Dylan Cozloff

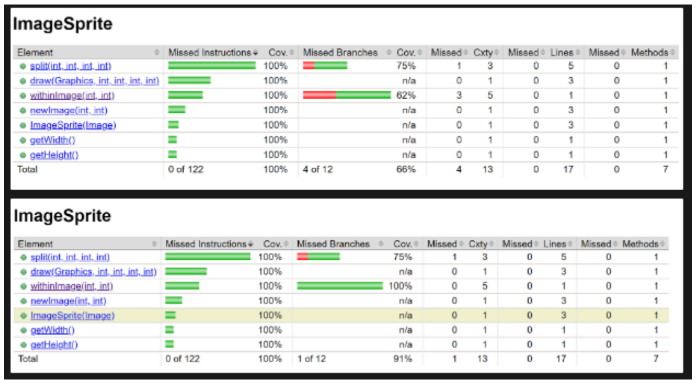
# ***Unit Testing IntelliJ vs JaCoCo***

***https://github.com/cozloff/2DRogueLikeUnityGame***

In this report, I overview my unit test for three methods in ImageSprite: getWidth(), getHeight(), withinImage(). I used BufferedImage to generate an image, and that image to generate a sprite (`new BufferedImage(10, 10, BufferedImage.TYPE_INT_ARGB`).

```
void testGetWidth(){ assertThat(testImage.getWidth()).isEqualTo(10); }

void testGetHeight(){ assertThat(testImage.getHeight()).isEqualTo(10); }

void testWithinImage(){
       assertThat(sprite.withinImage(-1,0)).isFalse(); // x < 0
       assertThat(sprite.withinImage(0,-1)).isFalse(); // y < 0
       assertThat(sprite.withinImage(10,0)).isFalse(); //edge cases
       assertThat(sprite.withinImage(0,10)).isFalse(); //edge cases
       assertThat(sprite.withinImage(0,0)).isTrue(); // least inside
       assertThat(sprite.withinImage(9,9)).isTrue(); // greatest inside
}
```

In this overview I compare the methods of unit test analysis JaCoCo and IntelliJ. Although IntelliJ is integrated into a desktop IDE and JaCoCo is an HTML application, they are very similar in terms of information provided. Although JaCoCo differs in its ability to indicate missed branches in unit test coverage. This was paticularly useful because it helped me fix the withinImage() test to have full coverage. If you look at the code above, the edge cases used to have the value 11 instead of 10, this caused 62% missed branches and I fixed it because of JaCoCo.



**ImageSprite**

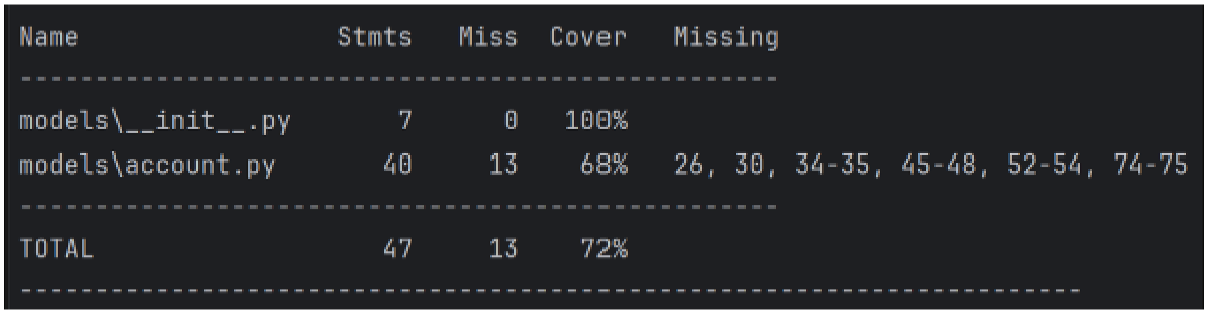
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| split(int, int, int, int) | 🟩 | 100% | 🟥🟩 | 75% | 1 | 3 | 0 | 5 | 0 | 1 |
| draw(Graphics, int, int, int, int) | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| withinImage(int, int) | 🟩 | 100% | 🟥🟩 | 62% | 3 | 5 | 0 | 1 | 0 | 1 |
| newImage(int, int) | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| ImageSprite(Image) | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| getWidth() | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| getHeight() | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 0 of 122 | 100% | 4 of 12 | 66% | 4 | 13 | 0 | 17 | 0 | 7 |

