Assignment 2 Solution

?

February 16, 2019

Introductory blurb.

1 Testing of the Original Program

Description of approach to testing. Rationale for test case selection. Summary of results. Any problems uncovered through testing.

2 Results of Testing Partner's Code

Consequences of running partner's code. Success, or lack of success, running test cases. Explanation of why it worked, or didn't.

3 Critique of Given Design Specification

Advantages and disadvantages of the given design specification.

4 Answers

1.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author wengy12
#  @brief
#  @date 02/11/2019
from SeqADT import *
from enum import Enum
from typing import *


class GenT(Enum):
    male = 0
    female = 1


class DeptT(Enum):
    civil = 0
    chemical = 1
    electrical = 2
    mechanical = 3
    software = 4
    materials = 5
    engphys = 6


class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: DeptT
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @author wengy12
#  @brief
#  @date 02/11/2019


class SeqADT:

    def __init__(self, s):
        self.s = s
        self.i = 0
        #return self

    def start(self):
        self.i = 0

    def next(self):
        if self.i >= len(self.s):
            raise StopIteration
        self.i = self.i + 1
        return self.s[self.i - 1]

    def end(self):
        return (self.i >= len(self.s))
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @author wengy12
#  @brief
#  @date 02/11/2019

from StdntAllocTypes import *


class dCapALst:

    def init(self):
        self.s = set()

    def add(self, d, n):
        for i in self.s:
            if i[0] == d:
                raise KeyError
        self.s.add((d, n))

    def remove(self, d):
        n = len(self.s)
        for i in self.s:
            if i[0] == d:
                self.s.remove(i)
        if n != len(self.s) + 1:
            raise KeyError

    def elm(self, d):
        for i in self.s:
            if i[0] == d:
                return True
        return False

    def capacity(self, d):
        for i in self.s:
            if i[0] == d:
                return i[1]
        raise KeyError

DCapALst = dCapALst()
```

H Code for AALst.py

```
## @file AAList.py
#  @author wengy12
#  @brief
#  @date 02/11/2019

from StdntAllocTypes import *


class AALst:

    s = {}

    def __init__(self):
        for i in DeptT:
            self.s[i] = []

    def add_stdnt(self, dep, m):
        self.s[dep].append(m)

    def lst_alloc(self, d):
        print(self.s[d])

    def num_alloc(self, d):
        print(len(self.s[d]))
```

I Code for SALst.py

```
## @file SALst.py
#  @author wengy12
#  @brief
#  @date 02/11/2019

from StdntAllocTypes import *
from AALst import *
from DCapALst import *


class sALst:

    def init(self):
        self.s = set()

    def add(self, m, i):
        for j in self.s:
            if j[0] == m:
                raise KeyError
        self.s.add((m, i))

    def remove(self, m):
        n = len(self.s)
        for i in self.s:
            if i[0] == m:
                self.s.remove(i)
        if n != len(self.s) + 1:
            raise KeyError

    def elm(self, m):
        for i in self.s:
            if i[0] == m:
                return True
        return False

    def info(self, m):
        for i in self.s:
            if i[0] == m:
                return i[1]
        raise KeyError

SALst = sALst()
```

J Code for Read.py

K Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @title Sequence ADT
#  @author Dominik Buszowiecki
#  @date February 9, 2019


## @brief An abstract data type that represents a sequence of values
class SeqADT:

    ## @brief SeqADT constructor
    #  @details Initializes the state variables of SeqADT. The state variables are a list that
    #           is given as a parameter and a variable used to index the list
    #           (initialized to 0).
    #  @param x A list of values
    def __init__(self, x: list):
        self.__s = x
        self.__i = 0

    ## @brief start will reset the index state variable to 0
    def start(self):
        self.__i = 0

    ## @brief next will return the next value in the sequence
    #  @exception throws StopIteration if there is no more items in the sequence
    #  @return value of next item in the sequence
    def next(self):
        if self.__i >= len(self.__s):
            raise StopIteration
        self.__i += 1
        return self.__s[self.__i - 1]

    ## @breif end will check if there are more items in the sequence
    #  @return True if there are no more items in the sequence, otherwise False
    def end(self) -> bool:
        return self.__i >= len(self.__s)
```

L Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @title Department Capacity Association List
#  @author Dominik Buszowiecki
#  @date February 9, 2019

from StdntAllocTypes import *


## @brief An abstract data type containing the capacities of engineering departments as a list
class DCapALst:

    ## @brief Initializes the Department Capacity List to be empty
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief Adds a department and its capacity to the list
    #  @exception throws KeyError if the given department has been added before
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @param n An integer representing the capacity of the department (d parameter)
    @staticmethod
    def add(d: DeptT, n: int):
        for i in DCapALst.s:
            if d == i[0]:
                raise KeyError
        DCapALst.s.append((d, n))

