

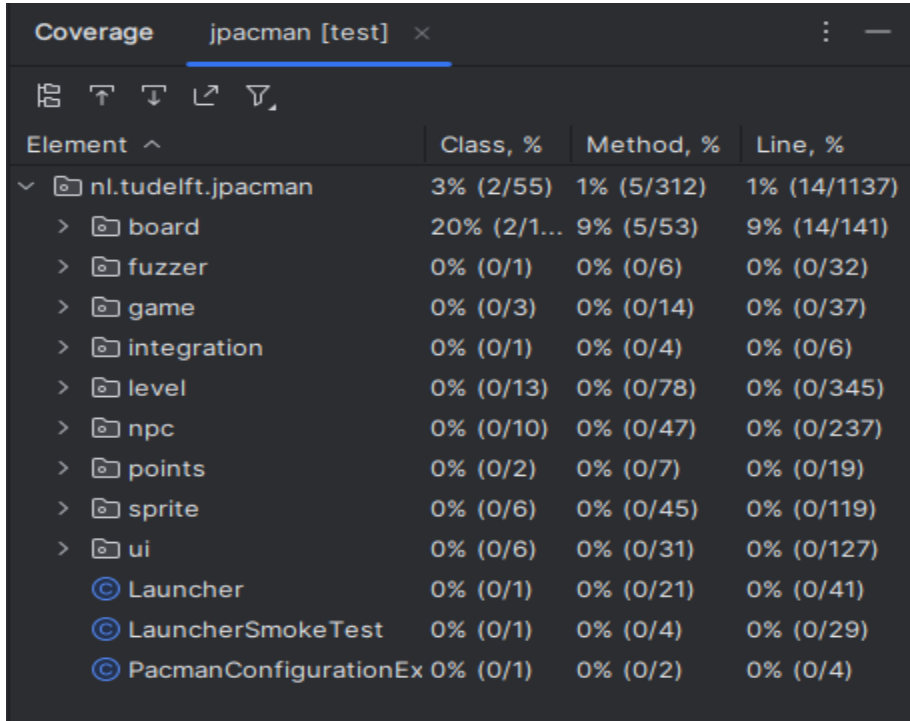
## Testing Lab

CS 472 - 1001

Justin Vence Llamas

Link to fork repo - <https://github.com/jusasin/cs472-group4>

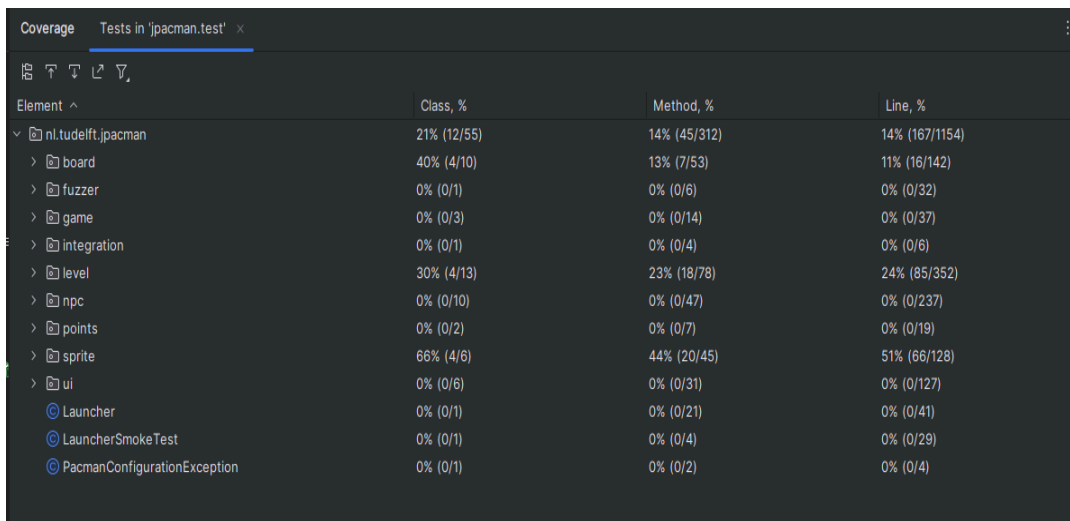
### Task 1 - JPacman Test Coverage (Can be used as a before picture reference)



Element ^	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	3% (2/55)	1% (5/312)	1% (14/1137)
> board	20% (2/10)	9% (5/53)	9% (14/141)
> fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
> game	0% (0/3)	0% (0/14)	0% (0/37)
> integration	0% (0/1)	0% (0/4)	0% (0/6)
> level	0% (0/13)	0% (0/78)	0% (0/345)
> npc	0% (0/10)	0% (0/47)	0% (0/237)
> points	0% (0/2)	0% (0/7)	0% (0/19)
> sprite	0% (0/6)	0% (0/45)	0% (0/119)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
⦿ Launcher	0% (0/1)	0% (0/21)	0% (0/41)
⦿ LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
⦿ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

- The coverage is not good enough, majority of the coverage is pretty low. Most of them being 0%.

