Unit Testing Lab Report

In this lab, we learned how to create unit tests for both a Java and Python project. Additionally, we experimented with Test Driven Development (TDD) using Python's **`unittest`** library.

Task 1 – JPacman Test Coverage

The following is the initial code coverage for the JPacman project:

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	3.6% (2/55)	1.5% (5/327)	1.2% (14/1162)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
nl.tudelft.jpacman	0% (0/3)	0% (0/29)	0% (0/77)
nl.tudelft.jpacman.board	20% (2/10)	8.9% (5/56)	9.5% (14/148)
nl.tudelft.jpacman.fuzzer	0% (0/1)	0% (0/7)	0% (0/33)
nl.tudelft.jpacman.game	0% (0/3)	0% (0/14)	0% (0/37)
nl.tudelft.jpacman.integration	0% (0/1)	0% (0/5)	0% (0/7)
nl.tudelft.jpacman.level	0% (0/13)	0% (0/79)	0% (0/351)
nl.tudelft.jpacman.npc	0% (0/1)	0% (0/4)	0% (0/8)
nl.tudelft.jpacman.npc.ghost	0% (0/9)	0% (0/45)	0% (0/231)
nl.tudelft.jpacman.points	0% (0/2)	0% (0/9)	0% (0/21)
nl.tudelft.jpacman.sprite	0% (0/6)	0% (0/48)	0% (0/122)
nl.tudelft.jpacman.ui	0% (0/6)	0% (0/31)	0% (0/127)

generated on 2024-02-05 12:39

I acquired this report by right-clicking on the **`tests`** folder, then selecting the "Run 'Tests in 'jpacman.tests'' with Coverage" option. From there, in the "Coverage" pane, I clicked the "Generate Coverage Report…" button.

In my opinion, this coverage is not good enough. Out of all the classes, only 3.6% of them were covered, and for all the methods in the project, only 1.5% were covered. This is subpar when you consider the fact that industry expectations for code coverage can be upwards of 90%!

Task 2 – Increasing Coverage on JPacman

The following is my unit test for the **`isAlive()`** method:

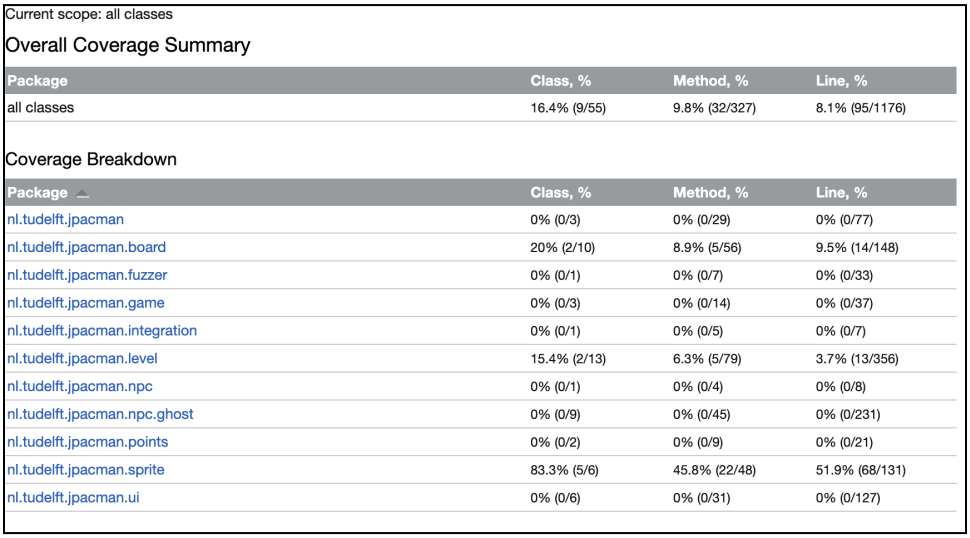
```
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.assertThat;

public class PlayerTest {
   private final PacManSprites SPRITES = new PacManSprites();
   private PlayerFactory Factory = new PlayerFactory(SPRITES);
   private Player pacMan = Factory.createPacMan();
```

```
    @Test
    void isAlive() {
        assertThat(pacMan.isAlive()).isEqualTo(true);
    }
}
```

