Name: Thien Tran
ID#: 1222245570

CSE 565: Software Verification and Validation
Project 2 Report
Sep-05-2021

Part 1

1. Description of the tool used and the types of coverage it provides

Tool name: Jacoco

Jacoco is a tool. Its main competitive edge is simple and light, compared to other tools on the market. It also provides code coverage metrics for statement, decision, class and method. Furthermore, it can also be used for code delta testing (if implemented in IntelliJ). Last but not least, it has 2 additional features: output reports and pinpointing uncovered lines.

2. Set of test cases:

#	Item	Input
1	candy	21
2	candy	20
3	candy	19
4	coke	24
5	coffee	44

Though the requirements ask for 100% decision coverage, in reality it only covers at most 93% of all the branches. The achievement of 15/16 decision coverage can be shown in the snapshot. This project creates the control flow (appendix 2.A, 2.B) to construct the data flows. By systemically mapping the data flows in the appendix 3 to the test cases we obtain the data flow coverage mapping table (appendix 1). Using the aforementioned table(appendix 1), we pinpointed the predicate PVI whose False result is unreachable. We will now prove that achieving the last one of the 16 decisions is impossible.

Proof by contradiction:
Assume the path (PIV: False, PV: False, PVI: False) is reachable.
Let cost(item) be the cost of an item of 3 item types.
We notice 2 facts:

1. The path requires the decisions of predicates PIV, PV, PVI to be false-false-false regardless of all of the decisions of the other predicates (appendix 1, in red).
2. All of the items cost less than or equal to 45, cost(item) <= 45 (appendix 4).

From the path construction in the appendices 2,3, we have the following:

def1(input1) -> PIV(input1) -> PV(input1) -> PVI(input1)
For these to be false, all of the following conditions must be true:

(1) input1 <= cost(item)
(2) input1 != cost(item)
(3) input1 >= 45

Combining (1), (2) and (3), we have the inequality:
45 <= input1 < cost(item) <=> 45 < cost(item)
The premise 45 < cost(item) contradicts the fact that all of the items cost less than 45 (which is that 45 >= cost(item)).
Therefore, path (PIV: False, PV: False, PVI: False) is unreachable

Note that: this was done in accordance with the requirement in the CSE 565 Project 2_Analyzing Code Coverage.pdf file. Since the clarification statement during the live event was made late and the result of the proof above did not affect the overall quality of the project, this project decided not to make changes to the project report.

3. Screenshot showing the coverage achieved for the test cases developed

Element	Class, %	Method, %	Line, %	Branch, %
VendingMachine	100% (1/1)	100% (2/2)	100% (32/32)	93% (15/16)
com				
images				
java				
javax				
jdk				
kotlin				
META-INF				
netscape				
org				
sun				
toolbarButtonGraphics				

4. Evaluation of the tool’s usefulness

With the tool, software testers can reduce the workload by automating the code coverage results of the test cases. It also supports proof of correctness of the actual results of the test cases against the expected results which were created during the planning of the data flow coverage and control flow coverage.

Part 2:

1. Description of the tool used and the types of analysis it provides

Tool name: IntelliJ IDEA code inspection

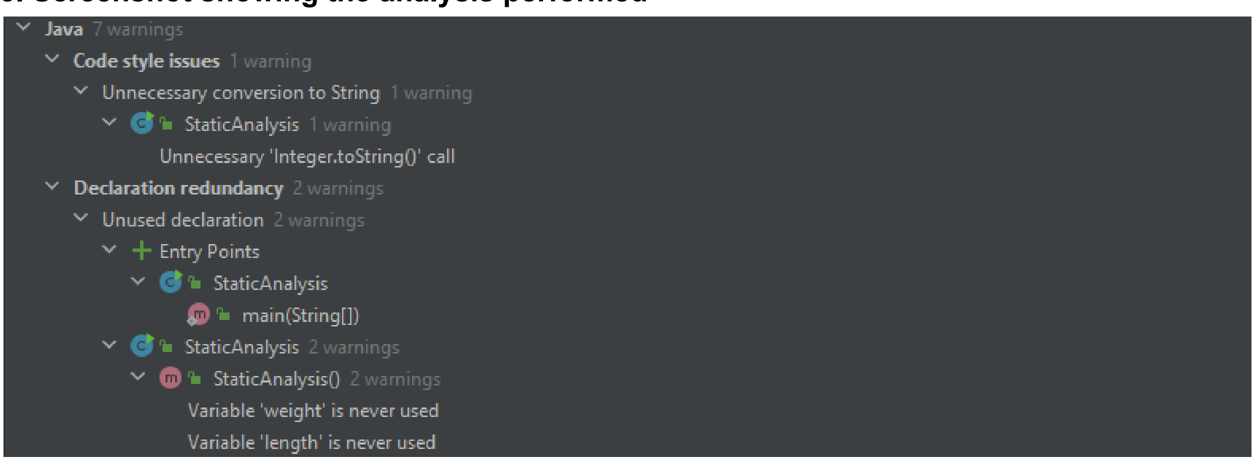
IntelliJ IDEA code inspection, a feature embedded into IntelliJ IDEA , is a static code analyzer. It provides major analyses such as: data flow anomaly, code analysis, etc.

