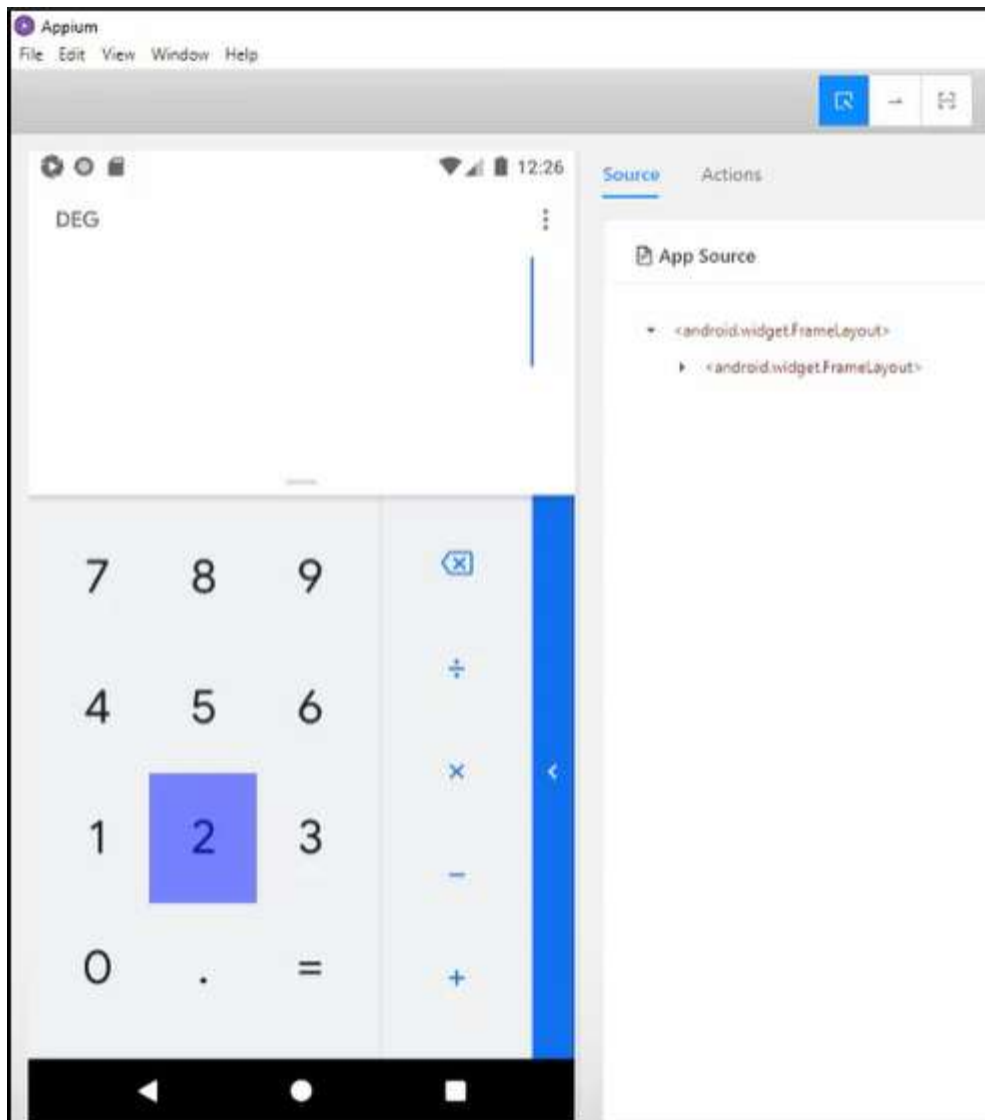
xpath

/hierarchy/android.widget.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.view.ViewGroup/android.widget.LinearLayout/androidx.slidingpanelayout.widget.SlidingPanelLayout/android.widget.LinearLayout/android.view.ViewGroup[1]/android.widget.Button[5]

Get Timing

Attribute	Value
elementId	2020086b-5922-4e67-a59c-aa3055b509e8
index	4
package	com.google.android.calculator
class	android.widget.Button
text	5