After implementing this test, this was the new coverage report:



Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	16.4% (9/55)	9.8% (32/327)	8.1% (95/1176)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
nl.tudelft.jpacman	0% (0/3)	0% (0/29)	0% (0/77)
nl.tudelft.jpacman.board	20% (2/10)	8.9% (5/56)	9.5% (14/148)
nl.tudelft.jpacman.fuzzer	0% (0/1)	0% (0/7)	0% (0/33)
nl.tudelft.jpacman.game	0% (0/3)	0% (0/14)	0% (0/37)
nl.tudelft.jpacman.integration	0% (0/1)	0% (0/5)	0% (0/7)
nl.tudelft.jpacman.level	15.4% (2/13)	6.3% (5/79)	3.7% (13/356)
nl.tudelft.jpacman.npc	0% (0/1)	0% (0/4)	0% (0/8)
nl.tudelft.jpacman.npc.ghost	0% (0/9)	0% (0/45)	0% (0/231)
nl.tudelft.jpacman.points	0% (0/2)	0% (0/9)	0% (0/21)
nl.tudelft.jpacman.sprite	83.3% (5/6)	45.8% (22/48)	51.9% (68/131)
nl.tudelft.jpacman.ui	0% (0/6)	0% (0/31)	0% (0/127)

Task 2.1 - Creating My Own Unit Tests

For this section, I had to implement my own unit tests for the JPacman project. I decided to test the following methods:

- `src/main/java/nl/tudelft/jpacman/points/DefaultPointCalculator.consumedAPellet()`
- `src/main/java/nl/tudelft/jpacman/points/DefaultPointCalculator.collidedWithAGhost()`
- `src/main/java/nl/tudelft/jpacman/level/PlayerCollisions.playerVersusGhost()`

Task 2.1.1 - `consumedAPellet()` Test

Code

```
package nl.tudelft.jpacman.points;

import nl.tudelft.jpacman.level.LevelFactory;
import nl.tudelft.jpacman.level.Pellet;
import nl.tudelft.jpacman.level.Player;
import nl.tudelft.jpacman.level.PlayerFactory;
import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.sprite.PacManSprites;
```

```java
import static org.assertj.core.api.Assertions.assertThat;
import org.junit.jupiter.api.Test;

public class DefaultPointCalculatorTest {
   // Create PacMan (player)
   private final PacManSprites SPRITES = new PacManSprites();
   private final PlayerFactory PLAYER_FACTORY = new PlayerFactory(SPRITES);
   private final Player PACMAN = PLAYER_FACTORY.createPacMan();

   // Create a Pellet
   private final GhostFactory GHOST_FACTORY = new GhostFactory(SPRITES);
   private final DefaultPointCalculator CALCULATOR = new
DefaultPointCalculator();
   private final LevelFactory LEVEL_FACTORY = new LevelFactory(SPRITES,
GHOST_FACTORY, CALCULATOR);
   private final Pellet PELLET = LEVEL_FACTORY.createPellet();

   @Test
   void consumedAPellet() {
       CALCULATOR.consumedAPellet(PACMAN, PELLET);
       assertThat(PACMAN.getScore()).isEqualTo(PELLET.getValue());
   }
}
```