**ImageSprite**

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| split(int, int, int, int) | 🟩 | 100% | 🟥🟩 | 75% | 1 | 3 | 0 | 5 | 0 | 1 |
| draw(Graphics, int, int, int, int) | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| withinImage(int, int) | 🟩 | 100% | 🟩 | 100% | 0 | 5 | 0 | 1 | 0 | 1 |
| newImage(int, int) | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| ImageSprite(Image) | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| getWidth() | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| getHeight() | 🟩 | 100% |  | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 0 of 122 | 100% | 1 of 12 | 91% | 1 | 13 | 0 | 17 | 0 | 7 |

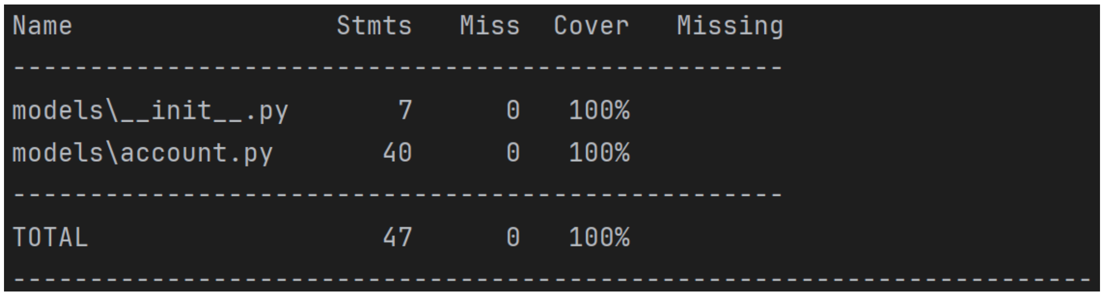
As you can see above my three tests have full coverage. I prefered using JaCoCo because it showed missed branches and was generally more concise in its reporting.

## Account Tests:



```
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%
-----------------------------------------------------------------------
```

