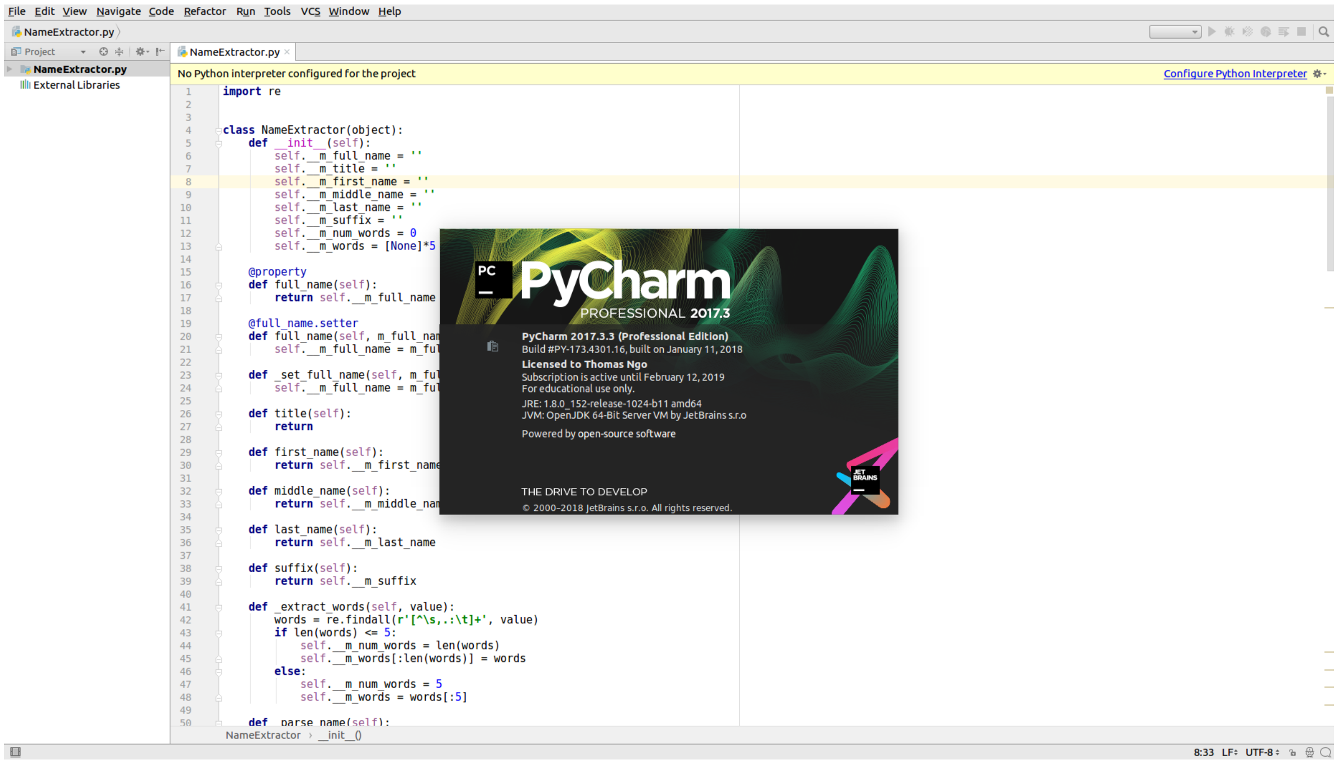
Thomas Ngo
CPSC463-01
Professor Ning chen
Assignment 5

INTRODUCTION:

