SOFTWARE TESTING

SSE3305-1

REPORT TITLE:

Individual Assignment (Pension Calculator)

Prepared For:

Assoc. Prof. Dr Salmi Binti Baharom

Prepared By:

Julia Nurfadhilah Binti Mohamad Fauzi

Matric Number:

208256

Table of Content

List Of Tables

List Of Images

1. Test Strategy

1.1 Introduction

Software testing is a process of verifying and validating that a software application meets the requirement specification and the client's requirement specification. The purpose behind it is to briefly detect any defects as soon as possible.

1.2 Scope

The scope of testing is black box and white box functional testing for features developed in Pension Calculator from F001 to F007. It would focus on meeting the requirement of the system and making sure that the system is fit for its purpose.

1.3 Test Item And Test Traceability Matrix

The test item used is the test basis of Pension Calculator. The following table contains the functions to be tested in Pension Calculator and their traceability

Function ID	Functions	Source of Function
F001	Calculate Pension Year and Month	Test Basis
F002	Determine Pension Age	Test Basis
F003	Calculate total month-service	Test Basis
F004	Determine benefit eligibility	Test Basis
F005	Calculate Pension Benefit	Test Basis
F006	Calculate Cash Award in lieu of Leave (GCR)	Test Basis
F007	Calculate Gratuity	Test Basis

Table 1.

Based on reviewing the test basis and source code of the system, the system will calculate pension year and age, determine pension age, calculate total service month, determine benefit eligibility and calculate benefits such as pension, gratuity and GCR. Next, only when the user passes the requirement, benefits calculation results will be displayed. If the pension age is insufficient, the user will receive an error message and benefits will be displayed as zero (RM0.00). If the total month-service is insufficient, the user will receive a different error message and benefits will also be displayed as zero (RM0.00).

1.4 Test Approach

The test on Pension Calculator is a unit level functional test that focuses only on the functional part of the system. Test cases are derived such as

Black-Box Testing

I. Equivalence Partitioning
II. Boundary Value Analysis
III. Decision Table Testing

White-Box Testing

I. Control Structure Testing

Note that repeating test cases on different techniques is ignored.

1.5 Item Pass/Fail Criteria

The system must satisfy the criteria as such all test cases must be passed and fulfil requirements as stated in Test Basis.

1.6 Entry and Exit Criteria

The test basis of the Pension Calculator is needed before the testing can begin. Before the testing can end, test execution must be completed.

1.7 Test Environment and Infrastructure

The required test resources are the software tester and a laptop computer. Hence, the test environment is set and the specification of the device is as follows,

Processor	: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz
Installed RAM	: 16.0 GB (15.9 GB usable)
Device ID	: D0C34A9E-2997-4782-91D9-01FAA71E5041
Product ID	: 00342-41303-89601-AAOEM
System type	: 64-bit operating system, x64-based processor

Windows installed on the device is as follows,

Edition	: Windows 10 Home Single Language
Version	: 21H2
OS build	: 19044.1706
Experience	: Windows Feature Experience Pack 120.2212.4170.0

The testing tool used in the testing process are as follows,

IDE	: Eclipse IDE for Enterprise Java and Web Developers
Version	: 2022-03 (4.23.0)
Build id	: 20220310-1457
Runtime Server	: Apache Tomcat 9.0.54 Server
Testing Framework	: JUnit 5

2. Test design

Test design is the activity of deriving and specifying test cases from test conditions to test software. The purpose of a test design technique is to identify test conditions, test cases and test data.

The conditions in test cases are listed as follows:

2.1 Black box Testing

a) Input Domain :

<u>Age</u>

A1: Age ≥ 40

A2: Age < 40

A3: Age = "Dont care"

<u>Service Month</u>

B1: Service > 120 month

B2: Service ≤ 120 month

B3: Service = "Dont Care"

<u>Service Date</u>

C1: Service Commencement date before 1 April 1991 = “Y”

C2: Service Commencement date before 1 April 1991 = “N”

C2: Service Commencement date before 1 April 1991 = “Dont Care”

<u>Gender Female</u>

D1: Gender = ‘P’ = “Y”

D2: Gender = ‘P’ = “N”

D3: Gender = ‘P’ = “Dont Care”

<u>Male Special Officer</u>

E1 : Male Special Officer = “Y”

E2: Male Special Officer = “N”

E3: Male Special Officer = “Dont Care”

b) Output Domain:

<u>Eligibility</u>

F1: Benefit = 0, Calculate pension, Calculate gratuity, Calculate Cash Award

F2: Benefit = 1

F3: Benefit = 2

Pension Age

G1: Pension Age = 45

G2: Pension Age = 50

G3: Pension Age = 55

2.1.1 Equivalence Partitioning

a. **Benefit Eligibility**

Input Domain - Valid Partitioning

TC1: A1 && B1

Input Domain - Invalid Partitioning

TC2: A1 && B2

TC3: A2 && B1

TC4: A2 && B2

b. **Pension Age**

Input Domain

TC5: C1 && D1 && E1

TC6: C1 && D1 && E2

TC7: C1 && D2 && E1

TC8: C1 && D2 && E2

TC9: C2 && D1 && E1

TC10: C2 && D1 && E2

TC11: C2 && D2 && E1

TC12: C2 && D2 && E2

2.1.2 Boundary Value Analysis

a. **Benefit Eligibility**

Input Domain

TC13 - TC21 is generated based boundary on A1 and B2

Output Domain

TC22: F1

TC23: F2

TC24: F3

2.1.3 Decision Table

a. **Benefit Eligibility**

Conditions and Actions	**1**	**2**	**3**
Condition			
Age (year)	≥ 40	≥ 40	< 40
Service (month)	> 120	≤ 120	-
Actions			
Eligibility = 0	X		
Eligibility = 1			X
Eligibility = 2		X	
Calculate Pension	X		
Calculate Gratuity	X		
Calculate Cash Award	X		

Table 2.

