**COMP 370 – Software Engineering**  
**Term Project**  
Problem 11.24

# 15.33

*Draw up black-box test cases for the product you specified in Problem*[*12.20*](file:///C:\Users\Thomas\AppData\Local\atom\app-1.6.0\resources\app.asar\static\index.html#1220)*or*[*13.22*](file:///C:\Users\Thomas\AppData\Local\atom\app-1.6.0\resources\app.asar\static\index.html#1322)*. For each test case, state what is being tested and the expected outcome of that test case. Black-Box Tests* The functions outlined in the specifications document are outlined in the test cases:

## Black-Box Test Cases

**Database**

Equivalence classes for *Database password*.

| **Password** | **Result** |
| --- | --- |
| Password is string | true |
| Password's first character is alphanumeric | true |
| Password < 1 character | true |

| **Database.member\_number** | **Result** |
| --- | --- |
| member\_number is created | true |
| member\_number cannot be set as "0" | acceptable |
| member\_number can be "1" | true |
| member\_number can be "999999999" | true |
| member\_number cannot be "1000000000" | acceptable |
| member\_number cannot be "a" | acceptable |
| member\_number cannot be "" | acceptable |

| **Database.member\_name** | **Result** |
| --- | --- |
| member\_name is created | true |
| member\_name can be "m" | true |
| member\_name can be "1" | true |
| member\_name can be "xxxx xxxx xxxx xxxx xxxx" | true |
| member\_name cannot be "xxxx xxxx xxxx xxxx xxxx xxxx" | true |
| member\_name cannot be "" | acceptable |

... Repeated for member\_street\_address, member\_city, member\_postal\_code, member\_email\_address.

| **Database.member\_province** | **Result** |
| --- | --- |
| member\_province is created | true |
| member\_province can be "BC" | true |
| member\_province cannot be "British Columbia" | acceptable |
| member\_province cannot be "CB" | acceptable |
| member\_province cannot be "C-B" | acceptable |
| member\_province cannot be "B.C." | acceptable |
| member\_province cannot be "NWT" | acceptable |
| member\_province cannot be "CB" | acceptable |

We are testing the following code here:

if ($this->getLength($province) > self::PROVINCE\_LENGTH) //p<2

{

$province = strtoupper($province);

if ($province[0] === "A") $province = "AB";

if ($province[0] === "B") $province = "BC";

if ($province[0] === "C") $province = "BC";

if ($province[0] === "M") $province = "MB";

if ($province[0] === "N" && strpos($province, 'B' !== false)) $province = "NB";

if ($province[0] === "N" && strpos($province, 'L' !== false)) $province = "NL";

if ($province[0] === "N" && strpos($province, 'U') !== false) $province = "NU";

if ($province[0] === "N" && $this->getLength($province) > self::PROVINCE\_LENGTH) $province = "NS";

if ($province[0] === "N" && strpos($province, 'T' !== false)) $province = "NT";

if ($province[0] === "O") $province = "ON";

if ($province[0] === "C") $province = "QC";

if ($province[0] === "S") $province = "SK";

if ($province[0] === "Y") $province = "YT";

}

| **Database.member\_status** | **Result** |
| --- | --- |
| member\_status is created | true |
| member\_status can be "A" | true |
| member\_status can be "S" | true |
| member\_status cannot be "Suspended" | acceptable |
| member\_status cannot be "Active" | acceptable |
| member\_status cannot be "a" | acceptable |
| member\_status cannot be "s" | acceptable |
| member\_status cannot be "0" | true |
| member\_status cannot be "null" | true |

This is repeated for Provider, except provider\_status is tested as follows:

| **Database.provider\_type** | **Result** |
| --- | --- |
| provider\_type is created | true |
| provider\_type can be "I" | true |
| provider\_type can be "E" | true |
| provider\_type can be "D" | true |
| provider\_type cannot be "Dietitian" | acceptable |
| provider\_type cannot be "Internist" | acceptable |
| provider\_type cannot be "Exercise Expert" | acceptable |
| provider\_type cannot be "" | true |
| provider\_type cannot be "0" | true |
| provider\_type cannot be "null" | true |

In both cases above, if the first letter of the string matches an acceptable value, it is chosen.

**Other Database Tests**

| **Test** | **Result** |
| --- | --- |
| Test if database accepts login | true |
| Test if 'claim' database is working | acceptable |
| Asserts the member database table is working. | true |
| Asserts the provider database table is working. | true |
| Asserts the service database table is working. | true |
| Test unique keys (run on claim timestamp key) | acceptable |
| Test unique keys (run on member key) | acceptable |
| Test unique keys (run on provider key) | acceptable |
| Test unique keys (run on service key) | acceptable |

**General tests**

| **Test** | **Result** |
| --- | --- |
| Asserts SERVER[REMOTE\_ADDR] is set, it's needed for testing and Database connectivity. | true |
| Test PHP's MySQLi plugin loads. | true |
| Checks if mysqli exists (but potentially hasn't been started) | true |
| Test that the calculation in private function Person()->getLength() is correct | true |

**Member Tests**

| **Member Test** | **Result** |
| --- | --- |
| Create a member that already exists | acceptable |
| Asserts members can be found in the database. | true |
| Asserts postal codes are in correct Canadian format. | true |

**Provider Tests**

| **Provider Test** | **Result** |
| --- | --- |
| Asserts postal codes are in correct Canadian format. | true |

**Person Tests**

| **Person Test** | **Result** |
| --- | --- |
| Checks if getting all members works. Also stress-tests mysql with a lot of queries at once. | acceptable |
| Checks if getting all providers works. Also stress-tests mysql with a lot of queries at once. | acceptable |
| Asserts that a database entry will be created for dummy example member. | true |

## Functional Analysis Test Cases

1. Manage Session
2. Verify Member
3. Submit Claim
4. Receive Order
5. Receive Provider Directory
6. Maintain Member (Add a new member)
7. Maintain Member (Update existing member)
8. Maintain Member (Delete existing member)
9. Maintain Provider (Add a new provider)
10. Maintain Provider (Update existing provider)
11. Maintain Provider (Delete existing provider)
12. Maintain Service (Add a service)
13. Maintain Service (update a service)
14. Maintain service (delete service)
15. Request Report (provider report)
16. Request Report (member report)
17. Request Report (accounts payable report) (Not implemented in project)

In addition to these direct tests, it is necessary to perform the following additional tests:

1. Attempt to create a Member
2. Attempt to create a Member that already exists.

All tests pass. Tests may be run at (http://COMP370.thomasmclennan.ca/tests/).