### Task 2.1 - After unit test picture



Element ^	Class, %	Method, %	Line, %
✓ nl.tudelft.jpacman	21% (12/55)	14% (45/312)	14% (167/1154)
> board	40% (4/10)	13% (7/53)	11% (16/142)
> fuzzer	0% (0/1)	0% (0/6)	0% (0/32)
> game	0% (0/3)	0% (0/14)	0% (0/37)
> integration	0% (0/1)	0% (0/4)	0% (0/6)
> level	30% (4/13)	23% (18/78)	24% (85/352)
> npc	0% (0/10)	0% (0/47)	0% (0/237)
> points	0% (0/2)	0% (0/7)	0% (0/19)
> sprite	66% (4/6)	44% (20/45)	51% (66/128)
> ui	0% (0/6)	0% (0/31)	0% (0/127)
⦿ Launcher	0% (0/1)	0% (0/21)	0% (0/41)
⦿ LauncherSmokeTest	0% (0/1)	0% (0/4)	0% (0/29)
⦿ PacmanConfigurationException	0% (0/1)	0% (0/2)	0% (0/4)

- Test coverage after completing unit tests for three methods. The three methods that I decided to work on was in Level.java: start, stop, registerPlayer methods. There were actually more methods that were affected by my unit test in Level but those were the three I mainly focused on.

```
@Test
public void testingPlayerRegister() throws NoSuchFieldException, IllegalAccessException{
    //create a new player
    Player player = mock(Player.class);

    //create starting squares and run a mock
    CollisionMap collisionMap = mock(CollisionMap.class);
    Square sq1 = mock(Square.class);
    Square sq2 = mock(Square.class);
    List<Square> startSquares = List.of(sq1, sq2);
    Level level = new Level(level.getBoard(), new ArrayList<>(), startSquares, collisionMap);

    //Testing when the player is not in the list and squares are not empty
    level.registerPlayer(player);

    //had to search up what reflection was to access the private 'players' field
    Field playersField = Level.class.getDeclaredField( name: "players");
    playersField.setAccessible(true);
    List<Player> players = (List<Player>) playersField.get(level);

    //Check to see if the player was added and is in the game
    assertTrue(players.contains(player));
    Mockito.verify(player).occupy(sq1);

    //Try to add the same player to the list if they are already there
    level.registerPlayer(player);

    //Did the player count change
    assertEquals( expected: 1,players.size());
}
```

- Code Snippet of RegisterPlayer: The above code will test if the player is already on the list for the level, giving them a starting position, and checking if they are already listed once, listing them again should not cause an issue.

```

//Test to check that the game will stop correctly in progress
@Test
public void testStopInProgress() throws NoSuchFieldException, IllegalAccessException {
    //create mock variables
    ScheduledExecutorService scheduledExecutorService = mock(ScheduledExecutorService.class);
    Ghost ghost = mock(Ghost.class);

    //Have to use reflection again to set npcs
    Field npcsField = Level.class.getDeclaredField(name: "npcs");
    npcsField.setAccessible(true);
    Map<Ghost, ScheduledExecutorService> npcs = (Map<Ghost, ScheduledExecutorService>) npcsField.get(level);

    //input the ghost into npc
    npcs.put(ghost, scheduledExecutorService);

    //begin the game
    level.start();
    assertTrue(level.isInProgress());

    //stop the level
    level.stop();

    //check that the level was stopped
    assertFalse(level.isInProgress());

    //had to search up a different way to check instead of using verify that no interaction happened
    Mockito.verifyZeroInteractions(scheduledExecutorService);
}

```

```

//Test the function of the game where it should not stop when not in progress
//Same layout but dont start the game
@Test
public void testStopNotInProgress() throws NoSuchFieldException, IllegalAccessException {
    //create mock variables
    ScheduledExecutorService executorService = mock(ScheduledExecutorService.class);
    Ghost ghost = mock(Ghost.class);

    //Have to use reflection again to set npcs
    Field npcsField = Level.class.getDeclaredField(name: "npcs");
    npcsField.setAccessible(true);
    Map<Ghost, ScheduledExecutorService> npcs = (Map<Ghost, ScheduledExecutorService>) npcsField.get(level);

    //input the ghost into npc
    npcs.put(ghost, executorService);

    //stop the level
    level.stop();

    //check scheduledexecutorservice was not called
    verify(executorService, never()).shutdownNow();

    //check that the level is still not in progress
    assertFalse(level.isInProgress());
}

```

- Code Snippet for Stop and Start (Effects both based on coverage): The first image with the function called “testStopInProgress” is starting a game then stopping it then checking for if that level was stopped properly. No movement was being made and all NPCs came

to a halt. Had to verify that the `shutDownNow()` function was called on. The second image with the function called “`testStopNotInProgress`” does the same thing but instead it does not start the level, it only tries to stop it. I used verify to check if `shutDownNow()` was not called on and the game was still not in progress.

