МІНІСТЕРСТВО ОСВІТИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря Сікорського»
НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС
«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»
Кафедра Системного проектування

Контрольна робота №5

з дисципліни: “Проектування інформаційних систем”

на тему “Модульне тестування (Unit-тести) та рефакторинг”

Виконав:
Студент 4 курсу
групи ДА-71
Задорожний А.С.

Київ – 2020

Мета роботи: оволодіти навичками створення програмного забезпечення за методологією TDD та ознайомитися з процедурами рефакторинга.

Завдання:

1. Використовувати методологію Test Driven Development для створення класів архітектурної програмної моделі.
2. Скласти тестові сценарії, які продемонструють функціювання всіх методів проектованої моделі.
3. Виконати юніт-тестування складових частин (внутрішніх класів), що реалізують об'єкт моделювання.
4. Виконати "зовнішнє" юніт-тестування для API.
5. Провести рефакторинг коду програми, для поліпшення реалізації.

Хід виконання лабораторної роботи:

1. Використовувати методологію Test Driven Development для створення класів архітектурної програмної моделі.

Для демонстрації використання методологію Test Driven Development для створення класів архітектурної програмної моделі розробимо простий калькулятор для додавання двох чисел. На першому кроці напишемо тест із перевіркою очікуваного результату при додаванні двох чисел:

```python
import unittest
from Calculator import Calculator


class TddTest(unittest.TestCase):
    def test_add_method_result(self):
        calc = Calculator()
        result = calc.add(2, 2)
        self.assertEqual(4, result)


if __name__ == '__main__':
    unittest.main()
```

Створимо клас Calculator із методом додавання із заглушкою:

```python
class Calculator(object):
    def __init__(self):
        pass

    def add(self, x, y):
        pass
```

Запустимо тест:

```
Testing started at 4:03 ...
D:\Projects\SysDesign\lab45\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pycharm\_jb_unittest_runner.py" --path
Launching unittests with arguments python -m unittest D:/Projects/SysDesign/lab45/test.py in D:\Projects\SysDesign\lab45



None != 4

Expected :4
Actual   :None
<Click to see difference>

Traceback (most recent call last):
  File "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pycharm\teamcity\diff_tools.py", line 32, in _patched_equals
    old(self, first, second, msg)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 912, in assertEqual
    assertion_func(first, second, msg=msg)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 905, in _baseAssertEqual
    raise self.failureException(msg)
AssertionError: 4 != None

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 60, in testPartExecutor
    yield
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 676, in run
    self._callTestMethod(testMethod)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 633, in _callTestMethod
    method()
  File "D:\Projects\SysDesign\lab45\test.py", line 9, in test_add_method_result
    self.assertEqual(4, result)



Ran 1 test in 0.006s

FAILED (failures=1)
```

Додамо до методу додавання створеного класу Calculator результат додавання:

```
1  class Calculator(object):
2      def __init__(self):
3          pass
4
5      def add(self, x, y):
6          return x + y
```

Запустимо тест:

```
Testing started at 4:08 ...
D:\Projects\SysDesign\lab45\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pych
Launching unittests with arguments python -m unittest D:/Projects/SysDesign/lab45/test.py in D:\Projects\SysDesign\lab45



Ran 1 test in 0.003s

OK

Process finished with exit code 0
```

Змінимо в класі тесту очікуваний результат додавання та запустимо тест:

```
Testing started at 4:09 ...
D:\Projects\SysDesign\lab45\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pycharm\_jb_unittest_runner.py" --path
Launching unittests with arguments python -m unittest D:/Projects/SysDesign/lab45/test.py in D:\Projects\SysDesign\lab45



4 != 5

Expected :5
Actual   :4
<Click to see difference>

Traceback (most recent call last):
  File "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pycharm\teamcity\diff_tools.py", line 32, in _patched_equals
    old(self, first, second, msg)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 912, in assertEqual
    assertion_func(first, second, msg=msg)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 905, in _baseAssertEqual
    raise self.failureException(msg)
AssertionError: 5 != 4

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 60, in testPartExecutor
    yield
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 676, in run
    self._callTestMethod(testMethod)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 633, in _callTestMethod
    method()
  File "D:\Projects\SysDesign\lab45\test.py", line 9, in test_add_method_result
    self.assertEqual(5, result)



Ran 1 test in 0.006s

FAILED (failures=1)
```