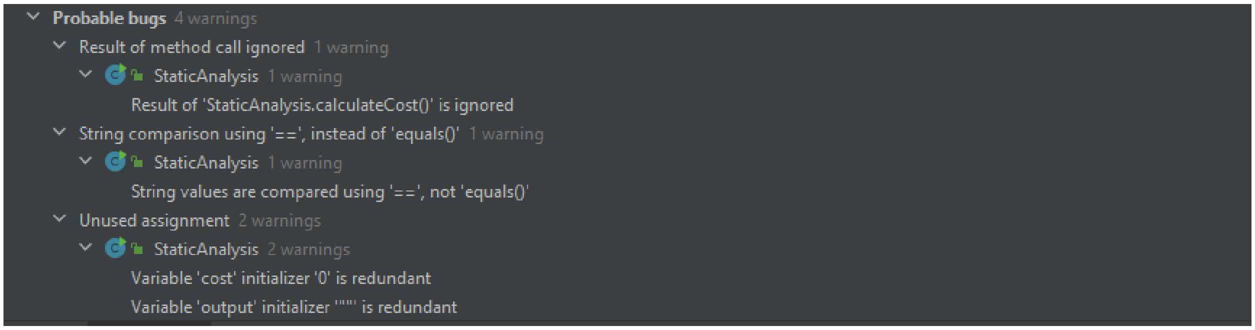
2. Description of the two data flow anomalies

Line	Code	Description
8	calculateCost(5, 10, "Electronics");	The variable output is returned without a container or being used.
15	int weight = 0;	Variable weight is declared and defined without being used
16	String length = "";	Variable length is declared and defined without being used
20	int cost = 0;	the value 0 of cost is never used
21	String output = "";	the value “” of output is never used

Note: there are other warnings but not related to the data flow anomalies, and this project has captured more than 2 anomalies required.

3. Screenshot showing the analysis performed





4. Evaluation of the tool’s usefulness

It helps software testers and programmers to prevent the errors from lurking in the program. It reduces time, cost and resources to find the bugs via test cases.