### Task 3 - JaCoCo Report on JPacman

`nl.tudelft.jpacman.level`

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Level		86%		70%	25	55	6	105	1	15	0	1
CollisionInteractionMap		0%		0%	19	19	46	46	7	7	1	1
MapParser		87%		78%	7	26	7	69	1	10	0	1
PlayerCollisions		75%		57%	5	14	6	28	1	7	0	1
LevelFactory.RandomGhost		0%		0%	6	6	12	12	3	3	1	1
Player		85%		66%	3	11	3	24	1	8	0	1
LevelFactory		89%		80%	1	8	1	17	0	4	0	1
Level.NpcMoveTask		100%		100%	0	3	0	10	0	2	0	1
DefaultPlayerInteractionMap		0%		n/a	5	5	17	17	5	5	1	1
CollisionInteractionMap.InverseCollisionHandler		0%		n/a	2	2	5	5	2	2	1	1
PlayerFactory		100%		n/a	0	3	0	5	0	3	0	1
Pellet		100%		n/a	0	3	0	6	0	3	0	1
Total	438 of 1,365	67%	69 of 165	58%	73	155	103	344	21	69	4	12

- The coverage results from JaCoCo and the coverage result from IntelliJ are completely different. After doing the Unit Tests for the three methods, most of the coverage percentages from IntelliJ were either 0 or had very little coverage. While JaCoCo’s coverage results are showing what instructions were missed and what branches. It even color codes it to show specific lines for the coverage.
- I did find it helpful that the source code visualization from JaCoCo was color coded and neatly presented on uncovered branches.
- I prefer looking at JaCoCo’s report in terms of visualization because it gives me a clear picture of what lines are covered, which ones aren’t and I am not quite sure what highlighted yellow lines mean. IntelliJ isn’t bad either because its already incorporated in the framing of my IDE so I don’t have to open anything else like a website like JaCoCo has.

### Task 4

The following task had us create test cases for missing lines given. The following images are code snippets of those test cases that I created. I also have a screenshot showing that I was able to accomplish 100% coverage.

```
56 def test_repr(self):
57     """Test the representation of an account"""
58     account = Account()
59     account.name = "Foo"
60     self.assertEqual(str(account), "<Account 'Foo'>")
61
62 def test_to_dict(self):
63     """ Test account to dict """
64     data = ACCOUNT_DATA[self.rand] # get a random account
65     account = Account(**data)
66     result = account.to_dict()
67     self.assertEqual(account.name, result["name"])
68     self.assertEqual(account.email, result["email"])
69     self.assertEqual(account.phone_number, result["phone_number"])
70     self.assertEqual(account.disabled, result["disabled"])
71     self.assertEqual(account.date_joined, result["date_joined"])
72
73 def test_from_dict(self):
74     """Test if dictionary is set from given attributes"""
75     account_data = {
76         'name': 'First Last',
77         'email': 'first.last@fake.com',
78         'phone_number': '808422222',
79         'disabled': True,
80     }
81     account = Account()
82     account.from_dict(account_data)
83     self.assertEqual(account.name, 'First Last')
84     self.assertEqual(account.email, 'first.last@fake.com')
85     self.assertEqual(account.phone_number, '808422222')
86     self.assertEqual(account.disabled, True)
```

```
88     def test_update(self):
89         """Test if data can be updated"""
90         account_data = {
91             'name': 'First Last',
92             'email': 'first.last@fake.com',
93             'phone_number': '808422222',
94             'disabled': True,
95         }
96         account = Account()
97         account.from_dict(account_data)
98         account.create()
99         account.email = 'first.last@real.com'
100        account.update()
101        self.assertNotEqual(account.email, 'first.last@fake.com')
102
103    def test_update_noId(self):
104        """Test if data can be updated without id"""
105        account = Account()
106        with self.assertRaises(DataValidationError):
107            account.update()
108
109    def test_delete(self):
110        """Test if account can be deleted"""
111        account_data = {
112            'name': 'First Last',
113            'email': 'first.last@fake.com',
114            'phone_number': '808422222',
115            'disabled': True,
116        }
117        account = Account()
118        account.from_dict(account_data)
119        account.create()
120        account = account.delete()
121        self.assertIsNone(account)
122
```