The main purpose of this paper is about unit testing. With a small portion of the source code like unit, component and function in the same program that is tested. In other words, the notion of unit testing means that every separate part of the product is tested individually to ensure each of them functions and runs as the developer anticipated. This activity makes sure that every unit corresponds correctly to the design specification. After studying the example given by the assignment's instruction, I decided to choose framework named "unittest" and its programming language "python" as my main focus during the course. Also, I have implemented the class NameExtractor along with its members such as ExtractWords, FindTitle, FindSuffix, ParseName in Python language in order to practice implementing unit testing with the testing framework. In addition, even though they are not the requirements in this assignment, I also implemented some extra functions like FirstName, MiddleName and LastName to see check if my implementation is correct or not. In order to perform unit test by using Python framework "unittest", I created another class named "TestNameExtractor" to set up the testing methods and include the test cases. I imported both framework unittest and class NameExtractor in to this new script. There are 8 methods that I defined in class TestNameExtractor. The first two methods are setUpClass and tearDownClass. These methods are known as classmethod. They are used to to separate each unit to be tested so that I could see the performance clearly. Next, I defined setUp method to create instances to test each function (unit) of class NameExtractor. The tearDown method indicates the termination of a unit test. Finally, the remaining methods are the 6 unit tests that I would like to perform: test_full_name, test_title, test_first_name, test_middle_name, test_last_name, and test_suffix. With that said, I provided the screenshot of my IDE for python,which is Pycharm and the source code below.

Screenshot of Pycharm on Ubuntu 16.04


File Edit View Navigate Code Refactor Run Tools VCS Window Help
NameExtractor.py
Project
NameExtractor.py
External Libraries
NameExtractor.py
Configure Python Interpreter
NameExtractor › __init__()
8:33 LF: UTF-8:

No Python interpreter configured for the project

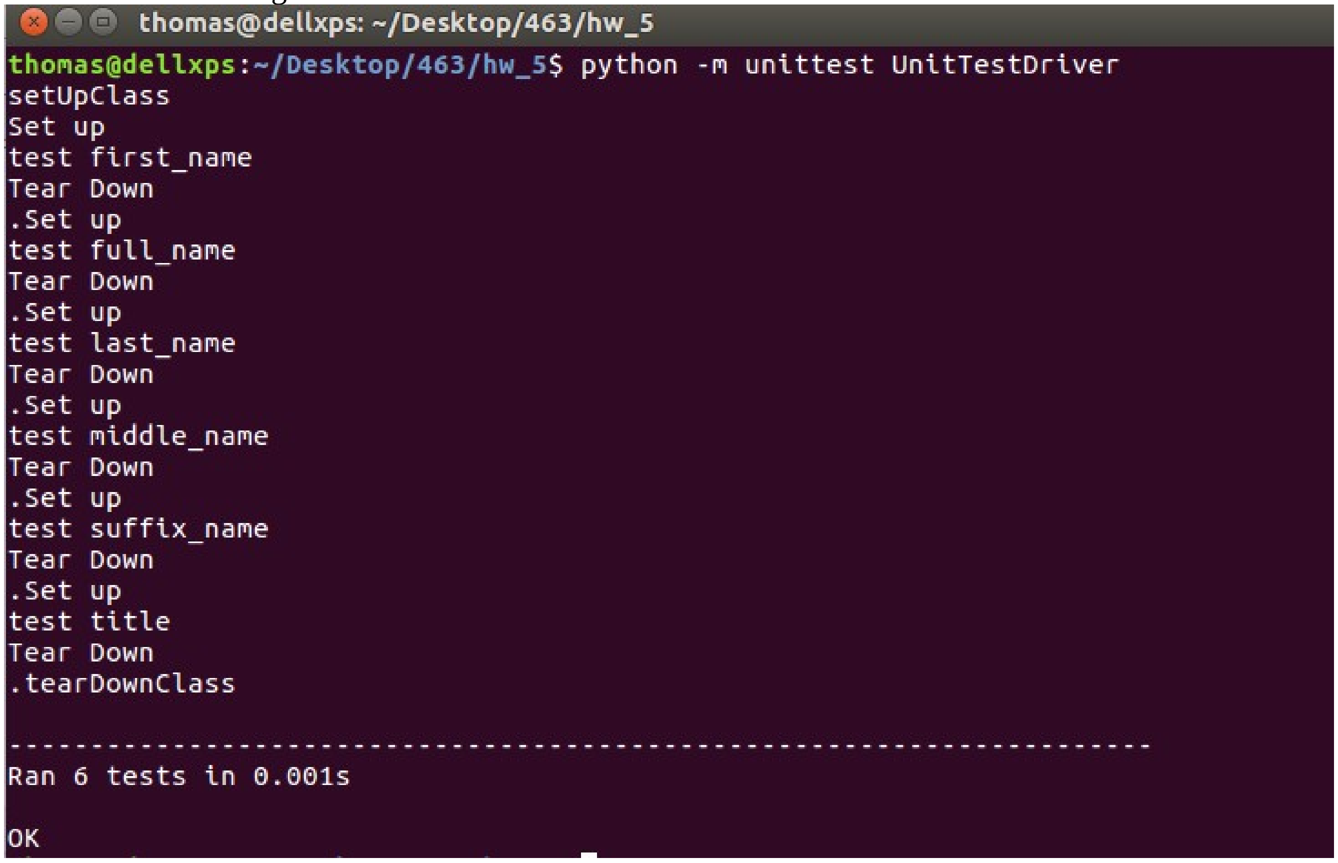
```python
import re


class NameExtractor(object):
    def __init__(self):
        self.__m_full_name = ''
        self.__m_title = ''
        self.__m_first_name = ''
        self.__m_middle_name = ''
        self.__m_last_name = ''
        self.__m_suffix = ''
        self.__m_num_words = 0
        self.__m_words = [None]*5

    @property
    def full_name(self):
        return self.__m_full_name

    @full_name.setter
    def full_name(self, m_full_nam
        self.__m_full_name = m_ful

    def _set_full_name(self, m_ful
        self.__m_full_name = m_ful

    def title(self):
        return

    def first_name(self):
        return self.__m_first_name

    def middle_name(self):
        return self.__m_middle_nam

    def last_name(self):
        return self.__m_last_name

    def suffix(self):
        return self.__m_suffix

    def _extract_words(self, value):
        words = re.findall(r'[^\s,.:\t]+', value)
        if len(words) <= 5:
            self.__m_num_words = len(words)
            self.__m_words[:len(words)] = words
        else:
            self.__m_num_words = 5
            self.__m_words = words[:5]

    def _parse_name(self):
```