Coverage Report

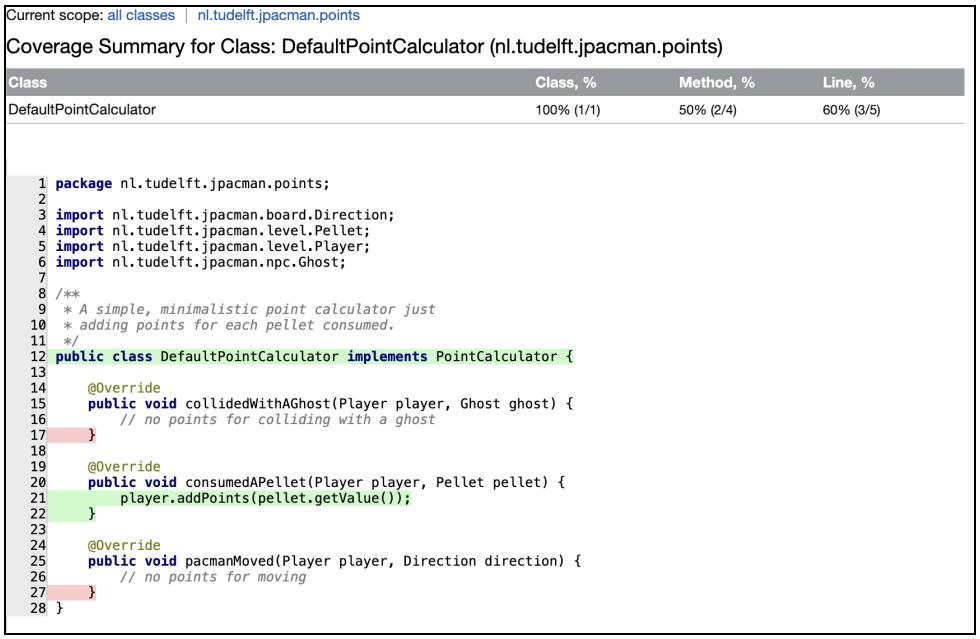
Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	23.6% (13/55)	12.8% (42/327)	9.9% (117/1186)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
nl.tudelft.jpacman	0% (0/3)	0% (0/29)	0% (0/77)
nl.tudelft.jpacman.board	20% (2/10)	8.9% (5/56)	9.5% (14/148)
nl.tudelft.jpacman.fuzzer	0% (0/1)	0% (0/7)	0% (0/33)
nl.tudelft.jpacman.game	0% (0/3)	0% (0/14)	0% (0/37)
nl.tudelft.jpacman.integration	0% (0/1)	0% (0/5)	0% (0/7)
nl.tudelft.jpacman.level	30.8% (4/13)	13.9% (11/79)	7.8% (28/359)
nl.tudelft.jpacman.npc	0% (0/1)	0% (0/4)	0% (0/8)
nl.tudelft.jpacman.npc.ghost	11.1% (1/9)	2.2% (1/45)	1.3% (3/237)
nl.tudelft.jpacman.points	50% (1/2)	22.2% (2/9)	13.6% (3/22)
nl.tudelft.jpacman.sprite	83.3% (5/6)	47.9% (23/48)	52.7% (69/131)
nl.tudelft.jpacman.ui	0% (0/6)	0% (0/31)	0% (0/127)



Current scope: all classes | nl.tudelft.jpacman.points

Coverage Summary for Class: DefaultPointCalculator (nl.tudelft.jpacman.points)

Class	Class, %	Method, %	Line, %
DefaultPointCalculator	100% (1/1)	50% (2/4)	60% (3/5)

```
 1 package nl.tudelft.jpacman.points;
 2
 3 import nl.tudelft.jpacman.board.Direction;
 4 import nl.tudelft.jpacman.level.Pellet;
 5 import nl.tudelft.jpacman.level.Player;
 6 import nl.tudelft.jpacman.npc.Ghost;
 7
 8 /**
 9  * A simple, minimalistic point calculator just
10  * adding points for each pellet consumed.
11  */
12 public class DefaultPointCalculator implements PointCalculator {
13
14     @Override
15     public void collidedWithAGhost(Player player, Ghost ghost) {
16         // no points for colliding with a ghost
17     }
18
19     @Override
20     public void consumedAPellet(Player player, Pellet pellet) {
21         player.addPoints(pellet.getValue());
22     }
23
24     @Override
25     public void pacmanMoved(Player player, Direction direction) {
26         // no points for moving
27     }
28 }
```