Test Account Model

- Test creating multiple Accounts
- Test Account creation using known data
- Test if account can be deleted
- Test if account can be found
- Test if dictionary is set from given attributes
- Test the representation of an account
- Test account to dict
- Test if data can be updated
- Test if data can be updated without id

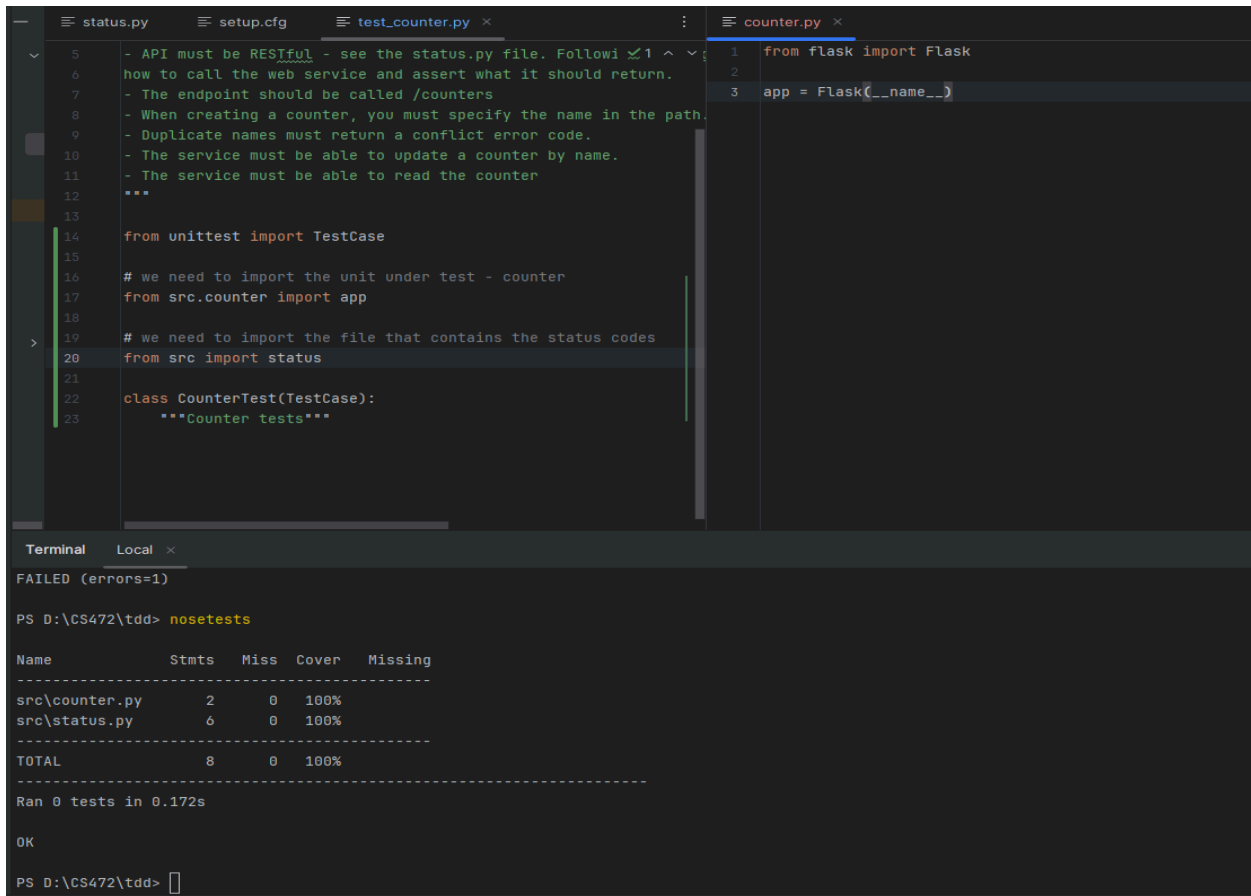
Name	Stmts	Miss	Cover	Missing
models\__init__.py	7	0	100%	
models\__init__.py	7	0	100%	
models\account.py	40	0	100%	
TOTAL	47	0	100%	

Ran 9 tests in 0.959s

OK

PS D:\CS472\test\_coverage>

## Task 5 - TDD



The screenshot shows a code editor with three open files: `status.py`, `test_counter.py`, and `counter.py`. The `test_counter.py` file contains a test class `CounterTest` that imports `Flask` and `FlaskTest` from `src`. The `counter.py` file contains a `Flask` application. The terminal at the bottom shows the output of the `nosetests` command, indicating that the tests passed successfully.

```
5 - API must be RESTful - see the status.py file. Follow the
6 how to call the web service and assert what it should return.
7 - The endpoint should be called /counters
8 - When creating a counter, you must specify the name in the path.
9 - Duplicate names must return a conflict error code.
10 - The service must be able to update a counter by name.
11 - The service must be able to read the counter
12 ***
13
14 from unittest import TestCase
15
16 # we need to import the unit under test - counter
17 from src.counter import app
18
19 # we need to import the file that contains the status codes
20 from src import status
21
22 class CounterTest(TestCase):
23     """Counter tests"""
```

```
1 from flask import Flask
2
3 app = Flask(__name__)
```