15.4 Examples:

```
package src_Appium;
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
//import org.openqa.selenium.remote.CapabilityType;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.annotations.*;
```



```

public class Calculator {
    WebDriver driver;

    @BeforeClass
    public void setUp() throws MalformedURLException{
        //Set up desired capabilities and pass the Android app-activity and app-package to Appium
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability("BROWSER_NAME", "Android");
        capabilities.setCapability("VERSION", "4.4.2");
        capabilities.setCapability("deviceName", "Emulator");
        capabilities.setCapability("platformName", "Android");

        capabilities.setCapability("appPackage", "com.android.calculator2");
        // This package name of your app (you can get it from apk info app)
        capabilities.setCapability("appActivity", "com.android.calculator2.Calculator"); // This is Launcher activity
        // of your app (you can get it from apk info app)
        //Create RemoteWebDriver instance and connect to the Appium server
        //It will launch the Calculator App in Android Device using the configurations specified in Desired Capabilities
        driver = new RemoteWebDriver(new URL("https://127.0.0.1:4723/wd/hub"), capabilities);
    }

    @Test
    public void testCal() throws Exception {
        //locate the Text on the calculator by using By.name()
        WebElement two=driver.findElement(By.name("2"));
        two.click();
        WebElement plus=driver.findElement(By.name("+"));
        plus.click();
        WebElement four=driver.findElement(By.name("4"));
        four.click();
        WebElement equalTo=driver.findElement(By.name("="));
        equalTo.click();
        //locate the edit box of the calculator by using By.tagName()
        WebElement results=driver.findElement(By.tagName("EditText"));
        //Check the calculated value on the edit box
        assert results.getText().equals("6"):"Actual value is : "+results.getText()+" did not match with expected value: 6";
    }

    @AfterClass
    public void teardown(){
        //close the app
        driver.quit();
    }
}

```




```
}  
}
```

Appium Server and Android Emulator from 'AVD Manager' and Click Run >> TestNG. Above program will run the 'Calculator.app' on selected emulator and Result displayed under Eclipse console using a TestNG framework.

15.5 Exercises for lab:

Pom.xml can be like;

```
<dependencies>  
<dependency>  
<groupId>io.appium</groupId>  
<artifactId>java-client</artifactId>  
<version>5.0.4</version>  
</dependency>  
<dependency>  
<groupId>org.seleniumhq.selenium</groupId>  
<artifactId>selenium-java</artifactId>  
<version>3.9.1</version>  
</dependency>  
<dependency>  
<groupId>org.apache.commons</groupId>  
<artifactId>commons-lang3</artifactId>  
<version>3.7</version>  
</dependency>  
<dependency>  
<groupId>org.testng</groupId>  
<artifactId>testng</artifactId>  
<version>6.14.3</version>  
<scope>test</scope>  
</dependency>  
</dependencies>
```

Question:

Test Name: testBasicNoTitle()

* Given: Application is installed and launched



- * And: User is on the home page
- * When: User selects button "BASIC (NO TITLE)"
- * Then: User should see popup with text "This app wants to access your location", "DISAGREE" and "AGREE"
- * When: User selects "AGREE"
- * Then: Popup should be dismissed

```

/*
 */

@Test (enabled=true) public void testBasicNoTitle() throws InterruptedException {

    // Find the button BASIC (NO TITLE) and click it

    driver.findElementById("com.afollestad.materialdialogssample:id/basicNoTitle").click();


    // Assert the presence of the popup title, AGREE and DISAGREE buttons

    Boolean isTitlePresent =
!driver.findElementsById("com.afollestad.materialdialogssample:id/md_content").isEmpty();

    Boolean isDisagreePresent =
!driver.findElementsById("com.afollestad.materialdialogssample:id/md_buttonDefaultNegative").isEmpty();

    Boolean isAgreePresent =
!driver.findElementsById("com.afollestad.materialdialogssample:id/md_buttonDefaultPositive").isEmpty();

    Assert.assertTrue(isTitlePresent && isDisagreePresent && isAgreePresent); // Assert the contents of
the popup title, AGREE and DISAGREE button

```



```

String popupTitle =
driver.findElementById("com.afollestad.materialdialogssample:id/md_content").getText();

Assert.assertEquals(popupTitle, "This app wants to access your location.");

String disagreeText =
driver.findElementById("com.afollestad.materialdialogssample:id/md_buttonDefaultNegative").getText(
);

Assert.assertEquals(disagreeText, "DISAGREE");

String agreeText =
driver.findElementById("com.afollestad.materialdialogssample:id/md_buttonDefaultPositive").getText();

Assert.assertEquals(agreeText, "AGREE");// Click on the AGREE button

driver.findElementById("com.afollestad.materialdialogssample:id/md_buttonDefaultPositive").click();//
Assert that the popup is no longer visible

Boolean isTitleStillPresent =
!driver.findElementsById("com.afollestad.materialdialogssample:id/md_content").isEmpty();

Assert.assertFalse(isTitleStillPresent);

}

```

15.6 Homework:

Take any .apk file and test its any functionality and make a report.



