MacGyver Codilla
CS472
Task 3

Are the coverage results from JaCoCo similar to the ones you got from IntelliJ in the last task? Why so or why not?

They are similar. The lines of code I've written tests for are highlighted in green. Any code that I did not write tests for are yellow.

Did you find helpful the source code visualization from JaCoCo on uncovered branches?

I have found it to be helpful. It would have been nice to know this existed before this task.

Which visualization did you prefer and why? IntelliJ's coverage window or JaCoCo's report?

IntelliJ's coverage window only showed an approximation of the coverage as a percentage. JaCoCo showed which code was covered and which were missed. A visualization as well as a pointer as to where the coverage was missed would have been a lot more helpful than an approximation.

Code snippets:

UnitTest.java | GhostFactoryTest.java | Pellet.java | Player.java | GhostFac

```java

import nl.tudelft.jpacman.npc.Ghost;
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

new *
public class GhostFactoryTest
{
    2 usages
    private static final PacManSprites SPRITE_STORE = new PacManSprites();

    new *
    @Test
    public void createBlinkyTest()
    {
        GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);
        Ghost blinky = ghostFactory.createBlinky();

        assertTrue(blinky instanceof Blinky);
    }

    new *
    @Test
    public void createPinkyTest()
    {
        GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);
        Ghost pinky = ghostFactory.createPinky();

        assertTrue(pinky instanceof Pinky);
    }
}
```

UnitTest.java × | GhostFactoryTest.java | Pellet.java | Player.java | GhostFactory.java

```java
import nl.tudelft.jpacman.sprite.PacManSprites;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;

new *
public class UnitTest
{
    1 usage
    private static final PacManSprites SPRITE_STORE = new PacManSprites();

    new *
    @Test
    public void testRandomMove()
    {
        GhostFactory ghostFactory = new GhostFactory(SPRITE_STORE);
        Ghost inky = ghostFactory.createInky();

        Direction direction = inky.getDirection();

        if (direction == Direction.EAST)
        {
            assertEquals(direction, Direction.EAST);
        }
        else if (direction == Direction.NORTH)
        {
            assertEquals(direction, Direction.NORTH);
        }
        else if (direction == Direction.SOUTH)
        {
            assertEquals(direction, Direction.SOUTH);
        }
        else if (direction == Direction.WEST)
        {
            assertEquals(direction, Direction.WEST);
        }
        else
        {
            assertNull(direction);
        }
    }
}
```

Task 4

In task 4, I was tasked with improving the test coverage in this repository. Fortunately, the two methods to be tested are given which gave me enough working knowledge to work through the rest.

The basic idea is to request a random account and perform an operation on it using the class methods. If it calls for it, the database will be updated to reflect those changes.

```
# GIVEN 26
def test_repr(self):
    """Test the representation of an account"""
    account = Account()
    account.name = "Foo"
    self.assertEqual(str(account), "<Account 'Foo'>")

# GIVEN 30
def test_to_dict(self):
    """ Test account to dict """
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    result = account.to_dict()
    self.assertEqual(account.name, result["name"])
    self.assertEqual(account.email, result["email"])
    self.assertEqual(account.phone_number, result["phone_number"])
    self.assertEqual(account.disabled, result["disabled"])
    self.assertEqual(account.date_joined, result["date_joined"])
```

For my own test cases, I had requested a random account and performed an operation on that account. Some methods requested that an id is to be assigned, however, I could not find what that corresponded with. I was able to get a decent enough coverage at 92%. The only hiccups came from committing to the database. None of my group members could replicate the issue. I was running python 3.10.10…