Terminal Local x

FAILED (errors=1)

PS D:\CS472\tdd> nosetests

Name	Stmts	Miss	Cover	Missing
src\counter.py	2	0	100%	
src\status.py	6	0	100%	
TOTAL	8	0	100%	

Ran 0 tests in 0.172s

OK

PS D:\CS472\tdd>

Add the first copy and paste parts and counter.py and showing no error.



```
status.py  setup.cfg  test_counter.py  counter.py

10  """The service must be able to update a counter by name.
11  - The service must be able to read the counter
12  """
13
14  from unittest import TestCase
15
16  # we need to import the unit under test - counter
17  from src.counter import app
18
19  # we need to import the file that contains the status codes
20  from src import status
21
22  class CounterTest(TestCase):
23      """Counter tests"""
24      def test_create_a_counter(self):
25          """It should create a counter"""
26          client = app.test_client()
27          result = client.post('/counters/foo')
28          self.assertEqual(result.status_code, status.HTTP_201_CREATED)

1  from flask import Flask
2
3  app = Flask(__name__)

Terminal  Local x + v
PS D:\CS472\tdd> nosetests

Counter tests
- It should create a counter (FAILED)

=====
FAIL: It should create a counter
-----
Traceback (most recent call last):
  File "D:\CS472\tdd\tests\test_counter.py", line 28, in test_create_a_counter
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
AssertionError: 404 != 201

Name          Stmts  Miss  Cover   Missing
-----
src\counter.py    2     0   100%
```

Adding the first test and getting an error message

```
status.py  setup.cfg  test_counter.py  counter.py

10 - The service must be able to update a counter by name.
11 - The service must be able to read the counter
12 """
13
14 from unittest import TestCase
15
16 # we need to import the unit under test - counter
17 from src.counter import app
18
19 # we need to import the file that contains the status codes
20 from src import status
21
22 class CounterTest(TestCase):
23     """Counter tests"""
24     def test_create_a_counter(self):
25         """It should create a counter"""
26         client = app.test_client()
27         result = client.post('/counters/foo')
28         self.assertEqual(result.status_code, status.HTTP_201_CREATED)

4 app = Flask(__name__)
5
6 COUNTERS = {}
7
8 # We will use the app decorator and create a route called slash count
9 # specify the variable in route <name>
10 # let Flask know that the only methods that is allowed to called
11 # on this function is "POST".
12
13 @app.route('/counters/<name>', methods=['POST'])
14 def create_counter(name):
15     """Create a counter"""
16     app.logger.info(f"Request to create counter: {name}")
17     global COUNTERS
18     if name in COUNTERS:
19         return {"Message": f"Counter {name} already exists"}, status.
20     COUNTERS[name] = 0
21     return {name: COUNTERS[name]}, status.HTTP_201_CREATED

Terminal  Local  x  +  v

Counter tests
- It should create a counter

Name          Stmts  Miss  Cover   Missing
-----
src\counter.py    11      1    91%    19
src\status.py      6      0   100%
-----
TOTAL             17      1    94%
-----
Ran 1 test in 0.189s

OK

PS D:\CS472\tdd>
d > src 2 counter.py
```

Adding the POST code and getting green

The image shows a code editor with two files open: `test_counter.py` and `counter.py`.

**test\_counter.py**

```
16 # we need to import the unit under test - counter
17 from src.counter import app
18
19 # we need to import the file that contains the status codes
20 from src import status
21
22 class CounterTest(TestCase):
23     """Counter tests"""
24     def setUp(self):
25         self.client = app.test_client()
26
27     def test_create_a_counter(self):
28         """It should create a counter"""
29         client = app.test_client()
30         result = client.post('/counters/foo')
31         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
32
33
```

**counter.py**

```
1 from flask import Flask
2 from src import status
3
4 app = Flask(__name__)
5
6 COUNTERS = {}
7
8 # We will use the app decorator and create a route called slash cou
9 # specify the variable in route <name>
10 # let Flask know that the only methods that is allowed to called
11 # on this function is "POST".
12
13 @app.route('/counters/<name>', methods=['POST'])
14 def create_counter(name):
15     """Create a counter"""
16     app.logger.info(f'Request to create counter: {name}')
17     global COUNTERS
18     COUNTERS[name] = 0
19     return {name: COUNTERS[name]}, status.HTTP_201_CREATED