TC1: A1 && B1

TC2: A1 && B2

TC25: A2 && B3

b. **Pension Age**

Conditions and Actions	**1**	**2**	**3**	**4**
Conditions				
Service Commencement date before 1 April 1991	Y	Y	Y	N
Gender = ‘P’	Y	N	N	-
Male Special Officer	-	Y	N	-
Actions				
Pension Age = 45	X	X		
Pension Age = 50			X	
Pension Age = 55				X

Table 3.

TC26: C1 && D1 && E3

TC7: C1 && D2 && E1

TC8: C1 && D2 && E2

TC27: C2 && D3 && E3

2.2 White Box Testing

White-box testing is a method of software testing that tests internal structures or workings of an application, as opposed to its functionality. Based on the code provided, test cases are generated using control structure testing.

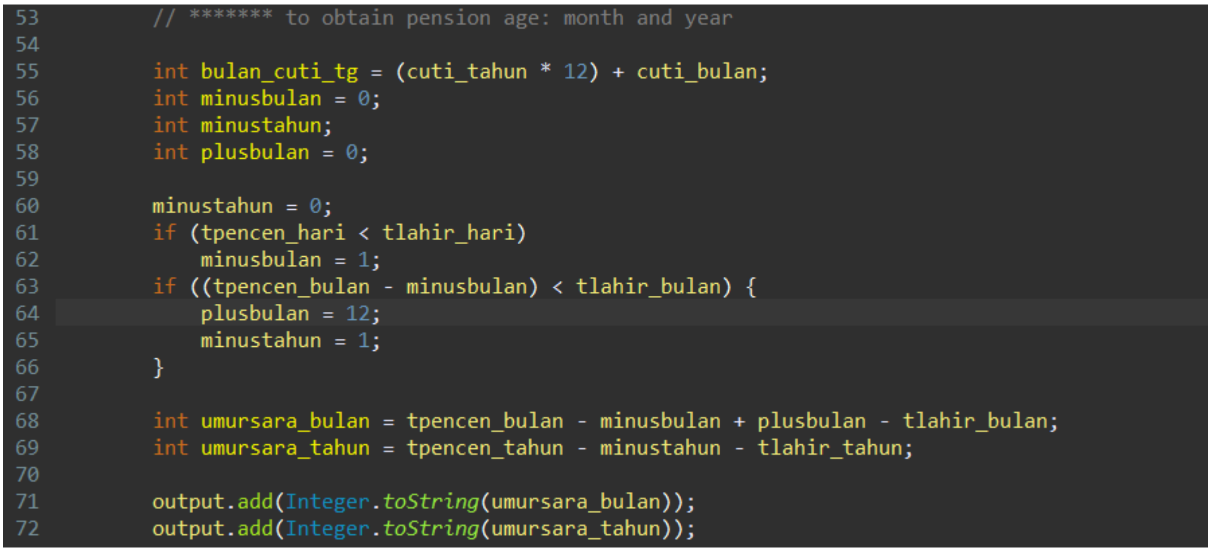
The Software Under Test (SUT) is PensionServiceIMP.java

2.2.1 Control Structure Testing

This technique tests on logical conditions in program module. It involves testing of both relational expressions and arithmetic expressions. Therefore, any if statements or conditional statement are tested based on statement, branch, condition and compound coverage. All statements without condition or branch statements are considered to be executed at least once. Repeating test cases are ignored if already covered by other coverage type.

The branched statements are listed as follows :

a. Determining 'umursara_bulan' and 'umursara_tahun'



```
        // ******* to obtain pension age: month and year

        int bulan_cuti_tg = (cuti_tahun * 12) + cuti_bulan;
        int minusbulan = 0;
        int minustahun;
        int plusbulan = 0;

        minustahun = 0;
        if (tpencen_hari < tlahir_hari)
            minusbulan = 1;
        if ((tpencen_bulan - minusbulan) < tlahir_bulan) {
            plusbulan = 12;
            minustahun = 1;
        }

        int umursara_bulan = tpencen_bulan - minusbulan + plusbulan - tlahir_bulan;
        int umursara_tahun = tpencen_tahun - minustahun - tlahir_tahun;

        output.add(Integer.toString(umursara_bulan));
        output.add(Integer.toString(umursara_tahun));
```

Figure 2.

Statement Coverage

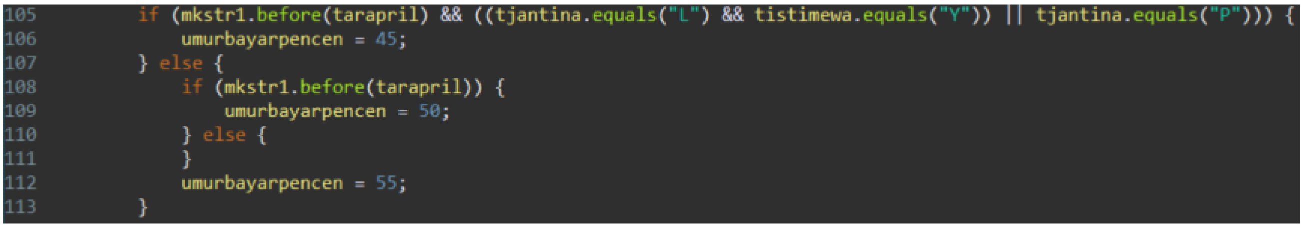
TC28: tlahir_hari = 12 && tpencen_hari = 9

tlahir_bulan = 2 && tpencen_bulan = 10

TC29: tlahir_bulan = 12 && tpencen_bulan = 10 && minusbulan= 0

tlahir_hari = 9 && tpencen_hari = 12

b. Determining Pension Age



```
105        if (mkstr1.before(tarapril) && ((tjantina.equals("L") && tistimewa.equals("Y")) || tjantina.equals("P"))) {
106            umurbayarpencen = 45;
107        } else {
108            if (mkstr1.before(tarapril)) {
109                umurbayarpencen = 50;
110            } else {
111            }
112            umurbayarpencen = 55;
113        }
```

Figure 3.