Task 2.1.2 - `collidedWithAGhost()` Test

Code

This test includes the same private members as the first test, with the addition of the **`clyde GhostFactory`**.

```
// Create a Ghost (Clyde!)
private final GhostFactory GHOST_FACTORY = new GhostFactory(SPRITES);
private Ghost clyde = GHOST_FACTORY.createClyde();

@Test
void collidedWithAGhost() {
   CALCULATOR.collidedWithAGhost(PACMAN, clyde);
   assertThat(PACMAN.getScore()).isEqualTo(0);
}
```

Coverage Report

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	29.1% (16/55)	15% (49/327)	11.9% (141/1186)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
nl.tudelft.jpacman	0% (0/3)	0% (0/29)	0% (0/77)
nl.tudelft.jpacman.board	20% (2/10)	8.9% (5/56)	9.5% (14/148)
nl.tudelft.jpacman.fuzzer	0% (0/1)	0% (0/7)	0% (0/33)
nl.tudelft.jpacman.game	0% (0/3)	0% (0/14)	0% (0/37)
nl.tudelft.jpacman.integration	0% (0/1)	0% (0/5)	0% (0/7)
nl.tudelft.jpacman.level	30.8% (4/13)	13.9% (11/79)	7.8% (28/359)
nl.tudelft.jpacman.npc	100% (1/1)	25% (1/4)	62.5% (5/8)
nl.tudelft.jpacman.npc.ghost	33.3% (3/9)	11.1% (5/45)	7.6% (18/237)
nl.tudelft.jpacman.points	50% (1/2)	33.3% (3/9)	18.2% (4/22)
nl.tudelft.jpacman.sprite	83.3% (5/6)	50% (24/48)	55% (72/131)
nl.tudelft.jpacman.ui	0% (0/6)	0% (0/31)	0% (0/127)

Current scope: all classes | nl.tudelft.jpacman.points

Coverage Summary for Class: DefaultPointCalculator (nl.tudelft.jpacman.points)

Class	Class, %	Method, %	Line, %
DefaultPointCalculator	100% (1/1)	75% (3/4)	80% (4/5)

```
 1 package nl.tudelft.jpacman.points;
 2
 3 import nl.tudelft.jpacman.board.Direction;
 4 import nl.tudelft.jpacman.level.Pellet;
 5 import nl.tudelft.jpacman.level.Player;
 6 import nl.tudelft.jpacman.npc.Ghost;
 7
 8 /**
 9  * A simple, minimalistic point calculator just
10  * adding points for each pellet consumed.
11  */
12 public class DefaultPointCalculator implements PointCalculator {
13
14     @Override
15     public void collidedWithAGhost(Player player, Ghost ghost) {
16         // no points for colliding with a ghost
17     }
18
19     @Override
20     public void consumedAPellet(Player player, Pellet pellet) {
21         player.addPoints(pellet.getValue());
22     }
23
24     @Override
25     public void pacmanMoved(Player player, Direction direction) {
26         // no points for moving
27     }
28 }
```

Task 2.1.3 - `playerVersusGhost()` Test

Code

```java
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.npc.Ghost;
import nl.tudelft.jpacman.npc.ghost.GhostFactory;
import nl.tudelft.jpacman.points.DefaultPointCalculator;
import nl.tudelft.jpacman.sprite.PacManSprites;
import static org.assertj.core.api.Assertions.assertThat;
import org.junit.jupiter.api.Test;

public class PlayerCollisionsTest {
```

```
    // Create PacMan (player)
    private final PacManSprites SPRITES = new PacManSprites();
    private final PlayerFactory PLAYER_FACTORY = new PlayerFactory(SPRITES);
    private final Player PACMAN = PLAYER_FACTORY.createPacMan();

    // Create a Ghost (Clyde!)
    private final GhostFactory GHOST_FACTORY = new GhostFactory(SPRITES);
    private Ghost clyde = GHOST_FACTORY.createClyde();

    // Create collisions tracker
    private DefaultPointCalculator calculator = new DefaultPointCalculator();
    private PlayerCollisions collisions = new PlayerCollisions(calculator);

    @Test
    void playerVersusGhost() {
        collisions.playerVersusGhost(PACMAN, clyde);
        assertThat(PACMAN.isAlive()).isFalse();
        assertThat(PACMAN.getKiller()).isEqualTo(clyde);
    }
}
```