```

**Terminal**

Counter tests  
- It should create a counter

Name	Stmts	Miss	Cover	Missing
src\counter.py	9	0	100%	
src\status.py	6	0	100%	
TOTAL	15	0	100%	

Ran 1 test in 0.189s  
OK  
PS D:\CS472\td>

Doing the refactor step, nothing changes.

```
16 # we need to import the unit under test - counter
17 from src.counter import app
18
19 # we need to import the file that contains the status codes
20 from src import status
21
22 class CounterTest(TestCase):
23     """Counter tests"""
24     def setUp(self):
25         self.client = app.test_client()
26
27     def test_create_a_counter(self):
28         """It should create a counter"""
29         client = app.test_client()
30         result = client.post('/counters/foo')
31         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
32
33     def test_duplicate_a_counter(self):
34         """It should return an error for duplicates"""
35         result = self.client.post('/counters/bar')
36         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
37         result = self.client.post('/counters/bar')
38         self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
```

```
1 from flask import flask
2 from src import status
3
4 app = Flask(__name__)
5
6 COUNTERS = {}
7
8 # We will use the app decorator and create a route called slash cou
9 # specify the variable in route <name>
10 # let Flask know that the only methods that is allowed to called
11 # on this function is "POST".
12
13 @app.route('/counters/<name>', methods=['POST'])
14 def create_counter(name):
15     """Create a counter"""
16     app.logger.info(f"Request to create counter: {name}")
17     global COUNTERS
18     COUNTERS[name] = 0
19     return {name: COUNTERS[name]}, status.HTTP_201_CREATED
```

Terminal Local x + v

```
self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
AssertionError: 201 != 409
----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: bar
src.counter: INFO: Request to create counter: bar
----- >> end captured logging << -----
```

Name	Stmts	Miss	Cover	Missing
src/counter.py	9	0	100%	
src/status.py	6	0	100%	
TOTAL	15	0	100%	

Ran 2 tests in 0.187s

Adding more snippets and I get the red phase error message.

```
16 # we need to import the unit under test - counter
17 from src.counter import app
18
19 # we need to import the file that contains the status codes
20 from src import status
21
22 class CounterTest(TestCase):
23     """Counter tests"""
24     def setUp(self):
25         self.client = app.test_client()
26
27     def test_create_a_counter(self):
28         """It should create a counter"""
29         client = app.test_client()
30         result = client.post('/counters/foo')
31         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
32
33     def test_duplicate_a_counter(self):
34         """It should return an error for duplicates"""
35         result = self.client.post('/counters/bar')
36         self.assertEqual(result.status_code, status.HTTP_201_CREATED)
37         result = self.client.post('/counters/bar')
38         self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
```

```
1 from flask import Flask
2 from src import status
3
4 app = Flask(__name__)
5
6 COUNTERS = {}
7
8 # We will use the app decorator and create a route called slash counter
9 # specify the variable in route <name>
10 # let Flask know that the only methods that is allowed to called
11 # on this function is "POST".
12
13 @app.route('/counters/<name>', methods=['POST'])
14 def create_counter(name):
15     """Create a counter"""
16     app.logger.info(f"Request to create counter: {name}")
17     global COUNTERS
18     if name in COUNTERS:
19         return {"Message": f"Counter {name} already exists"}, status.HTTP_409_CONFLICT
20     COUNTERS[name] = 0
21     return {name: COUNTERS[name]}, status.HTTP_201_CREATED
```

Counter tests

- It should create a counter
- It should return an error for duplicates

Name	Stmts	Miss	Cover	Missing
src\counter.py	11	0	100%	
src\status.py	6	0	100%	
TOTAL	17	0	100%	