PyCharm
PROFESSIONAL 2017.3

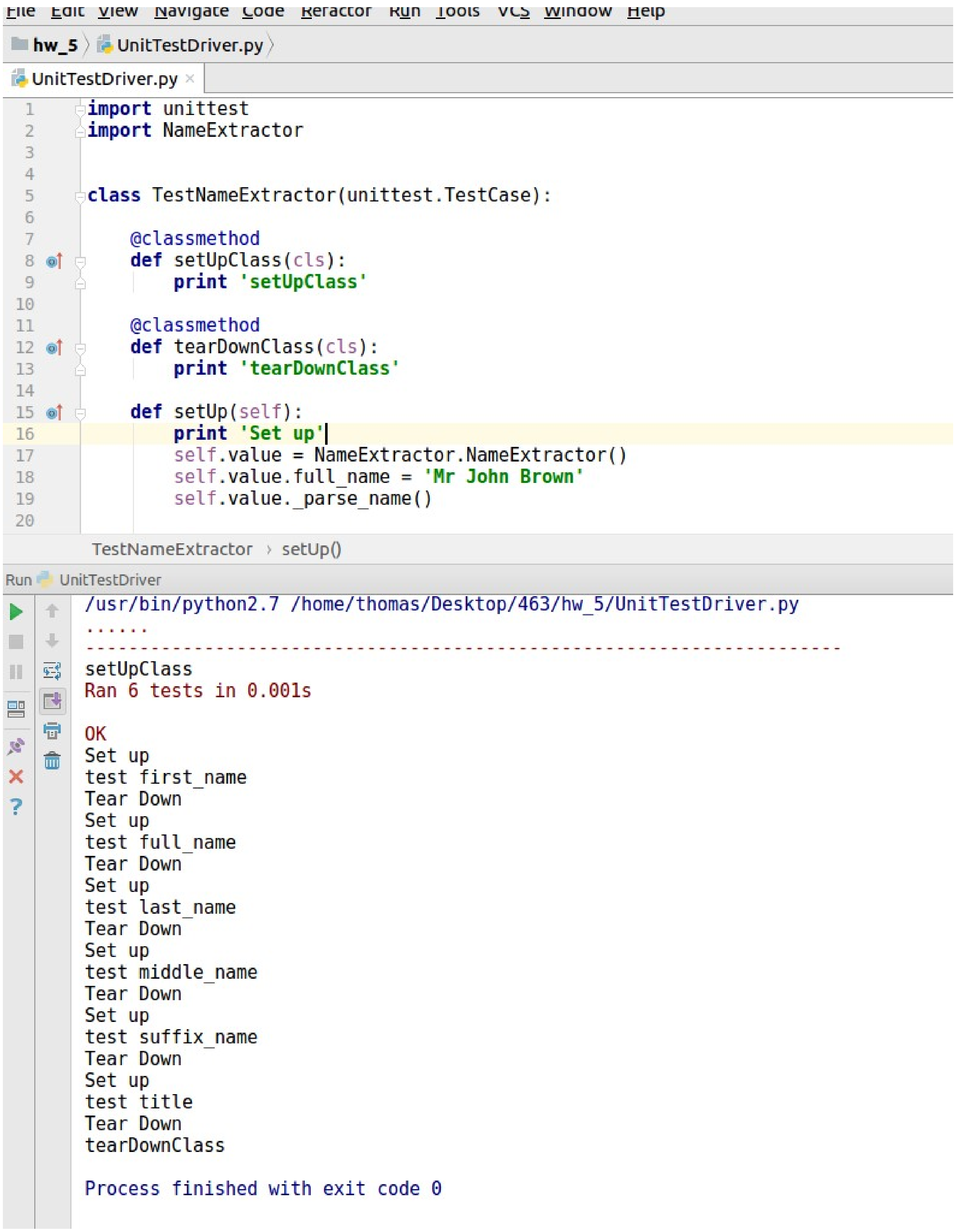
PyCharm 2017.3.3 (Professional Edition)
Build #PY-173.4301.16, built on January 11, 2018
Licensed to Thomas Ngo
Subscription is active until February 12, 2019
For educational use only.
JRE: 1.8.0_152-release-1024-b11 amd64
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o
Powered by open-source software

