테스트 보고서

컴퓨터공학부
20153243 허준녕

UI는 유연한 과제 진행을 위해 유닛테스트를 진행하지 않았습니다.
핵심 로직과 예외 처리가 명시된 과제 목표를 충족 시키는지를 중점으로 테스트를 진행했습니다.

테스트를 진행한 모듈은 data 모듈의 dataHandler 클래스, io 모듈의 fileHandler 클래스 입니다. ui로 부터 입력받거나(Entry), dataHandler의 data_dic 등의 종속성 문제는 간편한 테스트를 위해 관련 데이터를 모두 테스트함수의 인자로 받도록 수정하여 테스트를 진행하였습니다.

1.dataHandler 클래스의 유닛테스트

(1) setIndex 메소드 테스트

```
def setIndex(self,dic):
        self.data_dic = dic
        self.temp_idx = 1
        for k,v in self.data_dic.items():
                if k == self.temp  idx:
                        self.temp_idx += 1
        self.index =self.temp_idx
        return self.index
```

인덱스 부여를 위한 기존의 함수를 테스트를 위해 데이터를 인자로 받습니다.

아래는 테스트 함수입니다.

```
def test_SetIndexAndAdd(self):
        self.temp_dic = {1:["test1",50],2:["2test",70],5:["i'm test",100]}
        self.assertEqual(3,self.setIndex(self.temp_dic))
```

예측한 빈 인덱스가 리턴 된 인덱스와 일치하면 테스트 통과입니다.

(2) add 메소드 테스트

```
def add(self,Name,Score,dic):
        self.data_dic = dic
        self.name = Name.strip()
        if self.name == "":
                print "[추가  실패] 이름란이  공백  입니다."
                return False
        try:
                self.score = eval(Score)
        except:
                print "[추가  실패] 점수를  정확히  입력하세요."
                return False
        for k,v in self.data_dic.items():
```

```
            if v[0] == self.name :
                print "[추가 실패] 동일한 이름이 이미 존재합니다."
                return False
        self.index = self.setIndex(dic)
        self.data_dic[self.index] = [self.name,eval(Score)]
        print "성공적으로 추가하였습니다"
        print self.data_dic
        return True
```

엔트리로부터 가져오는 이름과 점수를 인자로 받게 수정하고 데이터 딕셔너리 역시 인자로 받게 했습니다. 이름이 공백이거나 동일한 이름이 있는 경우, 정상적으로 성공한 경우 3 가지를 테스트를 했습니다.
아래는 테스트 함수입니다.

```
def test_SetIndexAndAdd(self):
        self.assertFalse(self.add("\t","77",self.temp_dic))
        self.assertFalse(self.add("test1","77",self.temp_dic))
        self.assertTrue(self.add("unitTest","89",self.temp_dic))
```

(3) delete 메소드 테스트

```
def delete(self,Number,DIC):
    try:
        self.number = eval(Number)
    except:
        print "[삭제 실패] 정확한 번호를 입력하세요."
        return False
    self.hasNum = False
    self.data_dic = DIC
    for k,v in self.data_dic.items():
        if k == self.number :
            self.hasNum = True
            break
    if not self.hasNum:
        print "[삭제 실패] 존재하지 않는 번호입니다."
        return False
    else:
        self.data_dic.pop(self.number)
        print "성공적으로 제거하였습니다."
        print self.data_dic
        return True
```

기존의 함수에서 번호와 사전을 인자로 입력 받도록 수정한 함수입니다.
테스트의 목표는 올바른 번호, 존재하는 번호, 성공한 경우 3 가지를 테스트 했습니다.

```
def test_delete(self):
    self.temp_dic = {1: ["test1", 50], 2: ["2test", 70], 5: ["i'm test", 100]}
    self.assertFalse(self.delete(" ", self.temp  dic))
    self.assertFalse(self.delete("9", self.temp_dic))
    self.assertTrue(self.delete("2", self.temp_dic))
```

(4) sortName,sortScore,sortScore_REV 메소드 테스트

번호순 정렬은 테스트를 진행하지 않았습니다. 파이썬 딕셔너리는 기본적으로 키를 기준으로 정렬을 하는데 저의 경우 키가 번호를 가지게 되어 자동 정렬되는 결과를 보여주기에 테스트를 생략했습니다.