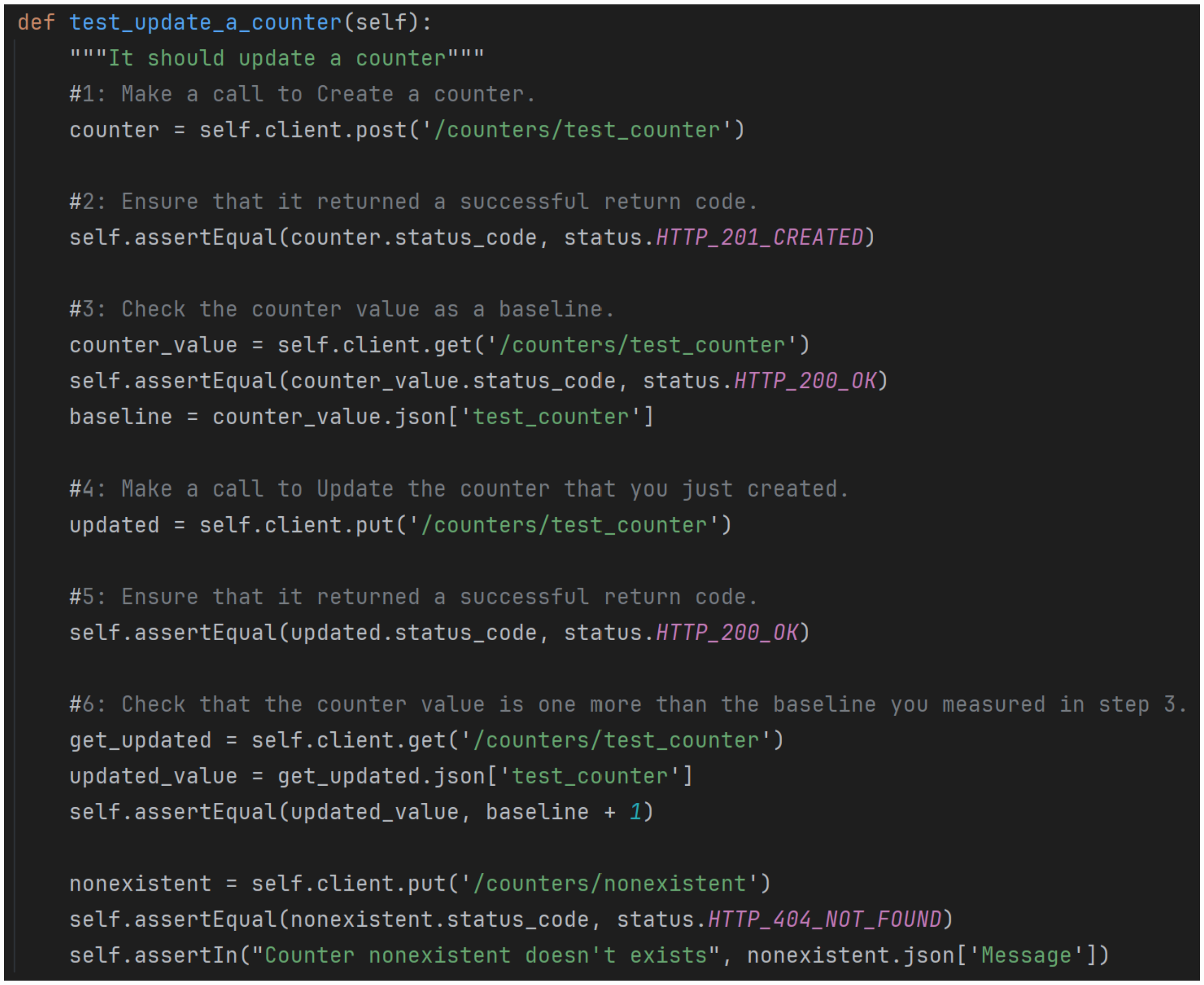
```python
def test_repr(self):
    """26: Test the representation of an account"""
    account = Account()
    account.name = "Foo"
    self.assertEqual(str(account), "<Account 'Foo'>")

def test_to_dict(self):
    """30: 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"])

def test_from_dict(self):
    """34-35: Test setting attributes from a dictionary"""
    account = Account()
    dict_data = {
        'name': 'test_name',
        'email': 'test_email',
        'phone_number': 'test_phone_number',
        'disabled': True,
        'date_joined': 'test_date_joined'
    }
    account.from_dict(dict_data)
    self.assertEqual(account.name, 'test_name')
    self.assertEqual(account.email, 'test_email')
    self.assertEqual(account.phone_number, 'test_phone_number')
    self.assertEqual(account.disabled, True)
    self.assertEqual(account.date_joined, 'test_date_joined')

def test_update(self):
    """45-48: Test updating an Account in the database"""
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()

    dict_data = {
        'name': 'test_name',
        'email': 'test_email',
        'phone_number': 'test_phone_number',
        'disabled': True,
        'date_joined': datetime.datetime.now()
    }

    account.from_dict(dict_data)
    self.assertIsNotNone(account.id)
    account.update()

    updated_account = Account.find(account.id)
    self.assertEqual(account.name, 'test_name')
    self.assertEqual(account.email, 'test_email')
    self.assertEqual(account.phone_number, 'test_phone_number')
    self.assertEqual(account.disabled, True)
    self.assertTrue(abs(
        updated_account.date_joined - dict_data['date_joined'] <
        datetime.timedelta(seconds = 1)
    ))

    invalid_acc = Account()
    invalid_acc.from_dict(dict_data)
    with self.assertRaises(DataValidationError):
        invalid_acc.update()

def test_delete(self):
    """52-54: Test removing an Account from the database"""
    data = ACCOUNT_DATA[self.rand] # get a random account
    account = Account(**data)
    account.create()

    self.assertIsNotNone(account.id)
    account.delete()

    deleted_account = Account.find(account.id)
    self.assertIsNone(deleted_account)
```