Statement Coverage

TC30: mkstr1 = Jan 2, 1989 && tjantina = "P" && tistimewa = "N"

TC31: mkstr1 = Jan 2, 1982 && tjantina ="L" && tistimewa = "N"

TC32: mkstr1 = 4 Oct, 2001 && jantina = "P" && tistimewa = "N"

Branch Coverage

TC31 covers false for branch line 105

TC32 covers false for branch line 108

TC30 covers false for branch line 107

Condition Coverage

TC33: mkstr1 = 4 Oct, 2001 && jantina = "L" && tistimewa = "Y"

Compound Condition Coverage

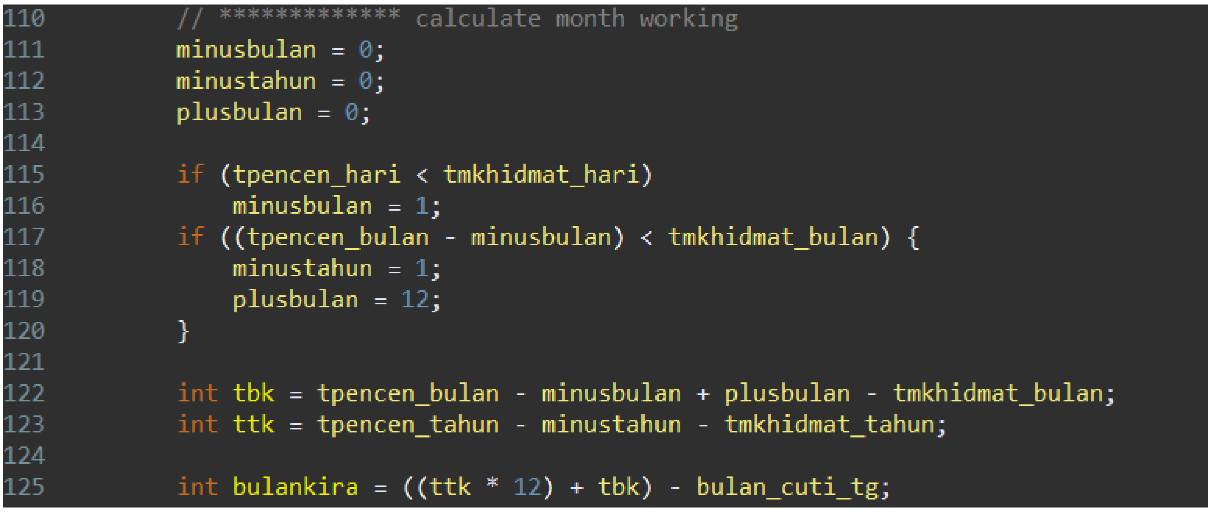
TC34: mkstr1 = 4 Oct, 1987 && jantina = "P" && tistimewa = "Y"

TC35: mkstr1 = 4 Oct, 1987 && jantina = "L" && tistimewa = "N"

TC36: mkstr1 = Jan 2, 1989 && jantina = "P" && tistimewa = "Y"

TC37: mkstr1 = Jan 2, 1989 && jantina = "L" && tistimewa = "Y"

c. Determining 'bulankira'



```
        // ************** calculate month working
        minusbulan = 0;
        minustahun = 0;
        plusbulan = 0;

        if (tpencen_hari < tmkhidmat_hari)
            minusbulan = 1;
        if ((tpencen_bulan - minusbulan) < tmkhidmat_bulan) {
            minustahun = 1;
            plusbulan = 12;
        }

        int tbk = tpencen_bulan - minusbulan + plusbulan - tmkhidmat_bulan;
        int ttk = tpencen_tahun - minustahun - tmkhidmat_tahun;

        int bulankira = ((ttk * 12) + tbk) - bulan_cuti_tg;
```

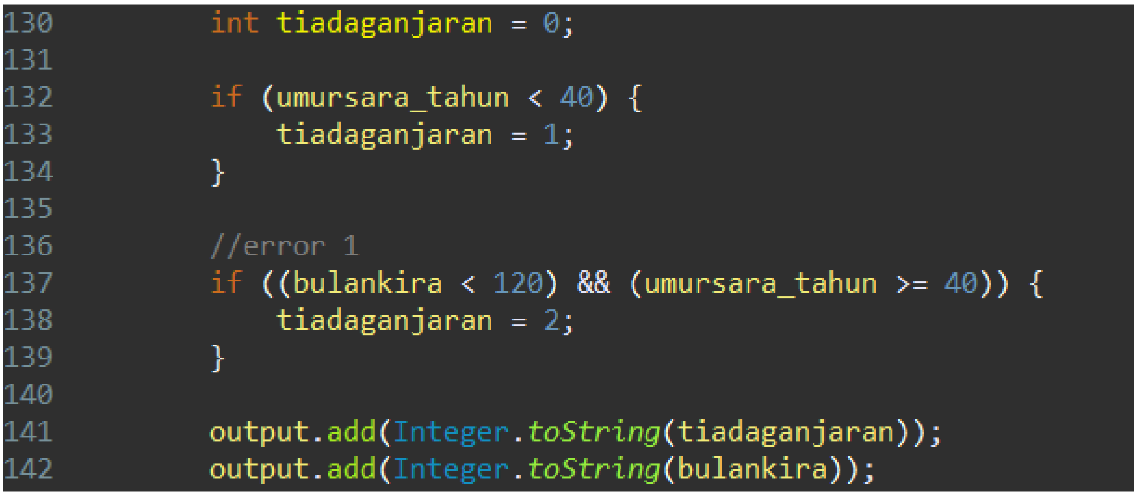
Figure 4.