README.md | account.py | test_account.py

Plugins supporting *.py files found. Install Python plugin Ignore extension

```python
    ######################################################################
    #  T E S T   C A S E S
    ######################################################################

    # 74-75
    def test_find(self):
        """ Test account find """
        idx = self.rand
        data = ACCOUNT_DATA[idx] # get a random account
        account = Account(**data)

        result = account.Find(idx)
        if result == None:
            self.assertIsNone(result)
        else:
            d = account.to_dict()
            self.assertEqual(account.name, d["name"])

    # 52-54
    def test_delete(self):
        """ Test account delete """
        idx = self.rand
        data = ACCOUNT_DATA[idx] # get a random account
        account = Account(**data)
        account.delete()
        account2 = ACCOUNT_DATA[idx]
        assertNotEqual(account.name, account2.name)

    # 45-48
    def test_update(self):
        """ Test account update """
        idx = self.rand
        data = ACCOUNT_DATA[idx] # get a random account
        account = Account(**data)
        result = account.to_dict()
        self.assertEqual(account.name, result["name"])
        account.name = "CAROL BURNETT"
        account.id = idx
        account.update()

        idx = self.rand
        data = ACCOUNT_DATA[idx] # get a random account
        account = Account(**data)
        with self.assertRaises(DataValidationError) as context:
            account.update()


    # 34-35
    def test_from_dict(self):
        """ Test account from dict """
        data = ACCOUNT_DATA[self.rand] # get a random account
        account = Account(**data)
        result = account.to_dict()
        result["name"] = "PAUL HARRELL"
        account.from_dict(result)
        self.assertEqual(account.name, result["name"])
        self.assertEqual(account.email, result["email"])
        self.assertEqual(account.phone_number, result["phone_number"])
        self.assertEqual(account.disabled, result["disabled"])
        self.assertEqual(account.date_joined, result["date_joined"])

    # GIVEN 26
```

Task 5

This utilized a different development methodology called Test Driven Development. The task required a different frame of mind from multiple perspectives to approach the problems. Ideally someone would design the program, and a tester would take that design and write tests for it. The software developer codes the design to spec.

I've written the update method in counter.py to get the counter for the webpage. First, I fetched the counter from the list of available counters. If it did not exist, I returned an error message and status 404. I incremented the counter by 1, and returned the counter and a 200 status code. In the test, I made a call to create the counter, ensured it was successful and got the value, called the update, checked the response's status code, and then asserted that the counter was one more than the previous.

This task exposed me to different perspectives, and got me familiar with the thought process behind this methodology.

Link: https://github.com/39otsu/cs472-group4

README.md | status.py | setup.cfg | test_counter.py | counter.py

Plugins supporting *.py files found. Install Python plugin Ignore extension

```python
"""
Test Cases for Counter Web Service

Create a service that can keep a track of multiple counters
- API must be RESTful - see the status.py file. Following these guidelines, you can make ass
how to call the web service and assert what it should return.
- The endpoint should be called /counters
- When creating a counter, you must specify the name in the path.
- Duplicate names must return a conflict error code.
- The service must be able to update a counter by name.
- The service must be able to read the counter
"""
from unittest import TestCase

# we need to import the unit under test - counter
from src.counter import app

# we need to import the file that contains the status codes
from src import status

class CounterTest(TestCase):
    """Counter tests"""
    def setUp(self):
        self.client = app.test_client()

    def test_create_a_counter(self):
        """It should create a counter"""
        result = self.client.post('/counters/foo')
        self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    def test_duplicate_a_counter(self):
        """It should return an error for duplicates"""
        result = self.client.post('/counters/bar')
        self.assertEqual(result.status_code, status.HTTP_201_CREATED)
        result = self.client.post('/counters/bar')
        self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)

    def test_update_a_counter(self):
        """It should update the counter"""
        result = self.client.post('/counters/cherry')
        self.assertEqual(result.status_code, status.HTTP_201_CREATED)
        firstTimeCounter = result.json['cherry']
        result = self.client.put('/counters/cherry')
        secondTimeCounter = result.json['cherry']
        self.assertEqual(result.status_code, status.HTTP_200_OK)
        self.assertEqual(secondTimeCounter, firstTimeCounter + 1)
        result = self.client.put('/counters/cherries')
        self.assertEqual(result.status_code, status.HTTP_404_NOT_FOUND)
```

M↓ README.md × status.py setup.cfg test_counter.py counter.py ×

Plugins supporting *.py files found. Install Python plugin Ignore extension

```python
from flask import Flask
import status
app = Flask(__name__)

COUNTERS = {}

# We will use the app decorator and create a route called slash counters.
# specify the variable in route <name>
# let Flask know that the only methods that is allowed to called
# on this function is "POST".
@app.route('/counters/<name>', methods=['POST'])
def create_counter(name):
    """Create a counter"""
    app.logger.info(f"Request to create counter: {name}")
    global COUNTERS
    if name in COUNTERS:
        return {"Message":f"Counter {name} already exists"}, status.HTTP_409_CONFLICT
    COUNTERS[name] = 0
    return {name: COUNTERS[name]}, status.HTTP_201_CREATED

@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """update a counter"""
    app.logger.info(f"Request to update counter: {name}")
    global COUNTERS
    if name not in COUNTERS:
        return {"Message":f"Counter {name} does not exist."}, status.HTTP_404_NOT_FOUND
    COUNTERS[name] = COUNTERS[name] + 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```