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

**Make your report self-contained so that it is easy to follow without running your code**

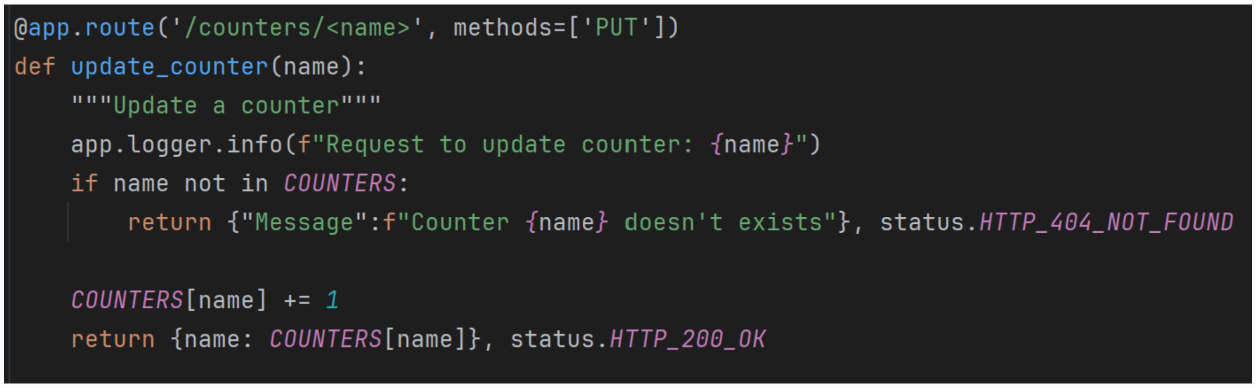
Step 1: `test_update_a_counter(self)`:



```python
def test_update_a_counter(self):
    """It should update a counter"""
    #1: Make a call to Create a counter.
    counter = self.client.post('/counters/test_counter')

    #2: Ensure that it returned a successful return code.
    self.assertEqual(counter.status_code, status.HTTP_201_CREATED)

    #3: Check the counter value as a baseline.
    counter_value = self.client.get('/counters/test_counter')
    self.assertEqual(counter_value.status_code, status.HTTP_200_OK)
    baseline = counter_value.json['test_counter']

    #4: Make a call to Update the counter that you just created.
    updated = self.client.put('/counters/test_counter')

    #5: Ensure that it returned a successful return code.
    self.assertEqual(updated.status_code, status.HTTP_200_OK)

    #6: Check that the counter value is one more than the baseline you measured in step 3.
    get_updated = self.client.get('/counters/test_counter')
    updated_value = get_updated.json['test_counter']
    self.assertEqual(updated_value, baseline + 1)

    nonexistent = self.client.put('/counters/nonexistent')
    self.assertEqual(nonexistent.status_code, status.HTTP_404_NOT_FOUND)
    self.assertIn("Counter nonexistent doesn't exists", nonexistent.json['Message'])
```