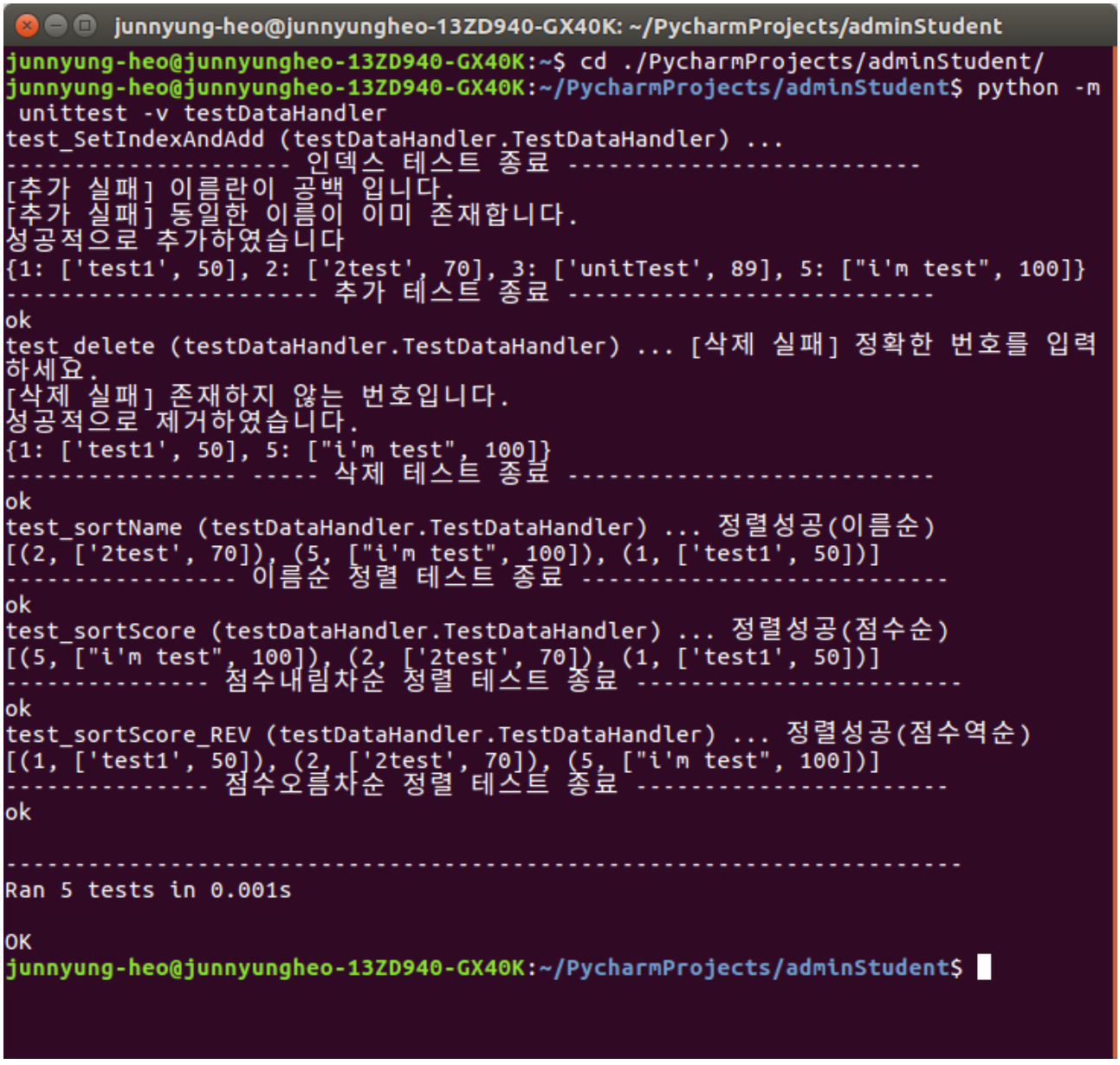
```
def sortName(self,DIC):
        self.data_dic = DIC
        self.name_sorted = sorted(self.data_dic.items(),key = lambda x:x[1][0])
        self.name_list = []
        for elems in self.name_sorted:
                self.name_list.append(elems[1][0])
        for idx in range(len(self.name_list)-1):
                if self.name_list[idx] > self.name_list[idx+1]:
                        print "정렬실패(이름순)"
                        return False
        print "정렬성공(이름순)"
        print self.name_sorted
        return True

def sortScore(self,DIC):
        self.data  dic = DIC
        self.score_sorted = sorted(self.data_dic.items(),key = lambda x:x[1][1],reverse=True)
        self.score_list = []
        for elems in self.score_sorted:
                self.score  list.append(elems[1][1])
        for idx in range(len(self.score_list)-1):
                if self.score_list[idx] < self.score_list[idx+1]:
                        print "정렬실패(점수순)"
                        return False
        print "정렬성공(점수순)"
        print self.score_sorted
        return True

def sortScore_REV(self,DIC):
        self.data_dic = DIC
        self.score_sorted = sorted(self.data_dic.items(), key=lambda x: x[1][1])
        self.score_list = []
        for elems in self.score_sorted:
                self.score  list.append(elems[1][1])
        for idx in range(len(self.score_list) - 1):
                if self.score_list[idx] > self.score_list[idx + 1]:
                        print "정렬실패(점수역순)"
                        return False
        print "정렬성공(점수역순)"
        print self.score_sorted
        return True
```

각 함수들의 정렬 성공의 기준은 sorted 함수에 의해 리턴된 튜플 원소들을 각각 비교 연산을 하여 기준에 맞게 정렬 되었는지를 확인합니다.