Appendix

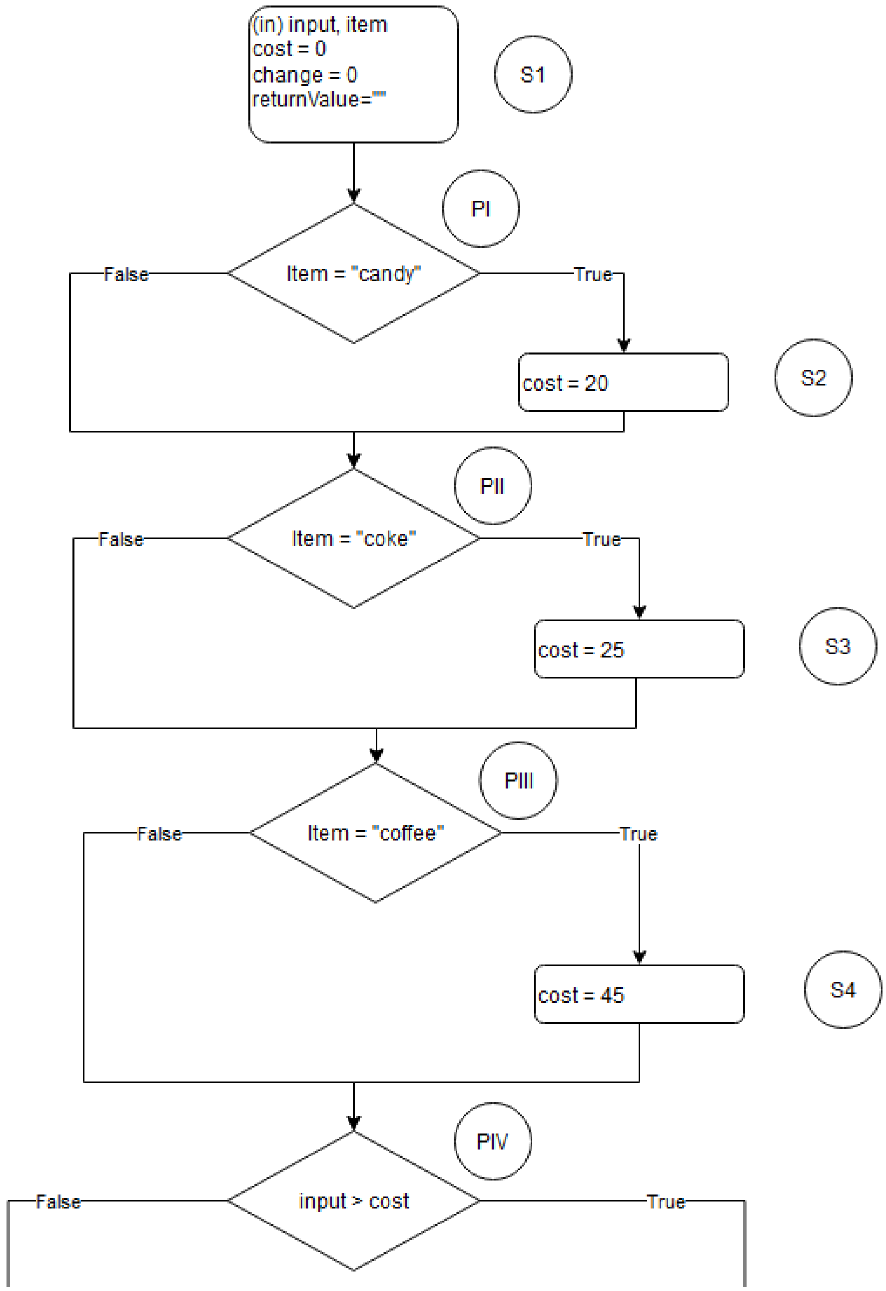
Appendix 1: Data flow path mapping table

Test case	1	2	3	4	5
def1input->PIV	T	F	F	F	F
def1input->PV	NA	T	F	F	F
def1input->C5	x				
def1input->C6			x	x	x
def1input->PVI			T	T	T
def1input->PVII			T	T	F
def1input->PVIII			T	F	F
def1(item) ->PI	T	T	T	F	F
def1(item) ->PII	F	F	F	T	F
def1(item) ->PIII	F	F		F	T
def2(cost) ->PIV	T	F		F	F
def2(cost) ->PV	NA	T	F	F	F
def2(cost) ->C5	x				
def2(cost) ->C6			x		
def3(cost) -> PIV				F	
def3(cost) -> PV				F	
def3(cost) -> C5					
def3(cost) -> C6				x	
def4(cost) -> PIV					F
def4(cost) -> PV					F
def4(cost) -> C5					
def4(cost) -> C6					x
def5(change)->C5	x				
def5(returnValue)->C11	x				