Statement Coverage

TC38: tmkhidmat_hari = 12 && tpencen_hari = 9 tmkhidmat_bulan = 2 && tpencen_bulan = 10

TC39: tmkhidmat_bulan = 12 && tpencen_bulan = 10 && minusbulan= 0 tmkhidmat_hari = 9 && tpencen_hari = 12

d. Determining Benefit



```
int tiadaganjaran = 0;

if (umursara_tahun < 40) {
    tiadaganjaran = 1;
}

//error 1
if ((bulankira < 120) && (umursara_tahun >= 40)) {
    tiadaganjaran = 2;
}

output.add(Integer.toString(tiadaganjaran));
output.add(Integer.toString(bulankira));
```

Figure 5.

Statement

TC40: umursara_tahun = 30

TC41: bulankira = 115 && umursara_tahun = 45

Branch

TC42: umursara_tahun = 50

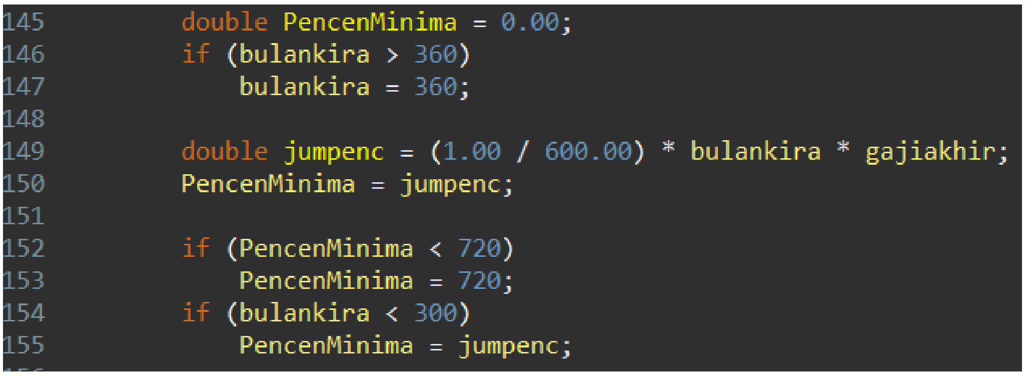
TC43: bulankira = 125 && umursara_tahun = 30

Compound

TC44: bulankira = 110 && umursara_tahun = 30

TC45: bulankira = 125 && umursara_tahun = 45

e. Determining Minimum Pension



```
double PencenMinima = 0.00;
if (bulankira > 360)
    bulankira = 360;

double jumpenc = (1.00 / 600.00) * bulankira * gajiakhir;
PencenMinima = jumpenc;

if (PencenMinima < 720)
    PencenMinima = 720;
if (bulankira < 300)
    PencenMinima = jumpenc;
```

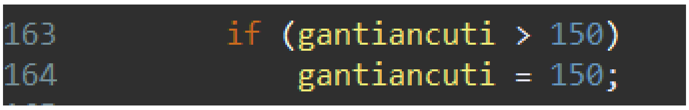
Figure 6.

Statement

TC46: bulankira = 400

TC47: bulankira = 120

f. Maximizing Cash Award Day



```
if (gantiancuti > 150)
    gantiancuti = 150;
```

Figure 7.

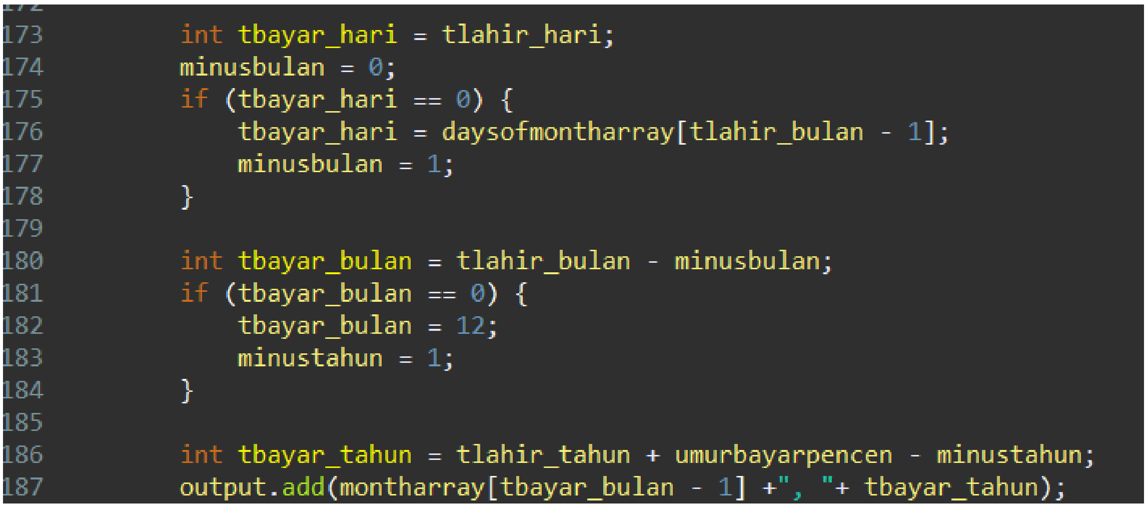
Statement

TC48: gantiancuti= 149

Branch

TC49: gantiancuti= 151

g. Calculating Start Pension Payment Date



```
int tbayar_hari = tlahir_hari;
minusbulan = 0;
if (tbayar_hari == 0) {
    tbayar_hari = daysofmontharray[tlahir_bulan - 1];
    minusbulan = 1;
}

int tbayar_bulan = tlahir_bulan - minusbulan;
if (tbayar_bulan == 0) {
    tbayar_bulan = 12;
    minustahun = 1;
}

int tbayar_tahun = tlahir_tahun + umurbayarpencen - minustahun;
output.add(montharray[tbayar_bulan - 1] +", "+ tbayar_tahun);
```

Figure 8.