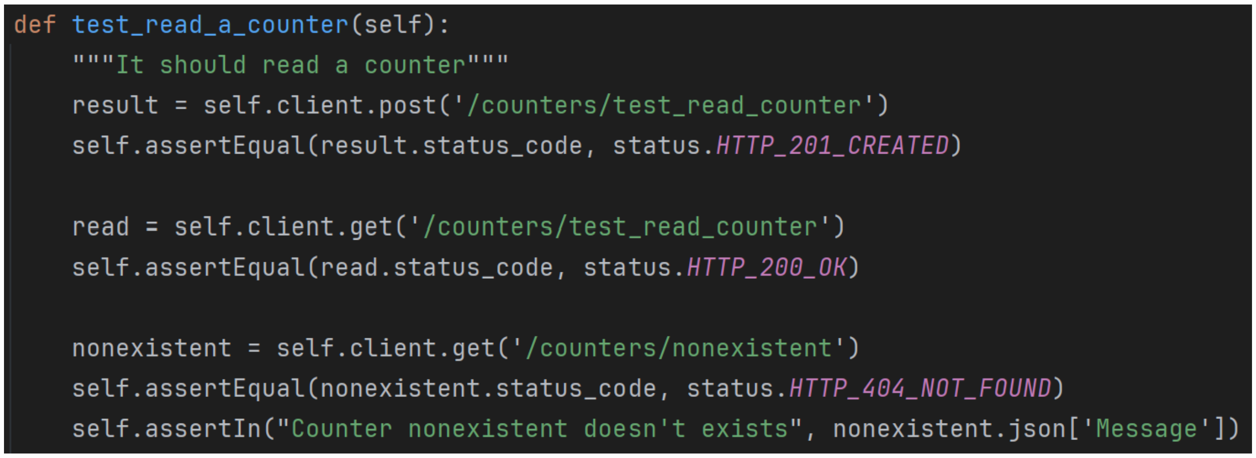
Run `nosetests` -> RED phase: AssertionError: 405 != 200