    ## @breif Removes a department and its capacity from the list
    #  @exception throws KeyError if the given department is not in DCapALst
    #  @param d A department of type StdntAllocTypes.DeptT to be removed
    @staticmethod
    def remove(d: DeptT):
        for i in range(0, len(DCapALst.s)):
            if d == DCapALst.s[i][0]:
                del DCapALst.s[i]
                return
        raise KeyError

    ## @brief elm checks if a department has been added
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @return True if the department has been added, otherwise False
    @staticmethod
    def elm(d: DeptT) -> bool:
        for i in DCapALst.s:
            if d == i[0]:
                return True
        return False

    ## @brief capacity returns the capacity of a department
    #  @exception throws KeyError if the department given is not in DCapALst
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @return An integer representing the capacity of the department given as a parameter.
    @staticmethod
    def capacity(d: DeptT) -> bool:
        for i in DCapALst.s:
            if d == i[0]:
                return i[1]
        raise KeyError
```

M Code for Partner's SALst.py

```
## @file SALst.py
#  @title Student Association List
#  @author Dominik Buszowiecki
#  @date February 9, 2019

from StdntAllocTypes import *
from AALst import *
from DCapALst import *
from typing import Callable


## @brief An abstract data type of all first year engineerng students
class SALst:

    ## @brief init initializes the list of students to be empty
    @staticmethod
    def init():
        SALst.s = []

    ## @brief Adds a student into the SALst
    #  @exception throws KeyError if the student given has been added before
    #  @param m A string of a student's macid
    #  @param i Information of a student given with the data type StdntAllocTypes.SInfoT
    @staticmethod
    def add(m: str, i: SInfoT):
        for student in SALst.s:
            if student[0] == m:
                raise KeyError
        SALst.s.append((m, i))

    ## @brief Removes a student from the SALst
    #  @exception throws KeyError if a student to be removed is not found
    #  @param m A string of a student's macid
    @staticmethod
    def remove(m: str):
        for i in range(0, len(SALst.s)):
            if SALst.s[i][0] == m:
                del SALst.s[i]
                return
        raise KeyError

    ## @brief elm checks if a student is already in the SALst
    #  @param m A string of a student's macid
    #  @return True if a student is in SALst, otherwise False
    @staticmethod
    def elm(m: str):
        for student in SALst.s:
            if student[0] == m:
                return True
        return False

    ## @brief returns the information assoaciated with a student
    #  @exception throws KeyError if the student is not found
    #  @param m A string of a student's macid
    #  @return A students information with the type StdntAllocTypes.SInfoT
    @staticmethod
    def info(m: str) -> SInfoT:
        for student in SALst.s:
            if student[0] == m:
                return student[1]
        raise KeyError

    ## @brief Sorts a subset of students based on GPA
    #  @details The method is given a function that is able to filter a student. The filter
    #           function takes in a student (SInfoT) and returns True if they pass the filter.
    #           The method will return a list of macids that passed the filter, sorted by
    #           their GPA in descending order.
    #  @param f A filtering function that returns a boolean
    #  @return A list of strings (each string is a macid) sorted by their GPA in
    #          descending order
    @staticmethod
    def sort(f: Callable[[], bool]) -> list:
        temp_l = []
        for student in SALst.s:
            if f(student[1]):
                temp_l.append(student)
```

```
    temp_l = sorted(temp_l, key=lambda gpa_student: gpa_student[1].gpa, reverse=True)
    sorted_list = []
    for i in temp_l:
        sorted_list.append(i[0])
    return sorted_list

## @brief Computes the average of a particular subset of students
#  @details The method is given a function that is able to filter a student. The function
#           takes in a student(SInfoT) and returns True if they pass the filter. The
#           method will then compute the average GPA amongst students who passed the
#           filter.
#  @exception throws ValueError if there are no students that pass the filter function.
#  @param f A filtering function that returns a boolean
#  @return A float representing the average GPA amongst a subset of students
@staticmethod
def average(f: Callable[[], bool]) -> float:
    i = 0
    size = 0
    for student in SALst.s:
        if f(student[1]):
            i += student[1].gpa
            size += 1
    if size == 0:
        raise ValueError
    else:
        return i / size

## @breif Allocates students in SALst into their program
#  @details Students are allocated into a department in AALst.
#  Students with free choice are allocated first. The remaining students are allocated in
#  a order based on their GPA, a student is allocated into their highest preferred choice
#  that is not full in capacity.
#  @exception throws RuntimeError if all of a student's choices are full.
@staticmethod
def allocate():
    AALst.init()
    f = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    for student in f:
        ch = SALst.info(student).choices
        AALst.add_stdnt(ch.next(), student)

    s = SALst.sort(lambda t: not t.freechoice and t.gpa >= 4.0)
    for m in s:
        ch = SALst.info(m).choices
        alloc = False
        while not alloc and not ch.end():
            d = ch.next()
            if AALst.num_alloc(d) < DCapALst.capacity(d):
                AALst.add_stdnt(d, m)
                alloc = True
        if not alloc:
            raise RuntimeError
```