Наступним кроком зробимо тест для перевірки додавання двох слів замість чисел:

```python
import unittest
from Calculator import Calculator


class TddTest(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator()

    def test_add_method_returns_correct_result(self):
        result = self.calc.add(2, 2)
        self.assertEqual(4, result)

    def test_calculator_returns_if_both_not_numbers(self):
        self.assertRaises(ValueError, self.calc.add, 'one', 'two')
```

Запустимо тест:

```
Testing started at 4:14 ...
D:\Projects\SysDesign\lab45\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pych
Launching unittests with arguments python -m unittest D:/Projects/SysDesign/lab45/test.py in D:\Projects\SysDesign\lab45


Failure
Traceback (most recent call last):
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 60, in testPartExecutor
    yield
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 676, in run
    self._callTestMethod(testMethod)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 633, in _callTestMethod
    method()
  File "D:\Projects\SysDesign\lab45\test.py", line 14, in test_calculator_returns_if_both_not_numbers
    self.assertRaises(ValueError, self.calc.add, 'one', 'two')
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 816, in assertRaises
    return context.handle('assertRaises', args, kwargs)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 202, in handle
    callable_obj(*args, **kwargs)
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 224, in __exit__
    self._raiseFailure("{} not raised by {}".format(exc_name,
  File "C:\Users\artel\AppData\Local\Programs\Python\Python38-32\lib\unittest\case.py", line 164, in _raiseFailure
    raise self.test_case.failureException(msg)
AssertionError: ValueError not raised by add



Ran 2 tests in 0.006s

FAILED (failures=1)
```

Змінимо відповідно клас Калькулятора вказавши, що додаємо лише ті змінні, які є числами:

```python
class Calculator(object):
    def __init__(self):
        pass

    def add(self, x, y):
        number_types = (int, float, complex)
        if isinstance(x, number_types) and isinstance(y, number_types):
            return x + y
        else:
            return ValueError
```

Запустимо тест:

```
Testing started at 4:19 ...
D:\Projects\SysDesign\lab45\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pych
Launching unittests with arguments python -m unittest D:/Projects/SysDesign/lab45/test.py in D:\Projects\SysDesign\lab45



Ran 2 tests in 0.003s

OK

Process finished with exit code 0
```

Аналогічно робимо якщо одне число із двох є слово:

```python
import unittest
from Calculator import Calculator


class TddTest(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator()

    def test_add_method_returns_correct_result(self):
        result = self.calc.add(2, 2)
        self.assertEqual(4, result)

    def test_calculator_returns_if_both_not_numbers(self):
        self.assertRaises(ValueError, self.calc.add, 'one', 'two')

    def test_calculator_returns_if_x_not_number(self):
        self.assertRaises(ValueError, self.calc.add, 'one', 2)

    def test_calculator_returns_if_y_not_number(self):
        self.assertRaises(ValueError, self.calc.add, 1, 'two')


if __name__ == '__main__':
    unittest.main()
```

```
Testing started at 4:21 ...
D:\Projects\SysDesign\lab45\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1\plugins\python-ce\helpers\pych
Launching unittests with arguments python -m unittest D:/Projects/SysDesign/lab45/test.py in D:\Projects\SysDesign\lab45


Ran 4 tests in 0.003s

OK

Process finished with exit code 0
```

2. Виконаємо рефакторинг коду:

2.1. Можинне присвоювання і розпакування кортежів

Частина коду до рефакторингу:

```python
def cities_coordinates(num_cities):
    x = []
    y = []
    for i in range(0, int(num_cities), 1):
        x.append(randint(0, 100))
        y.append(randint(0, 100))
    return [x, y]
```

Частина коду після рефакторингу:

```
13  def cities_coordinates(num_cities):
14      x, y = [], []
15      for i in range(0, int(num_cities), 1):
16          x.append(randint(0, 100))
17          y.append(randint(0, 100))
18      return [x, y]
```

2.2. Включення списків, словників і множин

Частина коду до рефакторингу:

```
21  def generate_individuals(population_size, num_cities):
22      cities = []
23      for i in range(1, int(num_cities)):
24          cities.append(i)
25      all_paths = []
26      for i in range(population_size):
27          temp = sample(cities, int(num_cities) - 1)
28          temp.insert(0, 0)
29          all_paths.append(temp)
30      return all_paths
```

Частина коду після рефакторингу:

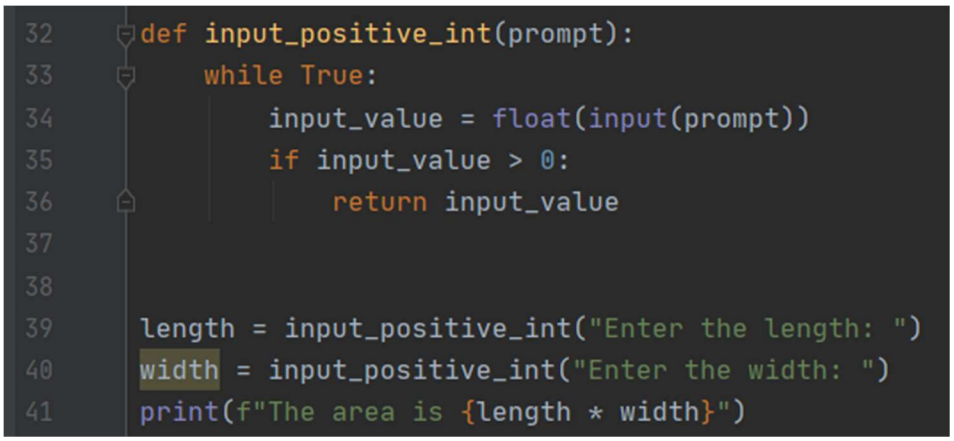
```
21  def generate_individuals(population_size, num_cities):
22      cities = [x for x in range(1, int(num_cities))]
23      all_paths = []
24      for i in range(population_size):
25          temp = sample(cities, int(num_cities) - 1)
26          temp.insert(0, 0)
27          all_paths.append(temp)
28      return all_paths
```

2.3. Винесемо схожі елементи в окрему функцію:

Частина коду до рефакторингу:

```
32  while True:
33      length = float(input("Enter the length: "))
34      if length > 0:
35          break
36      while True:
37          width = float(input("Enter the width"))
38          if length > 0:
39              break
40  print(f"The area is {length * width}")
```

Частина коду після рефакторингу:



```
def input_positive_int(prompt):
    while True:
        input_value = float(input(prompt))
        if input_value > 0:
            return input_value


length = input_positive_int("Enter the length: ")
width = input_positive_int("Enter the width: ")
print(f"The area is {length * width}")
```

Висновки: виконуючи лабораторну роботу було розглянуто створення програмного забезпечення за методологією TDD та процедури рефакторинга. Серед недоліків методології TDD є час розробки програмного забезпечення, але натомість цей метод має значущу перевагу в тому, що програмне забезпечення в результаті є продуманим із багатьох боків. Рефакторинг теж є важливим етапом у розробці ПЗ, бо забезпечує більшу продуктивність, читабельність та масштабованість. Під час виконання лабораторної роботи було розглянуто декілька способів ре факторингу на простих прикладах.