Coverage Report

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	30.9% (17/55)	16.8% (55/327)	13.4% (160/1193)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
nl.tudelft.jpacman	0% (0/3)	0% (0/29)	0% (0/77)
nl.tudelft.jpacman.board	20% (2/10)	8.9% (5/56)	9.5% (14/148)
nl.tudelft.jpacman.fuzzer	0% (0/1)	0% (0/7)	0% (0/33)
nl.tudelft.jpacman.game	0% (0/3)	0% (0/14)	0% (0/37)
nl.tudelft.jpacman.integration	0% (0/1)	0% (0/5)	0% (0/7)
nl.tudelft.jpacman.level	38.5% (5/13)	20.3% (16/79)	11.7% (43/366)
nl.tudelft.jpacman.npc	100% (1/1)	25% (1/4)	62.5% (5/8)
nl.tudelft.jpacman.npc.ghost	33.3% (3/9)	11.1% (5/45)	7.6% (18/237)
nl.tudelft.jpacman.points	50% (1/2)	33.3% (3/9)	18.2% (4/22)
nl.tudelft.jpacman.sprite	83.3% (5/6)	52.1% (25/48)	58% (76/131)
nl.tudelft.jpacman.ui	0% (0/6)	0% (0/31)	0% (0/127)

```
    /**
     * Actual case of player bumping into ghost or vice versa.
     *
     * @param player
     *          The player involved in the collision.
     * @param ghost
     *          The ghost involved in the collision.
     */
    public void playerVersusGhost(Player player, Ghost ghost) {
        pointCalculator.collidedWithAGhost(player, ghost);
        player.setAlive(false);
        player.setKiller(ghost);
    }
```

Task 3 – JaCoCo Report on JPacman

The following coverage report was generated with the JaCoCo tool.

jpacman

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.jpacman.level		67%		57%	73	155	103	344	20	69	4	12
nl.tudelft.jpacman.npc.ghost		71%		55%	56	105	43	181	5	34	0	8
nl.tudelft.jpacman.ui		78%		47%	54	86	21	144	7	31	0	6
default		0%		0%	12	12	21	21	5	5	1	1
nl.tudelft.jpacman.sprite		86%		59%	30	70	11	113	5	38	0	5
nl.tudelft.jpacman.board		86%		58%	44	93	2	110	0	40	0	7
nl.tudelft.jpacman		67%		25%	12	30	18	52	6	24	1	2
nl.tudelft.jpacman.points		59%		75%	1	11	5	21	0	9	0	2
nl.tudelft.jpacman.game		87%		60%	10	24	4	45	2	14	0	3
nl.tudelft.jpacman.npc		100%		n/a	0	4	0	8	0	4	0	1
Total	1,242 of 4,755	73%	293 of 637	54%	292	590	228	1,039	50	268	6	47

The coverage results generated from JaCoCo are similar to the ones generated by IntelliJ, as they report essentially the same information: the percentage of code that's covered by tests. However, JaCoCo's report is far more granular because it also includes information on branch coverage and total overall coverage for the entire project.

I found the source code visualization from JaCoCo on uncovered branches to be extremely helpful when compared to IntelliJ's. The JaCoCo report will show if statements as yellow and uncovered branches in red, making them easy to pinpoint in a large codebase. In contrast, IntelliJ's source code visualization only marks entire methods in red if they're uncovered with no discrepancy for branch conditions.