THE DRIVE TO DEVELOP
© 2000–2018 JetBrains s.r.o. All rights reserved.

Screenshot of running test 2 method and its results



```
thomas@dellxps:~/Desktop/463/hw_5$ python -m unittest UnitTestDriver
setUpClass
Set up
test first_name
Tear Down
.Set up
test full_name
Tear Down
.Set up
test last_name
Tear Down
.Set up
test middle_name
Tear Down
.Set up
test suffix_name
Tear Down
.Set up
test title
Tear Down
.tearDownClass

----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
hw_5 UnitTestDriver.py
UnitTestDriver.py
TestNameExtractor › setUp()
Run UnitTestDriver

```python
import unittest
import NameExtractor


class TestNameExtractor(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        print 'setUpClass'

    @classmethod
    def tearDownClass(cls):
        print 'tearDownClass'

    def setUp(self):
        print 'Set up'
        self.value = NameExtractor.NameExtractor()
        self.value.full_name = 'Mr John Brown'
        self.value._parse_name()

```

```
/usr/bin/python2.7 /home/thomas/Desktop/463/hw_5/UnitTestDriver.py
......
----------------------------------------------------------------------
setUpClass
Ran 6 tests in 0.001s

OK
Set up
test first_name
Tear Down
Set up
test full_name
Tear Down
Set up
test last_name
Tear Down
Set up
test middle_name
Tear Down
Set up
test suffix_name
Tear Down
Set up
test title
Tear Down
tearDownClass