Statement

TC50: tbayar_hari = 0 && tbayar_bulan = 1

Branch

TC51: tbayar_hari = 14 && tbayar_bulan = 0

3. Test Script

Test Scripts are line-by-line descriptions or test cases containing the information about the system transactions that should be performed to validate the system under test. A test case is a set of actions performed on a system to determine if it satisfies software requirements and functions correctly.

A test case aims to determine if different features within a system are performing as expected and to confirm that the system satisfies all related standards, guidelines, and customer requirements. Writing a test case can also help reveal errors or defects within the system. Test scripts are divided into black box and white box testing, according to their functionality. In order to isolate each technique, constant variables are used to ensure consistency in testing.

3.1 Black Box Testing

3.1.1 Benefit Eligibility

Constant Variables:

Last drawn salary = 3000.00
Fixed Allowance = 150.00
Cash Award Leave = 100 days
Unpaid leaves = 0 days
Gender : "P"
Special Officer ="N"
Optional Retirement Date: 21/5/2022

3.1.1.1 Equivalence Partitioning

Test Case ID	Input		Expected Output	Actual Output	Pass/ Fail	Remarks
	Age	Service				
TC1	46	150	Benefit = 0 Pension = 750.00 Gratuity = 33750 Cash Award = 10500	Benefit = 0 Pension = 750.00 Gratuity = 33750 Cash Award = 10500	PASS	
TC2	45	108	Benefit = 2	Benefit = 2	PASS	
TC3	35	121	Benefit = 1	Benefit = 1	PASS	
TC4	36	115	Benefit = 1	Benefit = 1	PASS	

Table 4.

3.1.1.2 Boundary Value Analysis

Test Case ID	Input		Expected Output	Actual Output	Pass/ Fail	Remarks
	Age	Service				
TC13	39	119	Benefit = 1	Benefit = 1	PASS	
TC14	39	120	Benefit = 1	Benefit = 1	PASS	
TC15/ TC23	39	121	Benefit = 1	Benefit = 1	PASS	
TC16	40	119	Benefit = 2	Benefit = 2	PASS	
TC17	**40**	**120**	**Benefit = 2**	**Benefit = 0**	**FAIL**	
TC18 / TC22	40	121	Benefit = 0 Pension = 605 Gratuity = 27225 Cash Award = 10500	Benefit = 0 Pension = 605 Gratuity = 27225 Cash Award = 10500	PASS	
TC19 / TC24	41	119	Benefit = 2	Benefit = 2	PASS	
TC20	**41**	**120**	**Benefit = 2**	**Benefit = 0**	**FAIL**	
TC21	41	121	Benefit = 0 Pension = 605 Gratuity = 27225 Cash Award = 10500	Benefit = 0 Pension = 605 Gratuity = 27225 Cash Award = 10500	PASS	

Table 5.

3.1.1.3 Decision Table

Test Case ID	Input		Expected Output	Actual Output	Pass/ Fail	Remarks
	Age	Service				
TC25	23	150	Benefit = 1	Benefit = 1	PASS	

Table 6.

3.1.2 Pension Age

Constant Variables:
Birthday = 12/10/1978
Last drawn salary = 3000.00
Fixed Allowance = 150.00
Cash Award Leave = 100 days
Unpaid leaves = 0 days
Optional Retirement Date: 21/5/2022

3.1.2.1 Equivalence Partitioning

Test Case ID	**Input**			**Expected Output**	**Actual Output**	**Pass/ Fail**	**Remark**
	Service Commencement date before 1 April 1991	**Gender**	**Male Special Officer**				
TC5	2/1/1988	P	Y	Pension Date = Oct, 2023 (Age = 45)	Pension Date = Oct, 2023 (Age = 45)	PASS	
TC6	2/1/1988	P	N	Pension Date = Oct, 2023 (Age = 45)	Pension Date = Oct, 2023 (Age = 45)	PASS	
TC7	2/1/1988	L	Y	Pension Date = Oct, 2023 (Age = 45)	Pension Date = Oct, 2023 (Age = 45)	PASS	
TC8	**2 /1/1988**	**L**	**N**	**Pension Date = Oct, 2028 (Age = 50)**	**Pension Date = Oct, 2033 (Age = 50)**	**FAIL**	
TC9	4/10/2001	P	Y	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	
TC10	4/10/2001	P	N	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	

TC11	4/10/2001	L	Y	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	
TC12	4/10/2001	L	N	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	

Table 7.

3.1.2.2 Decision Table

Test Case ID	Input			Expected Output	Actual Output	Pass/ Fail	Remarks
	Service Commencement date before 1 April 1991	**Gender**	**Male Special Officer**				
TC26	2 /1/1988	P	Y	Pension Date = Oct, 2023 (Age = 45)	Pension Date = Oct, 2023 (Age = 45)	PASS	
TC27	4/10/2001	P	N	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	

Table 8.