다음은 정렬 테스트 함수들입니다.

```
def test_sortName(self):
        self.temp_dic = {1: ["test1", 50], 2: ["2test", 70], 5: ["i'm test", 100]}
        self.assertTrue(self.sortName(self.temp_dic))
        print "---------- 이름순 정렬 테스트 종료 ----------"
def test_sortScore(self):
        self.temp_dic = {1: ["test1", 50], 2: ["2test", 70], 5: ["i'm test", 100]}
        self.assertTrue(self.sortScore(self.temp_dic))
        print "---------- 점수내림차순 정렬 테스트 종료 ----------"
def test_sortScore_REV(self):
        self.temp_dic = {1: ["test1", 50], 2: ["2test", 70], 5: ["i'm test", 100]}
        self.assertTrue(self.sortScore_REV(self.temp_dic))
        print "---------- 점수오름차순 정렬 테스트 종료 ----------"
```

dataHandler 클래스 유닛 테스트 결과:



설계한 테스트가 모두 성공하여 ok가 출력됨을 확인할 수 있습니다.

2.fileHandler 클래스의 유닛테스트

(1) outFile 메소드 테스트

```
def outFile(self,FileName,dictionary):
        self.fileName = FileName.strip()
        if self.fileName == "":
                print "[저장 실패] 파일이름이 공백 입니다."
                return False
        f = open("./" + self.fileName + ".txt", 'w')
        for k, v in dictionary.items():
                f.write(str(k) + "\t" + v[0] + "\t" + str(v[1]) + "\n")
        f.close()
        self.FileName = ""
        print "성공적으로 저장하였습니다. (파일이름: " + self.fileName + ")"
        return True
```

기존의 outFile 함수에서 파일이름과 데이터 딕셔너리를 가져오는 부분을 인자로부터 가져오도록 대체했습니다.

테스트 목표는 잘못된 파일이름을 입력했을 때 실패 하는지와 올바른 이름을 입력했을 때 정상적으로 저장 되는지를 테스트하는 것 입니다.

테스트 함수는 다음과 같습니다.

```
def testOutFile(self):
        self.assertFalse(self.outFile(" ",{1:["test1",50],2:["2test",70],5:["i'm test",100]}))
        self.assertTrue(self.outFile("fileHandler_unittest", {1: ["test1", 50], 2: ["2test", 70], 5: ["i'm test", 100]}))
```

(2) inFile 메소드 테스트

```
def inFile(self,FileName):
        self.fileName = FileName.strip()
        if self.fileName == "":
                print "[읽기 실패] 파일이름이 공백 입니다."
                return False
        try:
                f = open("./" + self.fileName + ".txt", 'r')
        except:
                print "[읽기 실패] 존재하지 않는 파일입니다."
                return False
        self.data_dic = {}
        lines = f.readlines()
        for line in lines:
                self.elems = line.strip().split("\t")
                self.data_dic[eval(self.elems[0])] = [self.elems[1], eval(self.elems[2])]
        f.close()
        print "성공적으로 파일을 읽었습니다. (파일이름: " + self.fileName + ")"
        print "읽은 데이터 딕셔너리"
        print self.data_dic
        return True
```

파일 이름을 역시 인자로 받도록 대체 했으며, 파일 입력 실패 테스트는 유닛 테스트로, 정상적인 경우 데이터 상태를 육안으로 확인하는 테스트를 진행 했습니다.

```
def testInFile(self):

        self.assertFalse(self.inFile("\n"))
        self.assertTrue(self.inFile("fileHandler_unittest"))
```

위에는 테스트 함수이며 아래는 테스트 파일 내용입니다.

fileHandler_unittest.txt

```
1   test1  50

2  2test  70
5  i'm test   100
```

fileHandler 클래스 유닛 테스트 결과:

```
junnyung-heo@junnyungheo-13ZD940-GX40K:~/PycharmProjects/adminStudent$ python -m
 unittest -v testFileHandler
testInFile (testFileHandler.TestFileHandler) ... [읽기 실패] 파일이름이 공백 입
니다.
성공적으로 파일을 읽었습니다. (파일이름: fileHandler_unittest)
읽은 데이터 딕셔너리
{1: ['test1', 50], 2: ['2test', 70], 5: ["i'm test", 100]}
ok
testOutFile (testFileHandler.TestFileHandler) ... [저장 실패] 파일이름이 공백 입
니다.
성공적으로 저장하였습니다. (파일이름: fileHandler_unittest)
ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

입력한 파일의 데이터를 정상적으로 딕셔너리화 됨을 육안으로 확인할 수 있으며

모든 테스트가 성공했다는 ok 싸인을 얻었습니다.