Process finished with exit code 0
```

Source code:

NameExtractor.py

```python
import re
class NameExtractor(object):
    def __init__(self):
        self.__m_full_name = ''
        self.__m_title = ''
        self.__m_first_name = ''
        self.__m_middle_name = ''
        self.__m_last_name = ''
        self.__m_suffix = ''
        self.__m_num_words = 0
        self.__m_words = [None]*5
    @property
    def full_name(self):
        return self.__m_full_name
    @full_name.setter
    def full_name(self, m_full_name):
        self.__m_full_name = m_full_name
    def _set_full_name(self, m_full_name):
        self.__m_full_name = m_full_name
    def title(self):
        return
    def first_name(self):
        return self.__m_first_name
    def middle_name(self):
        return self.__m_middle_name
    def last_name(self):
        return self.__m_last_name
    def suffix(self):
        return self.__m_suffix
    def _extract_words(self, value):
        words = re.findall(r'[^\s,.:\t]+', value)
        if len(words) <= 5:
            self.__m_num_words = len(words)
            self.__m_words[:len(words)] = words
        else:
            self.__m_num_words = 5
            self.__m_words = words[:5]
    def _parse_name(self):
        if self.__m_full_name is not None and self.__m_full_name != '':
            self._extract_words(self.__m_full_name)
            self._find_title()
            self._find_suffix()
            self._find_last_name()
            self._find_first_name()
            self._find_middle_name()
    def _find_title(self):
        title_list = ['Mr.', 'Mr', 'Ms.', 'Ms', 'Miss.', 'Miss', 'Dr.',
'Dr',
                      'Mrs.', 'Mrs','Fr.','Capt.','Lt.', 'Gen.',
'President',
                      'Sister', 'Father', 'Brother', 'Major']
        if self.__m_words is not None:
            if self.__m_words[0] in title_list:
                self.__m_title = self.__m_words[0]
                return 0
            return -1
        return -1
    def _find_suffix(self):
        suffix_list = ['DDS', 'CFA', 'CEO', 'CFO', 'Esq', 'CPA', 'MBA',
'PhD',
                       'MD', 'DC', 'Sr', 'Jr', 'II', 'III', 'IV']
        if self.__m_words[4] is not None:
            self.__m_suffix = self.__m_words[4]
            return 0
        else:
            if self.__m_words[2] is not None and self.__m_words[2] in \
                    suffix_list:
                self.__m_suffix = self.__m_words[2]
                return 0
            if self.__m_words[3] is not None and self.__m_words[3] in \
                    suffix_list:
                self.__m_suffix = self.__m_words[3]
                return 0
        return -1
    def _find_first_name(self):
        if self.__m_num_words >= 2 and self.__m_title == '':
            self.__m_first_name = self.__m_words[0]
            return 0
        if self.__m_num_words > 2 and self.__m_title != '':
            self.__m_first_name = self.__m_words[1]
            return 0
        if self.__m_num_words == 5:
            self.__m_first_name = self.__m_words[1]
            return 0
        return -1
    def _find_middle_name(self):
        if self.__m_num_words == 5 or self.__m_num_words == 4 and \
                self.__m_suffix == '':
            self.__m_middle_name = self.__m_words[2]
            return 0
        if (self.__m_num_words == 4 and self.__m_title == '') or (
                self.__m_num_words == 4 and self.__m_suffix == ''):
            self.__m_middle_name = self.__m_words[1]
            return 0
        if self.__m_num_words == 3 and self.__m_suffix == '' and \
                self.__m_title == '':
```

```python
            self.__m_middle_name = self.__m_words[1]
        return -1
    def _find_last_name(self):
        if self.__m_num_words == 1:
            self.__m_last_name = self.__m_words[0]
            return 0
        if self.__m_num_words == 2:
            self.__m_last_name = self.__m_words[1]
            return 0
        if self.__m_num_words == 5:
            self.__m_last_name = self.__m_words[3]
            return 0
        if (self.__m_num_words == 3 and self.__m_suffix == '') or (
                self.__m_num_words == 4 and self.__m_suffix != ''):
            self.__m_last_name = self.__m_words[2]
            return 0
        else:
            if self.__m_num_words == 3:
                self.__m_last_name = self.__m_words[1]
            elif self.__m_num_words == 4:
                self.__m_last_name = self.__m_words[3]
                return 0
            return -1
    def test_print(self):
        print 'full name:', self.__m_full_name
        print 'title:', self.__m_title
        print 'first name:', self.__m_first_name
        print 'middle name:', self.__m_middle_name
        print 'last name:', self.__m_last_name
        print 'suffix:', self.__m_suffix
        print 'num words:', self.__m_num_words
        print 'words:', self.__m_words
        return 'complete\n'
class ENameExtractorError:
    def __init__(self):
        pass
    def e_name_extractor_error(self, message):
        pass
def main():
    name = NameExtractor()
    name.full_name = 'Mr. John Brown PhD'
    name._parse_name()
    print name.test_print()
    name1 = NameExtractor()
    name1.full_name = 'Mr.    John Brown'
    name1._parse_name()
    print name1.test_print()
    name2 = NameExtractor()
    name2.full_name = 'John Brown, PhD'
    name2._parse_name()
    print name2.test_print()
if __name__ == '__main__':
    main()
```

UniTestDriver.py