3.2 White Box Testing

3.2.1 Control Structure Testing

a. Determining 'umursara_bulan' and 'umursara_tahun'
Constant Variables:
Service Date = 4/10/2001
Last drawn salary = 3000.00
Fixed Allowance = 150.00
Cash Award Leave = 100 days
Unpaid leaves = 0 days
Gender : "P"
Special Officer ="N"

Test Case ID	Input		Expected Output	Actual Output	Pass /Fail	Remark
	Birthday	Optional Retirement Date				
TC28	12/2/1978	9/10/2022	Pension Month = 7 Pension Year = 44	Pension Month = 7 Pension Year = 44	PASS	
TC29	9/12/1978	12/10/2022	Pension Month = 10 Pension Year = 43	Pension Month = 10 Pension Year = 43	PASS	

Table 9.

b. Determining Pension Age
Constant Variables:
Birthday : 12 Oct 1978
Last drawn salary = 3000.00
Fixed Allowance = 150.00
Cash Award Leave = 100 days
Unpaid leaves = 0 days
Optional Retirement Date: 21/5/2022

Test Case ID	Input			Expected Output	Actual Output	Pass/ Fail	Remark
	Service Commencement date before 1 April 1991	Gender	Male Special Officer				
TC30	2/1/1988	P	N	Pension Date = Oct, 2023 (Age = 45)	Pension Date = Oct, 2023 (Age = 45)	PASS	

TC31	**2/1/1988**	**L**	**N**	**Pension Date = Oct, 2028 (Age = 50)**	**Pension Date = Oct, 2033 (Age = 50)**	**FAIL**	
TC32	4/10/2001	P	N	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	
TC33	4/10/2001	L	Y	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	
TC34	4/10/2001	P	Y	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	
TC35	4/10/2001	L	N	Pension Date = Oct, 2032 (Age = 55)	Pension Date = Oct, 2032 (Age = 55)	PASS	
TC36	2/1/1988	P	Y	Pension Date = Oct, 2023 (Age = 45)	Pension Date = Oct, 2023 (Age = 45)	PASS	
TC37	2/1/1988	L	Y	Pension Date = Oct, 2023 (Age = 45)	Pension Date = Oct, 2023 (Age = 45)	PASS	

Table 10.

c. Determining 'bulankira'

Constant Variables:

Birthday = 2/1/1969
Last drawn salary = 3000.00
Fixed Allowance = 150.00
Cash Award Leave = 100 days
Unpaid leaves = 0 days
Gender : "P"
Special Officer = "N"

Test Case ID	Input		Expected Output	Actual Output	Pass/Fail	Remarks
	Service Date	Optional Retirement Date				
TC38	12/2/1978	9/10/2022	bulankira = 256	bulankira = 256	PASS	
TC39	9/12/1978	1/10/2022	bulankira = 257	bulankira = 257	PASS	

Table 11.

d. Determining Benefit

Constant Variables:

Service Date = 4/10/2001
Last drawn salary = 3000.00
Fixed Allowance = 150.00
Cash Award Leave = 100 days
Unpaid leaves = 0 days
Gender : "P"
Special Officer ="N"
Optional Retirement Date: 21/5/2022

Test Case ID	Input		Expected Output	Actual Output	Pass /Fail	Remarks
	Birthday	Service Date				
TC40	2/1/1992	5/10/ 2012	Benefit = 1	Benefit = 1	PASS	
TC41	2/1/1977	5/10/2012	Benefit = 2	Benefit = 2	PASS	
TC42	4/4/1972	7/12/ 2011	Benefit = 0	Benefit = 0	PASS	
TC43	4/4/1992	7/12/2011	Benefit = 1	Benefit = 1	PASS	
TC44	6/3/1992	14/3/2013	Benefit = 1	Benefit = 1	PASS	
TC45	6/3/1977	7/12/10 2011	Benefit = 0	Benefit = 0	PASS	

Table 12.

e. Determining Minimum Pension

Constant Variables:

Service Date = 4/10/2001
Birthday : 12 Oct 1978
Last drawn salary = 2000.00
Fixed Allowance = 150.00
Cash Award Leave = 100 days
Unpaid leaves = 0 days
Optional Retirement Date: 21/5/2022
Gender : "P"
Special Officer ="N"

Test Case ID	Input		Expected Output	Actual Output	Pass/ Fail	Remarks
	Service Date	Pension				
TC46	2/1/1989	1200	Minimum Pension = 1200	Minimum Pension = 1200	PASS	
TC47	14/5/ 2012	400	Minimum Pension = 400	Minimum Pension = 400	PASS	

Table 13.

f. Maximizing Cash Award Day
Constant Variables:
Service Date = 4/10/2001
Birthday : 12 Oct 1978
Last drawn salary = 2000.00
Fixed Allowance = 150.00
Unpaid leaves = 0 days
Service D
Optional Retirement Date: 21/5/2022
Gender : "P"
Special Officer ="N"

Test Case ID	Input	Expected Output	Actual Output	Pass /Fail	Remarks
	Cash Award Day				
TC48	gantiancuti= 149	gantiancuti= 149	gantiancuti= 149	PASS	
TC49	gantiancuti= 151	gantiancuti= 150	gantiancuti= 150	PASS	

Table 14.

g. Calculating Start Pension Payment Date
Constant Variables:
Last drawn salary = 2000.00
Fixed Allowance = 150.00
Cash Award Leave = 100 days
Unpaid leaves = 0 days
Service Date : 2 Jan 1989
Optional Retirement Date: 21/5/2022
Gender : "P"
Special Officer ="N"
Due to these conditions, umurpencen = 45

Test Case ID	Input	Expected Output	Actual Output	Pass/ Fail	Remarks
	Birthday				
TC50	0/1/1978	Pension Payment Start Date = Dec, 2022	Pension Payment Start Date = Dec, 2022	PASS	
TC51	14/5/1978	Pension Payment Start Date = May,2023	Pension Payment Start Date = May,2023	PASS	

Table 15.