Ran 2 tests in 0.185s

OK

PS D:\CS472\td>

Fixing the red phase and making it all good.

```
status.py  setup.cfg  test_counter.py  counter.py

28  """It should create a counter"""
29  client = app.test_client()
30  result = client.post('/counters/foo')
31  self.assertEqual(result.status_code, status.HTTP_201_CREATED)
32
33  def test_duplicate_a_counter(self):
34  """It should return an error for duplicates"""
35  result = self.client.post('/counters/bar')
36  self.assertEqual(result.status_code, status.HTTP_201_CREATED)
37  result = self.client.post('/counters/bar')
38  self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
39
40  def test_update_a_counter(self):
41  """Tests to update Counters"""
42  #Creates a counter
43  result = self.client.post('/counters/updateCounter')
44  self.assertEqual(result.status_code, status.HTTP_201_CREATED)
45  #Check that it returns successful and check the counter value
46  oldValue = result.json['updateCounter']
47  newValue = self.client.put('/counters/updateCounter')
48  self.assertEqual(newValue.status_code, status.HTTP_200_OK)
49  #If the oldValue is less than the newValue that means we updated suc
50  self.assertLess(oldValue, newValue.json['updateCounter'])

1  from flask import Flask
2  from src import status
3
4  app = Flask(__name__)
5
6  COUNTERS = {}
7
8  # We will use the app decorator and create a route called slash counters
9  # specify the variable in route <name>
10 # let Flask know that the only methods that is allowed to called
11 # on this function is "POST".
12
13 @app.route('/counters/<name>', methods=['POST'])
14 def create_counter(name):
15     """Create a counter"""
16     app.logger.info(f'Request to create counter: {name}')
17     global COUNTERS
18     if name in COUNTERS:
19         return {"Message": f'Counter {name} already exists'}, status.HTTP_
20     COUNTERS[name] = 0
21     return {name: COUNTERS[name]}, status.HTTP_201_CREATED

Terminal  Local x + v
Counter tests
- It should create a counter
- It should return an error for duplicates
- Tests to update Counters (FAILED)

=====
FAIL: Tests to update Counters
=====
Traceback (most recent call last):
  File "D:\CS472\tdd\tests\test_counter.py", line 48, in test_update_a_counter
    self.assertEqual(newValue.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200

----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: updateCounter
----- >> end captured logging << -----
```

Adding update counter test. We get an error for this case.

```
status.py  setup.cfg  test_counter.py  counter.py

28 """It should create a counter"""
29 client = app.test_client()
30 result = client.post('/counters/foo')
31 self.assertEqual(result.status_code, status.HTTP_201_CREATED)
32
33 def test_duplicate_a_counter(self):
34     """It should return an error for duplicates"""
35     result = self.client.post('/counters/bar')
36     self.assertEqual(result.status_code, status.HTTP_201_CREATED)
37     result = self.client.post('/counters/bar')
38     self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
39
40 def test_update_a_counter(self):
41     """Tests to update Counters"""
42     #Creates a counter
43     result = self.client.post('/counters/updateCounter')
44     self.assertEqual(result.status_code, status.HTTP_201_CREATED)
45     #Check that it returns successful and check the counter value
46     oldValue = result.json['updateCounter']
47     newValue = self.client.put('/counters/updateCounter')
48     self.assertEqual(newValue.status_code, status.HTTP_200_OK)
49     #if the oldValue is less than the newValue that means we updated suc
50     self.assertLess(oldValue, newValue.json['updateCounter'])

6 COUNTERS = {}
7
8 # We will use the app decorator and create a route called slash counters
9 # specify the variable in route <name>
10 # let Flask know that the only methods that is allowed to called
11 # on this function is 'POST'.
12
13 @app.route('/counters/<name>', methods=['POST'])
14 def create_counter(name):
15     """Create a counter"""
16     app.logger.info(f"Request to create counter: {name}")
17     global COUNTERS
18     if name in COUNTERS:
19         return {"Message": f"Counter {name} already exists"}, status.HTTP_
20     COUNTERS[name] = 0
21     return {name: COUNTERS[name]}, status.HTTP_201_CREATED
22
23 @app.route('/counters/<name>', methods=['PUT'])
24 def update_counter(name):
25     """Update a counter"""
26     app.logger.info(f"Request to update counter: {name}")
27     COUNTERS[name] = COUNTERS[name] + 1
28     return {name: COUNTERS[name]}, status.HTTP_200_OK

Terminal  Local x + v
- It should create a counter
- It should return an error for duplicates
- Tests to update Counters

Name          Stmts  Miss  Cover   Missing
-----
src\counter.py    16     0   100%
src\status.py      6     0   100%
-----
TOTAL              22     0   100%
Ran 3 tests in 0.191s
OK
```