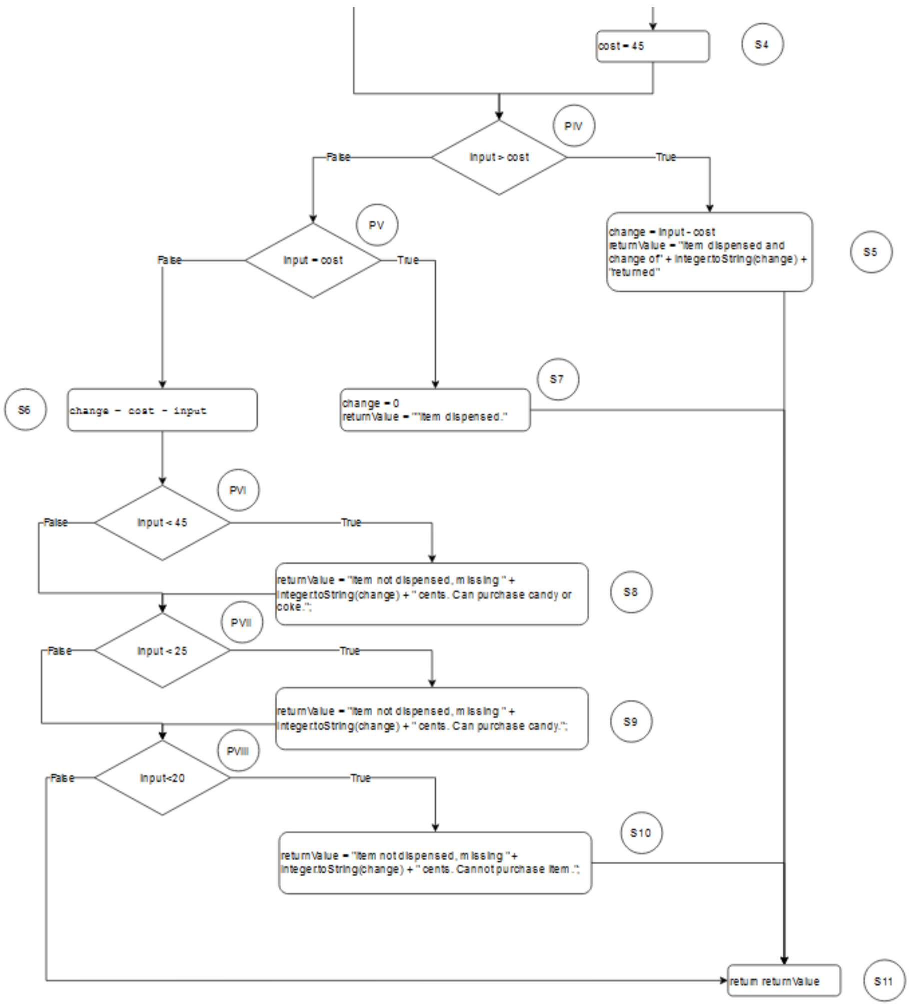
def6(change) -> C8			x	x	x
def6(change) -> C9			x	x	
def6(change) -> C10			x		
def7(returnValue) -> C11		x			
def8(returnValue) -> C11					x
def9(returnValue) -> C11				x	
def10(returnValue) -> C11			x		
def1(change)	N/A	N/A	N/A	N/A	N/A
def1(cost)	N/A	N/A	N/A	N/A	N/A
def1(returnValue)	N/A	N/A	N/A	N/A	N/A
def7(change)	N/A	N/A	N/A	N/A	N/A

Appendix 2

Appendix 2.A: Control flow diagram (Part A)



Appendix 2.B: Control Flow Diagram (Part B)



Appendix 3: Data flow paths

def1(input1) -> PIV(input1) | PV(input1) | C5(input1) | C6(input1) | PVI(input1) | PVII(input1) | PVIII(input1)
def1(item1) -> PI(item1) | PII(item1) | PIII(item1)
def1(change1)
def1(cost1)
def1(returnValue1)

def2(cost2) -> PIV(cost2) | PV(cost2) | C5(cost2) | C6(cost2)

def3(cost3) -> PIV(cost3) | PV(cost3) | C5(cost3) | C6(cost3)

def4(cost4) -> PIV(cost4) | PV(cost4) | C5(cost4) | C6(cost4)

def5(change5) -> C5(change5)
def5(returnValue5) -> C11(returnValue5)

def6(change) -> C8(change6) | C9(change6) | C10(change6)

def7(change7) ->
def7(returnValue7) -> C11(returnValue7)

def8(returnValue8) -> C11(returnValue8)

def9(returnValue9) -> C11(returnValue9)

def10(returnValue10) -> C11(returnValue10)

Appendix 4: Explanation of observation 1 and the premise gained from the symbolic execution

From the result of the symbolic execution of the program(below), we have the maximum cost of all the items at PVI is max(cost2,cost3,cost4) = max(20,25,45) = 45, which means that (cost(item) <= 45).

S1:
input1, item1
cost1 = 0
change1 = 0
returnValue1 = 0

S2
cost2 = 20

S3
cost3 = 25

S4
cost4 = 45

S5
change5 = input1 - cost(2|3|4)
returnValue5 = change5 = input - cost(2|3|4)

S6
change6 = cost1 - input1

S7
change7
returnValue7

S8
returnValue8 = change6 = cost1 - input1

S9
returnValue9 = change6 = cost1 - input1

S10
returnValue10 = change6 = cost1 - input1

S11
returnValue11 = returnValue(5|7|8|9|10)