```
import unittest
import NameExtractor
class TestNameExtractor(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        print 'setUpClass'
    @classmethod
    def tearDownClass(cls):
        print 'tearDownClass'
    def setUp(self):
        print 'Set up'
        self.value = NameExtractor.NameExtractor()
        self.value.full_name = 'Mr John Brown'
        self.value._parse_name()
        self.value1 = NameExtractor.NameExtractor()
        self.value1.full_name = 'Mr. John Paul Brown Phd'
        self.value1._parse_name()
        self.value2 = NameExtractor.NameExtractor()
        self.value2.full_name = 'John Brown'
        self.value2._parse_name()
        self.value3 = NameExtractor.NameExtractor()
        self.value3.full_name = 'Mr.    John Brown'
        self.value3._parse_name()
        self.value4 = NameExtractor.NameExtractor()
        self.value4.full_name = 'John Brown, PhD'
        self.value4._parse_name()
    def tearDown(self):
        print 'Tear Down'
    def test_full_name(self):
        print 'test full_name'
        self.assertEqual(self.value.full_name, 'Mr John Brown', 'Full name 
is '
                                                                 'not correct')
        self.assertEqual(self.value1.full_name, 'Mr. John Paul Brown Phd',
                         'Full name is not correct')
        self.assertEqual(self.value2.full_name, 'John Brown',
                         'Full name is not correct')
        self.assertEqual(self.value3.full_name, 'Mr.    John Brown',
                         'Full name is not correct')
        self.assertEqual(self.value4.full_name, 'John Brown, PhD',
                         'Full name is not correct')
    def test_title(self):
        print 'test title'
        self.assertEqual(self.value.title(), 'Mr', 'Title is not correct')
        self.assertEqual(self.value1.title(), 'Mr', 'Title is not
correct')
        self.assertEqual(self.value2.title(), '', 'Title is not correct')
        self.assertEqual(self.value3.title(), 'Mr', 'Title is not
correct')
    def test_first_name(self):
        print 'test first_name'
        self.assertEqual(self.value.first_name(), 'John',
                         'First Name is not correct')
        self.assertEqual(self.value1.first_name(), 'John',
                         'First Name is not correct')
        self.assertEqual(self.value2.first_name(), 'John',
                         'First Name is not correct')
        self.assertEqual(self.value3.first_name(), 'John',
                         'First Name is not correct')
        self.assertEqual(self.value4.first_name(), 'John',
                         'First Name is not correct')
    def test_middle_name(self):
        print 'test middle_name'
        self.assertEqual(self.value.middle_name(), '', 'Middle Name is not
'
                                                  'correct')
        self.assertEqual(self.value1.middle_name(), 'Paul', 'Middle Name
is '
                                                      'not correct')
        self.assertEqual(self.value2.middle_name(), '', 'Middle Name is '
                                                   'not correct')
        self.assertEqual(self.value3.middle_name(), '', 'Middle Name is '
                                                   'not correct')
    def test_last_name(self):
        print 'test last_name'
        self.assertEqual(self.value.last_name(), 'Brown', 'Last Name is
not '
                                                    'correct')
        self.assertEqual(self.value1.last_name(), 'Brown', 'Last Name is
not '
                                                     'correct')
        self.assertEqual(self.value2.last_name(), 'Brown', 'Last Name is
not '
                                                     'correct')
        self.assertEqual(self.value3.last_name(), 'Brown', 'Last Name is
not '
                                                     'correct')
        self.assertEqual(self.value4.last_name(), 'Brown', 'Last Name is
not '
                                                     'correct')
    def test_suffix(self):
        print 'test suffix_name'
        self.assertEqual(self.value.suffix(), '', 'Suffix is not correct')
        self.assertEqual(self.value1.suffix(), 'Phd', 'Suffix is not
correct')
```

```
        self.assertEqual(self.value2.suffix(), '', 'Suffix is not
correct')
        self.assertEqual(self.value3.suffix(), '', 'Suffix is not
correct')
        self.assertEqual(self.value4.suffix(), 'PhD', 'Suffix is not
correct')
if __name__ == '__main__':
    unittest.main()
```