Lab # 16

Document Inspection and Fault Estimation



Lab 16: Document Inspection and Fault Estimation

16.1 Objective:

To familiarize students with document inspection and removal of faults.

16.2 Scope:

At the end of lab, students will be able to learn about:

Inspecting Document and cross-checking outputs against errors.

Static inspection

16.3 Useful Concept:

Deriving test cases from document and making test cases against the requirements is a difficult task. The steps needed can be:

Review the **specification** document against the **requirements** (6 user stories)

- Try to find **issues**
- Duration: 40 to 60 min
- Define **issue types**, and **severity levels**
- Report issues

Issues types:

- Inconsistencies within the specification
- Unnecessary functionality
- Missing functionality
- Incorrect functionality
- Spelling

In order to perform such testing, one can make table as follows:



Id	Description	Location	Type	Severity
A-01	Missing button A	Screen 1	Omission	H
A-02	Spelling mistake	Sentence 1	Spelling	L
...				

Document inspection is an important static technique to detect faults early in the software life-cycle. The purpose of inspections is to manually scrutinize a software artifact, for example, requirements, design or code. In addition to inspections, estimation techniques can be applied in order to estimate the fault content.

16.4 Exercises for lab:

1. Read and understand the requirements and the related specification document (see below).
2. Review the specification document against the requirements (6 user stories). Assess the quality of the specification document. Identify issues in the specification document (e.g., inconsistencies within the specification, missing or incorrect functionality, unnecessary functionality ('gold plating'), unclear/ambiguous statements, spelling errors, and so on). Create a well-organised list of all the issues you spot. Each issue should have an ID, a brief description, a remark that helps localise the issue, the type of issue, the severity of the issue. Clearly define the categories 'type' and 'severity' and explain the underlying rationale for the definition of types and severity levels.
Important: Make sure to document your work – you will have to include it in the lab report!
3. To get maximum marks for Part A, you need to find at least 8 issues and you have to provide correctly all the information requested in point 2 above.
4. Before you proceed to Part B, show your work to the lab supervisor. If you don't do this, you will lose 1 mark for either not attending the lab or for not working on Part A during the lab time.
Part B: Work in Pairs.
5. Share your individual issue lists and create a consolidated list. This might require that you have to redefine your type categories and severity classes. Explain the reasoning that made you agree on the new definitions. The consolidated list should be well-organized and each issue should have an ID. Make sure that traceability to the individual lists is maintained. In particular, the



consolidated list should clearly indicate whether an issue was found by one student only or by both students.

6. In case you have a disagreement on whether an issue detected by one student is the same as the issue of the other student, or whether an issue found by one student (but not the other) is actually an issue, still take a decision on whether to count the issues as separate or identical and make a comment explaining the different viewpoints (and why it was difficult to come to an agreement).
7. Based on the consolidated issue list, i.e., the number of issues detected and the information given on who detected an issue, make an estimate of the number of remaining (i.e., undetected) issues in the specification document. Use any of the capture-recapture model formulas presented during the lecture. Show your calculations.

Requirements

The following list of user stories (US) have been received from a customer representative (i.e., marketing):

- US1: As a customer, I would like to be able to search for flights, hotels, rental cars, cruises, and packages that combine flights with hotels and car rentals
- US2: As a customer, I would like to search for one-way, return, and multiple-leg flights
- US3: As a customer, I would like to choose the classes/categories of my flights, hotels, and cars.
- US4: As a customer, I would like to search not only for myself but for my whole family
- US5: As a customer, I would like to search for nonstop and refundable flights
- US6: As a customer, I would like to restrict my searches to specific airlines, hotel chains, and car rental companies

Note: The following specification should only be checked against this set of requirements (i.e., the 6 user stories listed above).