With these factors in mind, I prefer JaCoCo's report over IntelliJ's. Being able to view branch coverage both within the overall report and in the source code visualization allows me to make more fine-tuned unit tests that are guaranteed to be robust. Additionally, I think JaCoCo's reporting method makes more sense. I'd rather see progress indicators and totals than scattered percentages that don't immediately make sense.

Task 4 – Working with Python Test Coverage

For this task, I had to implement tests for a Python project that creates, reads, updates, and deletes accounts from a database. Before I started, this was the test coverage report:

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data

Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
models/__init__.py      7      0   100%
models/account.py      40     13    68%   26, 30, 34-35, 45-48, 52-54, 74-75
-------------------------------------------------
TOTAL                  47     13    72%
----------------------------------------------------------------------
Ran 2 tests in 0.878s

OK
```

Then, after implementing the two tests provided in the lab instructions (**`test_repr()`** and **`test_to_dict()`**), I received this coverage report:

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test the representation of an account
- Test account to dict

Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
models/__init__.py      7      0   100%
models/account.py      40     11    72%   34-35, 45-48, 52-54, 74-75
-------------------------------------------------
TOTAL                  47     11    77%
----------------------------------------------------------------------
Ran 4 tests in 0.274s

OK
```

The following are all the tests that I implemented that brought me to 100% total coverage according to **`nosetests`**.

Test 1 - ***`from_dict()`***

```python
def test_from_dict(self):
    """ Test dict to account """
    test_data = {
        "name": "Jordan",
        "email": "kingj32@unlv.nevada.edu",
        "phone_number": "702.123.4567",
        "disabled": False,
    }
    account = Account()
    account.from_dict(test_data)
    self.assertEqual(account.name, test_data["name"])
```

```
    self.assertEqual(account.email, test_data["email"])
    self.assertEqual(account.phone_number, test_data["phone_number"])
    self.assertEqual(account.disabled, test_data["disabled"])
```

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test dict to account
- Test the representation of an account
- Test account to dict

Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models/__init__.py       7      0   100%
models/account.py       40      9    78%   45-48, 52-54, 74-75
--------------------------------------------------
TOTAL                   47      9    81%
----------------------------------------------------------------------
Ran 5 tests in 0.263s

OK
```

*Test 2 - **`update()`** - #1*

The tests for the **`update()`** method needed to be broken up into two different tests because there was a branching condition that raises an exception if the account doesn't have an **`id`** (i.e., doesn't exist in the database).

These could not be tested within the same test method because it was easier to mock an account that's in the database and another that's not in separate methods. Additionally, this allows **`nosetests`** to generate a more accurate coverage report.

```
def test_update_with_id(self):
    """ Test updating the database """

    # Create a new account from JSON and add it to the db.
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()

    # Change the name and update the db.
    account.name = "Businge"
    account.update()

    # Fetch the user from the db and assert the name changed.
    result = Account.find(account.id)
    self.assertEqual(result.name, account.name)
```

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test dict to account
- Test the representation of an account
- Test account to dict
- Test updating the database

Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
models/__init__.py      7      0   100%
models/account.py      40      6    85%   47, 52-54, 74-75
-------------------------------------------------
TOTAL                  47      6    87%
----------------------------------------------------------------------
Ran 6 tests in 0.283s

OK
```

Test 3 - ***`update()`*** *- #2*

```python
def test_update_without_id(self):
    """ Test updating the database when the account DOESN'T have an id """

    # Create a new account from JSON but don't add it to the db.
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)

    # Change the name and assert that an exception will be raised.
    account.name = "Businge"
    self.assertRaises(DataValidationError, account.update)
```

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test dict to account
- Test the representation of an account
- Test account to dict
- Test updating the database
- Test updating the database when the account DOESN'T have an id

Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
models/__init__.py      7      0   100%
models/account.py      40      5    88%   52-54, 74-75
-------------------------------------------------
TOTAL                  47      5    89%
----------------------------------------------------------------------
Ran 7 tests in 0.294s

OK
```

Test 4 - ***`delete()`***

```python
def test_delete(self):
    """ Test deleting an account """

    # Create a new account from JSON and add it to the db.
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()

    # Delete the account and assert it no longer exists.
    account.delete()
    result = Account.find(account.id)
    self.assertIsNone(result)
```

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test deleting an account
- Test dict to account
- Test the representation of an account
- Test account to dict
- Test updating the database
- Test updating the database when the account DOESN'T have an id

Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
models/__init__.py      7      0   100%
models/account.py      40      2    95%   74-75
-------------------------------------------------
TOTAL                  47      2    96%
----------------------------------------------------------------------
Ran 8 tests in 0.321s

OK
```

Test 5 - ***`find()`***

```python
def test_find(self):
    """ Test finding an account in the db """

    # Create a new account from JSON and add it to the db.
    data = ACCOUNT_DATA[self.rand]
    account = Account(**data)
    account.create()

    # Find the account we just created and assert it exists within the db.
    result = Account.find(account.id)
    self.assertIsNotNone(result)
```

```
Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test deleting an account
- Test finding an account in the db
- Test dict to account
- Test the representation of an account
- Test account to dict
- Test updating the database
- Test updating the database when the account DOESN'T have an id

Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
models/__init__.py      7      0   100%
models/account.py      40      0   100%
-------------------------------------------------
TOTAL                  47      0   100%
----------------------------------------------------------------------
Ran 9 tests in 0.314s

OK
```

Task 5 - TDD

For this section, we were tasked with first creating tests for features (methods) we wanted to implement (red phase), then implementing those features such that the tests we wrote passed (green phase). If there was any room for improvement, we refactored the code while ensuring the tests were still passing.

The following are the unit tests I implemented in **`tests/test_counter.py`**:

```python
def test_update_a_counter(self):
    """ It should update the counter by 1 when a PUT request is made. """

    # Post a new counter called 'jordan'
    name = 'jordan'
    result = self.client.post(f'/counters/{name}')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    counter = result.get_json()[name]

    # Update the counter via PUT
    result = self.client.put(f'/counters/{name}')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    new_counter = result.get_json()[name]

    self.assertEqual(new_counter, counter + 1)

def test_get_a_counter(self):
    """ It should GET a counter """

    # Post a new counter called 'jordan'
    name = 'businge'
```

```
    result = self.client.post(f'/counters/{name}')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)
    counter = result.get_json()

    # Retrieve the counter via GET
    result = self.client.get(f'/counters/{name}')
    self.assertEqual(result.status_code, status.HTTP_200_OK)
    retrieved_counter = result.get_json()

    self.assertEqual(counter, retrieved_counter)
```

And here are the respective methods I implemented in **`src/counter.py`**:

```
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """ Update a counter """
    app.logger.info(f"Request to update counter: {name}")
    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK


@app.route('/counters/<name>', methods=['GET'])
def get_counter(name):
    """ Get a counter """
    app.logger.info(f"Request to get counter: {name}")
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

Implementing these tests and methods helped me get a 100% test coverage report.

```
Counter tests
- It should create a counter
- It should return an error for duplicates
- It should GET a counter
- It should update the counter by 1 when a PUT request is made.

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 0.104s

OK
```

When it comes to the TDD phases, I only encountered the red and green phases. The requirements of both tests were straightforward enough that I didn't need to refactor anything once it came time to implement the actual code.

For both tests, they initially failed reporting a **`HTTP_409_CONFLICT`** error. This makes sense considering no other code was implemented to account for other status codes. Once I implemented the code pertaining to the tests (green phase), only the **`HTTP_200_OK`** status was being returned and all my tests were turning green, as shown above.

Jordan King

GitHub Fork: https://github.com/KingJordan152/CS472_Group1