Fixing that read error by adding to counter.py

```
status.py  setup.cfg  test_counter.py  counter.py

38 self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)
39
40 def test_update_a_counter(self):
41     """Tests to update Counters"""
42     #Creates a counter
43     result = self.client.post('/counters/updateCounter')
44     self.assertEqual(result.status_code, status.HTTP_201_CREATED)
45     #Check that it returns successful and check the counter value
46     oldValue = result.json['updateCounter']
47     newValue = self.client.put('/counters/updateCounter')
48     self.assertEqual(newValue.status_code, status.HTTP_200_OK)
49     #If the oldValue is less than the newValue that means we updated suc
50     self.assertLess(oldValue, newValue.json['updateCounter'])
51
52 def test_read_a_counter(self):
53     """Test to read a counter"""
54     #Creates a counter
55     result = self.client.post('/counters/readCounter')
56     self.assertEqual(result.status_code, status.HTTP_201_CREATED)
57     oldValue = result.json['readCounter']
58     newValue = self.client.get('/counters/readCounter')
59     self.assertEqual(newValue.status_code, status.HTTP_200_OK)
60     self.assertEqual(oldValue, newValue.json['readCounter'])

6 COUNTERS = {}
7
8 # We will use the app decorator and create a route called slash counters
9 # specify the variable in route <name>
10 # let Flask know that the only methods that is allowed to called
11 # on this function is "POST".
12
13 @app.route('/counters/<name>', methods=['POST'])
14 def create_counter(name):
15     """Create a counter"""
16     app.logger.info(f'Request to create counter: {name}')
17     global COUNTERS
18     if name in COUNTERS:
19         return {'Message': f'Counter {name} already exists'}, status.HTTP
20     COUNTERS[name] = 0
21     return {'name': COUNTERS[name]}, status.HTTP_201_CREATED
22
23 @app.route('/counters/<name>', methods=['PUT'])
24 def update_counter(name):
25     """Update a counter"""
26     app.logger.info(f'Request to update counter: {name}')
27     COUNTERS[name] = COUNTERS[name] + 1
28     return {'name': COUNTERS[name]}, status.HTTP_200_OK

Terminal  Local x + v
- It should create a counter
- It should return an error for duplicates
- Test to read a counter (FAILED)
- Tests to update Counters

=====
FAIL: Test to read a counter
=====
Traceback (most recent call last):
  File "D:\CS472\tdd\tests\test_counter.py", line 59, in test_read_a_counter
    self.assertEqual(newValue.status_code, status.HTTP_200_OK)
AssertionError: 405 != 200
----- >> begin captured logging << -----
src.counter: INFO: Request to create counter: readCounter
----- >> end captured logging << -----
```

Adding read a counter test case. This fails because there isn't something for GET in counter.py



The screenshot shows a code editor with two files: `test_counter.py` and `counter.py`. The `test_counter.py` file contains tests for creating, updating, and reading a counter. The `counter.py` file contains the Flask application logic for the counter.

```
test_counter.py
38 self.assertEqual(result.status_code, status.HTTP_409_CONFL
39
40 def test_update_a_counter(self):
41     """Tests to update Counters"""
42     #Creates a counter
43     result = self.client.post('/counters/updateCounter')
44     self.assertEqual(result.status_code, status.HTTP_201_CREATED)
45     #Check that it returns successful and check the counter value
46     oldValue = result.json['updateCounter']
47     newValue = self.client.put('/counters/updateCounter')
48     self.assertEqual(newValue.status_code, status.HTTP_200_OK)
49     #If the oldValue is less than the newValue that means we updated suc
50     self.assertLess(oldValue, newValue.json['updateCounter'])
51
52 def test_read_a_counter(self):
53     """Test to read a counter"""
54     #Creates a counter
55     result = self.client.post('/counters/readCounter')
56     self.assertEqual(result.status_code, status.HTTP_201_CREATED)
57     oldValue = result.json['readCounter']
58     newValue = self.client.get('/counters/readCounter')
59     self.assertEqual(newValue.status_code, status.HTTP_200_OK)
60     self.assertEqual(oldValue, newValue.json['readCounter'])

counter.py
12
13 @app.route('/counters/<name>', methods=['POST'])
14 def create_counter(name):
15     """Create a counter"""
16     app.logger.info(f"Request to create counter: {name}")
17     global COUNTERS
18     if name in COUNTERS:
19         return {"Message": f"Counter {name} already exists"}, status.HTTP
20     COUNTERS[name] = 0
21     return {name: COUNTERS[name]}, status.HTTP_201_CREATED
22
23 @app.route('/counters/<name>', methods=['PUT'])
24 def update_counter(name):
25     """Update a counter"""
26     app.logger.info(f"Request to update counter: {name}")
27     COUNTERS[name] = COUNTERS[name] + 1
28     return {name: COUNTERS[name]}, status.HTTP_200_OK
29
30 @app.route('/counters/<name>', methods=['GET'])
31 def read_counter(name):
32     """Read a counter"""
33     app.logger.info(f"Request to read counter: {name}")
34     return {name: COUNTERS[name]}, status.HTTP_200_OK
```

The terminal output shows the test results for the counter tests:

```
Counter tests
- It should create a counter
- It should return an error for duplicates
- Test to read a counter
- Tests to update Counters

Name           Stmts  Miss  Cover   Missing
-----
src\counter.py  20      0  100%
src\status.py   6      0  100%
TOTAL          26      0  100%

Ran 4 tests in 0.192s
```

Fixing the read error by adding GET into the counter.py

I am sorry for the terrible spacing of the pictures, I could not resize them to the point of having multiple on the same page without the test being too small to read.