Specification

The actual specification document to be reviewed starts on the next page. Before you start with your review, read the following notes:

- The specification presented on the next two pages is supposed to correspond exactly with the explicit (and implicit) requirements listed in section 'Requirements' above.
- The specification consists of both a specification text and two mock-up s of search screens that the web application to be developed will provide. The functionality contained in the two mockup search screens (and the related textual description) is supposed to address the needs of users (i.e., future customers) that want to search for flight offers.



- Screen 1 shows what a customer sees when selecting search option 'Flights' + 'Roundtrip'. When you review the screen and the corresponding specification text, you should restrict your review to the functionality that should be provided, if a customer wants to search for roundtrip flight offers. Note also that 'Advanced options' has not been selected. You can assume that the customer will see the same information that is shown in Screen 2, if 'Advanced options' had been selected.
- Screen 2 shows what a customer sees when selecting search option 'Flights' + 'One way'. Thus, when you review the screen and the corresponding specification text, you should restrict your review to the functionality that should be provided, if a customer wants to search for one-way flight offers. Note also that 'Advanced options' has been selected. You can assume that the customer will see the same information that is shown in Screen 1, if 'Advanced options' had not been selected.
- Specification Document (relating to the 6 user stories listed in the Requirements section):
Screen 1: Mock-up screen in the state after 'Flights' button and 'Roundtrip' button have been selected, and before any additional data has been entered or functions have been activated by the customer (i.e., user of the web-application to be developed)

TRAVEL MASTER – Flights, Hotels, Cars and Cruises at your finger tips

Flights

Hotels

Flight + Hotel

Cars

Cruises

Things to do

Roundtrip

One way

Multiple destinations

Adults (18+)

1

Children (3-17)

0

Flying from

City or airport

Flying to

City or airport

Departing

mm/dd/yyyy

Returning

mm/dd/yyyy

Add a hotel

Add a car

Advanced options

Search

Screen 2: Mock-up screen in the state after 'Flights' button and 'One way' button as well as 'Advanced options' have been selected, and before any additional data has been entered or functions have been activated by the customer (i.e., user of the web-application to be developed).

1. The search screen shows a welcome line with text in the top line
 2. The user can select 6 main functions by pressing any of the buttons 'Flights', 'Hotels', 'Flights + Hotels', 'Cars', 'Cruises', and 'Things to do'
 3. When 'Flights' has been selected, the user can select 3 search modes by pressing any of the buttons 'Roundtrip', 'One way', and 'Multiple destinations'
- <Note: the following assumes that the 'Flight' button has been selected>
4. When 'Roundtrip' has been selected (Screen 1), the user can do the following:
 - a) Specify flight start location (city or airport) and flight destination (city or airport); this is supported by a pull-down menu (not shown in detail in the screen mock-ups)
 - b) Departure date and return date (format: dd/mm/yyyy); this is supported by a calendar menu from where the user can pick the dates (not shown in detail in the screen mock-ups) – alternatively the dates can be entered by the user directly in the specified format.



5. When 'One way' has been selected (Screen 2), the user can do the following:
 - a) Specify flight start location (city or airport) and flight destination (city or airport); this is supported by a pull-down menu (not shown in detail in the screen mock-ups)
 - b) Departure date (format: dd/mm/yyyy); this is supported by a calendar menu from where the user can pick the dates (not shown in detail in the screen mock-ups) – alternatively the date can be entered by the user directly in the specified format
6. The user can select the number of adults and the number of children (supported by pull-down menus); the default selection for adults is '1', that for children is '0'
7. The user can check boxes 'Add a hotel' and 'Add a car', if offers for a hotel and/or a rental car should be added to the flight offers
8. If the user selects 'Advanced options', then he can restrict his flight search to 'Non-stop' and/or 'Refundable' flights. In addition, he can select the flight class (first – business – premium economy – economy)
9. Pressing the 'Search' button will start the retrieval of flight offers 10. Pressing the 'Check' button will start the retrieval of flight offers and check the availability of free seats.

Note: The specification of *output screens* is NOT shown. Therefore, you cannot know whether they are specified correctly, and thus you should not speculate about any issues related to them. You shall focus exclusively on reviewing whether the specification document actually provided contains any issues related to searches for roundtrip and one-way flights in the to-be-developed web application. Also, you should check whether the specification document contains functionality that has not been mentioned (explicitly or implicitly) in the list of requirements and thus would be 'gold plating'.

16.5 Homework:

Take any website of shopping magazine and do an artwork of making requirements (assumptions) and create document of inspection.