Step 2: `update_counter(name)`: **REFACTOR**



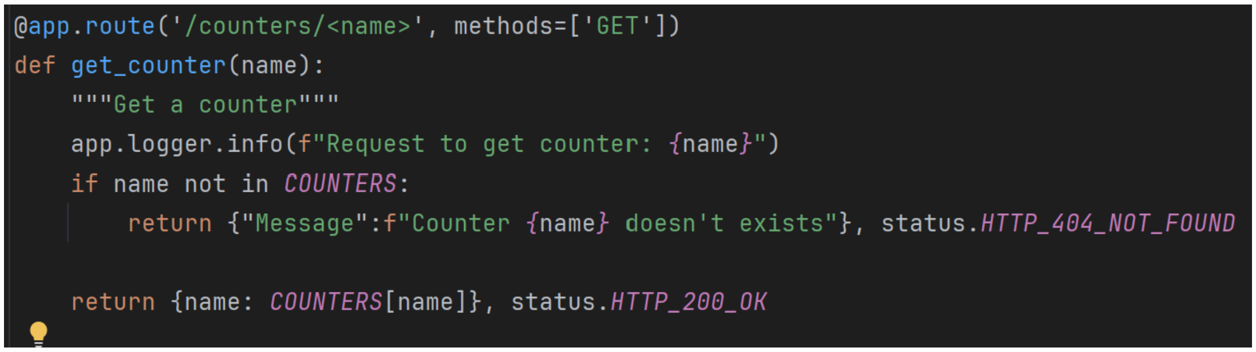
```
@app.route('/counters/<name>', methods=['PUT'])
def update_counter(name):
    """Update a counter"""
    app.logger.info(f"Request to update counter: {name}")
    if name not in COUNTERS:
        return {"Message":f"Counter {name} doesn't exists"}, status.HTTP_404_NOT_FOUND

    COUNTERS[name] += 1
    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

Step 3: `test_read_a_counter(name)`: RED phase: AssertionError: 405 != 200



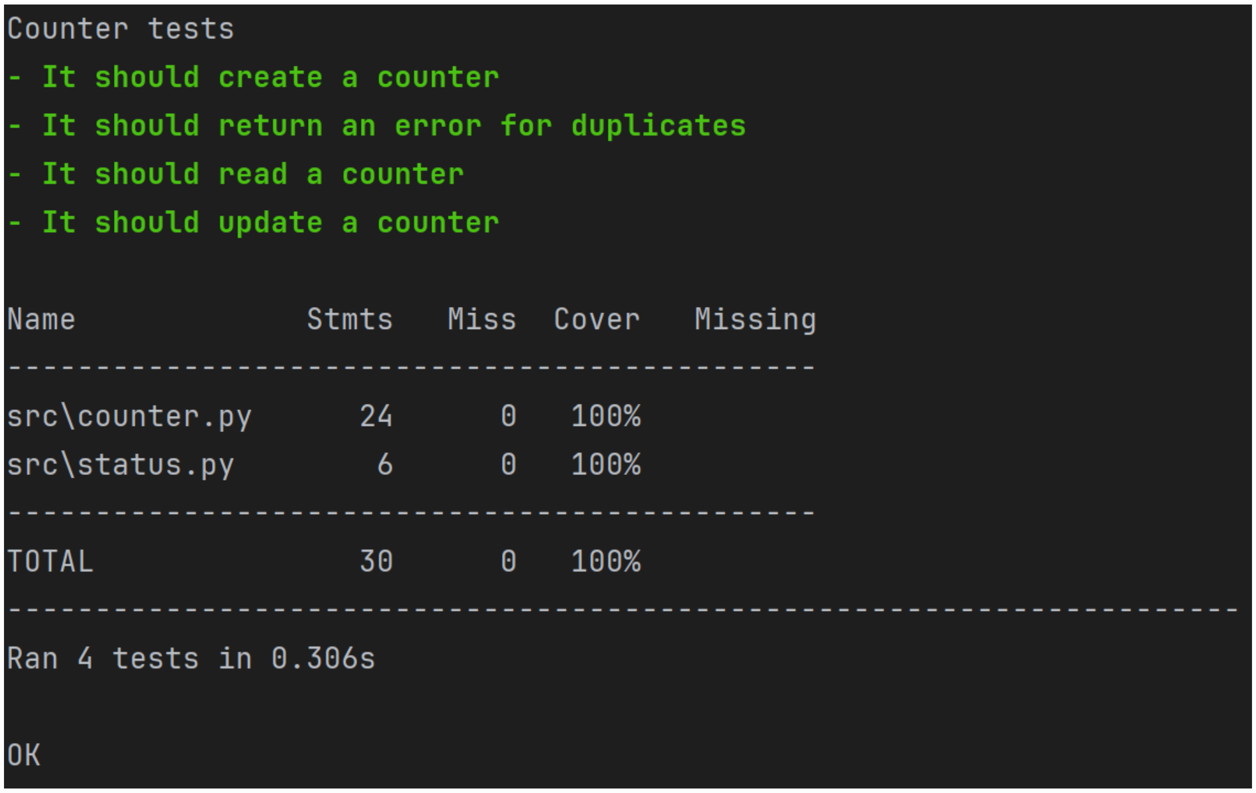
```
def test_read_a_counter(self):
    """It should read a counter"""
    result = self.client.post('/counters/test_read_counter')
    self.assertEqual(result.status_code, status.HTTP_201_CREATED)

    read = self.client.get('/counters/test_read_counter')
    self.assertEqual(read.status_code, status.HTTP_200_OK)

    nonexistent = self.client.get('/counters/nonexistent')
    self.assertEqual(nonexistent.status_code, status.HTTP_404_NOT_FOUND)
    self.assertIn("Counter nonexistent doesn't exists", nonexistent.json['Message'])
```

- Get request method: **REFACTOR**



```
@app.route('/counters/<name>', methods=['GET'])
def get_counter(name):
    """Get a counter"""
    app.logger.info(f"Request to get counter: {name}")
    if name not in COUNTERS:
        return {"Message":f"Counter {name} doesn't exists"}, status.HTTP_404_NOT_FOUND

    return {name: COUNTERS[name]}, status.HTTP_200_OK
```

Run `nosetests` -> GREEN phase:



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

Name               Stmts   Miss  Cover   Missing
----------------------------------------------
src\counter.py        24      0   100%
src\status.py          6      0   100%
----------------------------------------------
TOTAL                 30      0   100%
----------------------------------------------------------------------
Ran 4 tests in 0.306s

OK
```