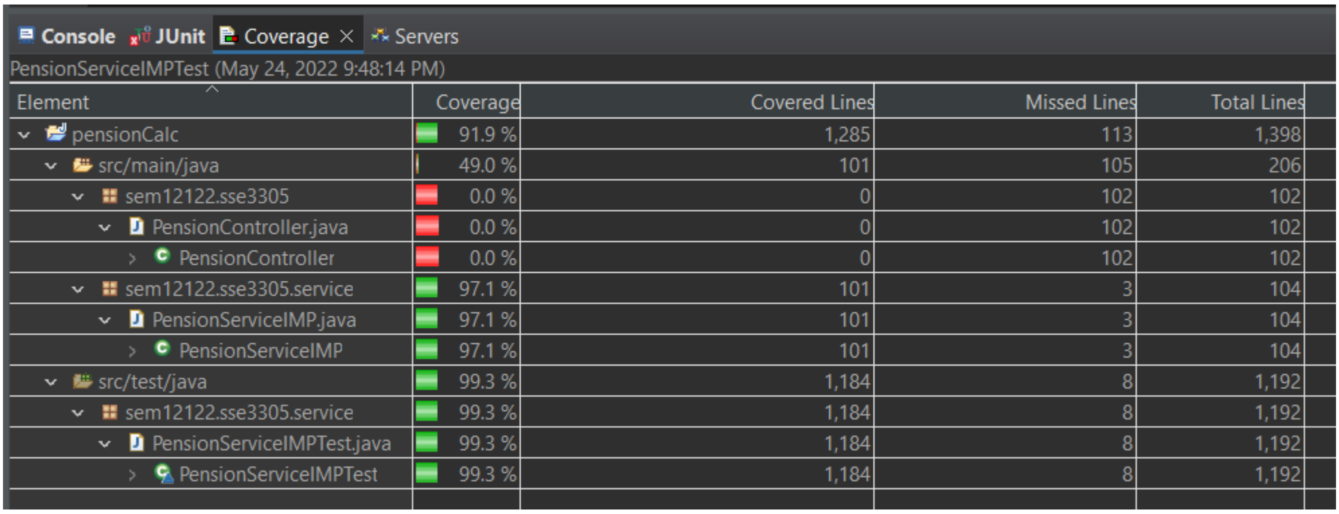
4. JUnit Test class

A JUnit test is a method contained in a class which is only used for testing. The test class that consists of test methods are uploaded to GitHub using the following link. The test class could also be accessed by scanning the QR code.

https://github.com/juliaxy0/SSE3305/blob/main/Individual%20Assignment/PensionServiceIMPTest

Figure 9.

5. Coverage Information



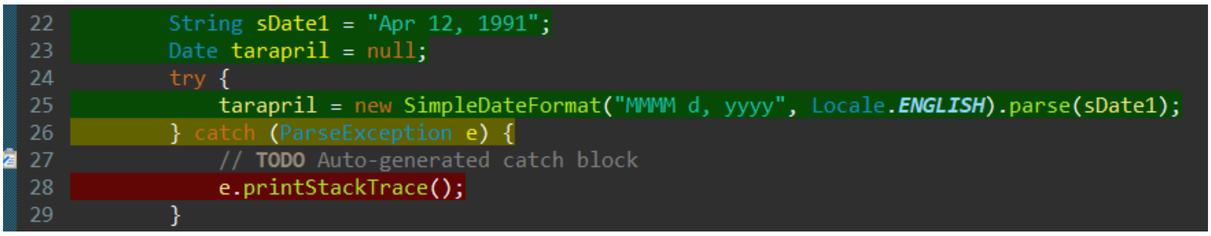
PensionServiceIMPTest (May 24, 2022 9:48:14 PM)

Element	Coverage	Covered Lines	Missed Lines	Total Lines
pensionCalc	91.9 %	1,285	113	1,398
src/main/java	49.0 %	101	105	206
sem12122.sse3305	0.0 %	0	102	102
PensionController.java	0.0 %	0	102	102
PensionController	0.0 %	0	102	102
sem12122.sse3305.service	97.1 %	101	3	104
PensionServiceIMP.java	97.1 %	101	3	104
PensionServiceIMP	97.1 %	101	3	104
src/test/java	99.3 %	1,184	8	1,192
sem12122.sse3305.service	99.3 %	1,184	8	1,192
PensionServiceIMPTest.java	99.3 %	1,184	8	1,192
PensionServiceIMPTest	99.3 %	1,184	8	1,192

Figure 10.

Based on the figure above, lines in PensionServiceIMPTest class are covered 99.3% and PensionServiceIMP class is covered 97.1%. Each line in PensionServiceIMPTest class is successfully executed and covered except for 8 lines which are the four (4) false assertEquals() method and their curly closed bracket accordingly. This is due to four test cases that failed the testing process.

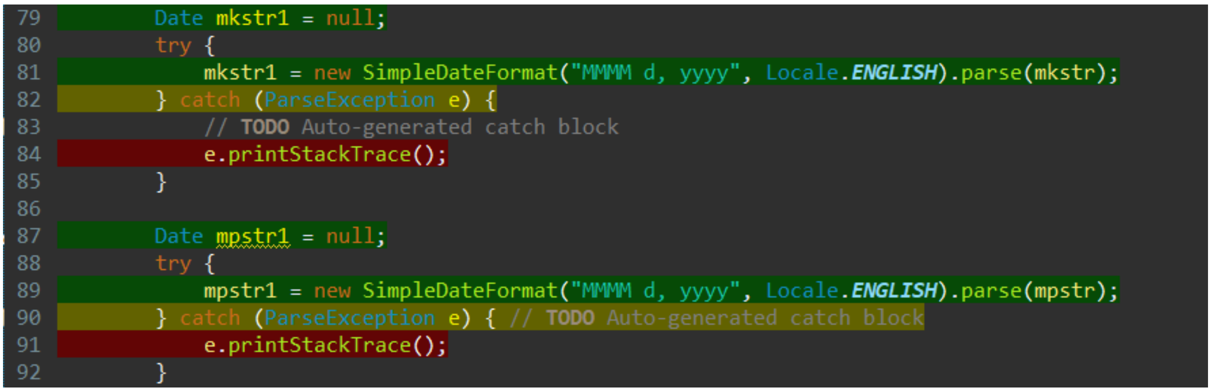
On the other hand, the PensionServiceIMP class is successfully executed and covered except for 3 lines which are catch blocks in lines 28, 84 and 91. It is impossible to trigger the catch block from the user side since the input is already filtered by the system. In catch block line 28 as shown in Figure 11, the parameter of 'sDate1' is already set by the developer. Hence, the only error that could be caused is by the system itself.



```
22         String sDate1 = "Apr 12, 1991";
23         Date tarapril = null;
24         try {
25             tarapril = new SimpleDateFormat("MMMM d, yyyy", Locale.ENGLISH).parse(sDate1);
26         } catch (ParseException e) {
27             // TODO Auto-generated catch block
28             e.printStackTrace();
29         }
```

Figure 11.

In catch block line 84 and 91 as shown in Figure 12, it is not easy to trigger the catch block since the form is already filtering user input through form validation. Therefore, error input such as alphabets instead of integers will not pass through the system from the start. Hence, three catch blocks are not covered by PensionServiceIMPTest class.



```
79          Date mkstr1 = null;
80          try {
81              mkstr1 = new SimpleDateFormat("MMMM d, yyyy", Locale.ENGLISH).parse(mkstr);
82          } catch (ParseException e) {
83              // TODO Auto-generated catch block
84              e.printStackTrace();
85          }
86  
87          Date mpstr1 = null;
88          try {
89              mpstr1 = new SimpleDateFormat("MMMM d, yyyy", Locale.ENGLISH).parse(mpstr);
90          } catch (ParseException e) { // TODO Auto-generated catch block
91              e.printStackTrace();
92          }
```

Figure 12.

In other words, the testing coverage of test methods in PensionServiceIMPTest class has been maximized based on PensionServiceIMP class. Each statement and branches are covered excluding the catch blocks.

6. Concluding Remarks

In the end, 51 unique test cases were generated using four testing techniques. Four test cases face failure, due to the actual output being different to the expected output. The failed test cases are as follows,

1. Boundary Value Analysis test case 17 and test case 20 (page 15)
2. Equivalence Partitioning Testing test case 8 (page 16)
3. Control Structure Testing test case 31 (page 18)

Based on the failed test cases, two defects were found in the system under test which are listed in table 16.

No	Line	Defect	Corrected Code

1.	137 - 139	if ((bulankira < 120) &&
		(umursara_tahun >= 40)) {
			if ((bulankira <= 120) &&
			(umursara_tahun >= 40)) {

2.	100 - 107	if (mkstr1.before(tarapril) &&
		((tjantina.equals("L") &&
		tistimewa.equals("Y")) ||
		tjantina.equals("P"))) {
		
		umurbayarpencen = 45;
		
		} else {
		if (mkstr1.before(tarapril)) {
		umurbayarpencen = 50;
		} else {
		}
		umurbayarpencen = 55;
		}
			if (mkstr1.before(tarapril) &&
			((tjantina.equals("L") &&
			tistimewa.equals("Y")) ||
			tjantina.equals("P"))) {
			
			umurbayarpencen = 45;
			
			} else {
			if (mkstr1.before(tarapril)) {
			umurbayarpencen = 50;
			} else {
			umurbayarpencen = 55;
			}
			}

Table 16.

In order to increase code coverage and any possible input values, four techniques were chosen. Firstly, Black-box Testing Techniques are considered based on the Test Basis. Equivalence Partitioning Testing is chosen since it ensures the selection of the right test cases to cover all possible scenarios. This is whereby equivalence classes are used to represent certain conditions on the input domain. Therefore, the number of test cases is reduced to the necessary minimum.

However, this is not sufficient since the output domain and boundary values are not exercised. Therefore, Boundary Value Analysis is used to generate complemented test cases of Equivalence Partitioning Testing. Therefore, not only minimum and maximum numbers in input conditions are exercised. The values just above and just below the minimum and maximum values are thoroughly exercised as well.

Since the system also requires complicated decision logic such as determining benefit eligibility and pension age of the user, the two chosen techniques are still insufficient. Therefore, Decision Table Testing is implied to cover any other possible conditions for the decision and resulting actions including conditions that are not affected by any type of input or simply 'Don't Care'. Therefore, all possible conditions and actions for each rule are considered in generating test cases. This method almost imitates the coverage of the Cause-Effect Graph since both map out input to output. Hence, Cause-Effect Graph is skipped since Decision Table Testing is already applied.

Next, White-box Testing is approached based on the system under test code. This is to examine whether the examined code works correctly as expected. This is achieved by testing the logical paths of the system by exercising a specific sequence of instructions. conditions and loops. Therefore, all statements and conditions will be executed at least once. A few techniques were considered such as Basis Path Testing and Control Structure Testing. Since there are too many possible paths to be listed down which suggest errors in generating test cases, Control Structure Testing is preferred over Basis Path Testing.

Other than a clearer generation of test cases, Control Structure Testing also ensures to increase the coverage of code such as statement, branch, condition and compound coverage. By applying these four techniques, the most possible inputs and conditions were considered in generating test cases. The test cases are then executed by using the JUnit tool which eases the process. Hence, it is sufficient to say that the strategy and technique proposed for the testing process are the most effective in testing the given system.

Until the defect in the system is corrected such as suggested in Table 16, the tested code is not ready for deployment. This is due to the fact that four test cases face failure during the testing process. Therefore, there is a possibility that the user will receive incorrect output unknowingly. This will cause more difficulty in the future if the code is deployed as it is.