FP5.0 Module-4 Assignments

Batch Name:	Infosys Internship Batch 2018
Enrollment No:	R110215062
SAP ID:	500044606
NAME:	Kanwaljit Singh
Semester:	VI
Branch:	CSE CCVT

Assignment 1

Identify Entities, list their properties (attributes) in following scenario:

a. *in a college library*
b. *in a classroom*

Solution :-

a. Entities in a college library
    - Students
    - Books
    - Librarian

b. Entities in a classroom
    - Students
    - Teacher

Assignment 2

List properties (attributes) and behaviors(methods) for following entities:

- *A Facebook account*
- *Bank Account*
- *Employee*

Solution :-

- A Facebook Account
    - Properties
        - Username
        - Password
        - Profile Pic
        - Posts

        - Friends list

    - Behaviours
        - Change Username()
        - Change Password()
        - Change Profile Pic()
        - Edit Posts()
        - Add Friend()
        - Remove Friend()

- Bank Account
    - Properties
        - Account Number
        - Account Name
        - Balance

    - Behaviours
        - Add Balance()
        - Remove Balance()
        - Check Balance()
        - Update Name()

- Employee
    - Properties
        - ID
        - Name
        - Salary

    - Behaviours
        - Get ID()
        - Update Name()
        - Check Salary()

Assignment 3

List entities, their properties in an online shopping website from following description:

- *The registration page takes customer name, mobile number, email id as input from new customer. A customer id is generated by the website on successful registration.*
- *On login, customer can view the list of product. For every product- product name, id, prize, description and its picture is displayed.*
- *The selected items by customer go to shopping cart. The shopping cart displays list of ordered items with their id, name, quantity and price.*
- *Customer can then place the order by entering shipping detail.*
- *The shop owner can see customer order information like order id, date of purchase, shipping address, customer information and every ordered item details like product id, product name, price and quantity*

Solution :-

Entities

- Customer
    - Properties
        - Name
        - Mobile
        - Email ID
        - Customer ID
    - Behaviours
        - Login()
        - Register()
        - Update()
        - View Products()
        - Purchase Products

- Product
  - Properties
    - Name
    - ID
    - Description
    - Picture

  - Behaviours
    - Change Picture()
    - Change Name()

- Shopping Cart
  - Properties
    - Order ID
    - Order Date
    - List of Products and their quantities
    - Shipping Address
    - Billing Amount
    - Customer Details

  - Behaviours
    - Add Product()
    - Remove Product()
    - Update Quantity()
    - Change Address()
    - Get Order ID()
    - Get Order Date()
    - Payment()

- Shop Owner
  - Behavour
    - Get Shopping Cart Details()

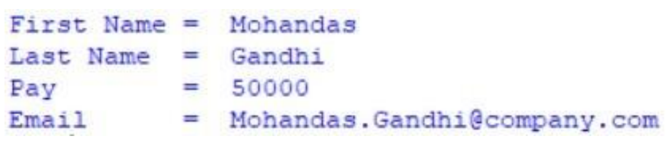
Assignment 4

- *Create a class Employee with following properties*
  - *First Name*
  - *Last Name*
  - *Pay*
  - *Email : should be automatically generated as*
    - *Firstname + '.' + Lastname + "@company.com"*
- *Test the code with following information of an Employee:*
  - *First name is : Mohandas*
  - *Last name is : Gandhi*
  - *Pay is : 50000*

Solution :-

```
class Employee:
    def___init__(self, f_name, l_name, pay):
        self.first_name = f_name
        self.last_name = l_name
        self.pay = pay
        self.email = self.first_name + "." + self.last_name + "@company.com"

emp = Employee("Mohandas", "Gandhi", 50000)

print("First Name = ", emp.first_name)
print("Last Name = ", emp.last_name)
print("Pay        = ", emp.pay)
print("Email      = ", emp.email)
```

Output :-



```
First Name =  Mohandas
Last Name  =  Gandhi
Pay        =  50000
Email      =  Mohandas.Gandhi@company.com
```

Assignment 5

In the previous example add following methods

- *getEmail : should return the email id*
- *getFullName : should return full name (first name followed by last name)*
- *getPay : should return the pay*

Test the implementation with object of emp_1 as follows

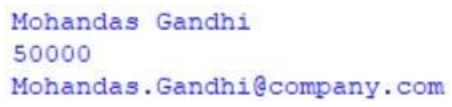
```
emp_1 = Employee('Mohandas','Gandhi',50000)
print(emp_1.getFullName())
print(emp_1.getPay())
print(emp_1.getEmail())
```

Solution :-

```
class Employee:
    def__init__(self, f_name, l_name, pay):
        self.first_name = f_name
        self.last_name = l_name
        self.pay = pay
        self.email = self.first_name + "." + self.last_name + "@company.com"

    def getEmail(self):
        return self.email

    def getFullName(self):
        return self.first_name + " " + self.last_name

    def getPay(self):
        return self.pay

emp_1 = Employee('Mohandas','Gandhi',50000)
```

```
print(emp_1.getFullName())
print(emp_1.getPay())
print(emp_1.getEmail())
```

Output :-



```
Mohandas Gandhi
50000
Mohandas.Gandhi@company.com
```

Assignment 6

```
class Account:
    def___init__(self, initial_amount):
         self.balance = initial_amount
    def withdraw(self,amount):
        self.balance = self.balance – amount
    def deposit(self,amount):
        self.balance = self.balance + amount
ac = Account(1000)
ac.balance = 2000 #stmt1
ac.balance = -1000 #stmt2
print(ac.balance) #stmt3
```

List the risk associated with the implementation of Account class. Suggest a solution.

Solution

- The balance can be set to very high/low value accidentally. (#stmt1)
- The balance can be accessed or changed by user of the class.
- The balance can be set to non-permitted value (#stmt2)

Make balance a private variable (__balance)

```
def__init__(self, initial_amount):
   self.__balance = initial_amount
```

Assignment 7

```
class Dog:

    tricks = []

    def__init__(self, name):
        self.name = name

    def add_trick(self, trick):
        self.tricks.append(trick)

d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
```

A dog trainer had two dogs- Fido and Buddy.Fido was trained a trick of "roll over" and Buddy was learned "play dead". Is the code written correctly to represent this situation?

A. Yes

- *prove by printing print(d.tricks)and print(e.tricks)*

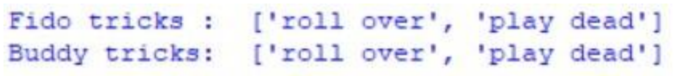
B. No

- *if no, then rewrite the code*

Solution

No
Although the add trick uses self instead of class to access variable it will not create seperate instance since the variable used is mutable and will be directly modified.

Output of the code :-

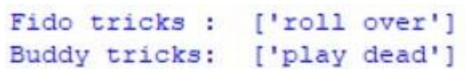


```
Fido tricks :  ['roll over', 'play dead']
Buddy tricks:  ['roll over', 'play dead']
```

Correct Approach

```
class Dog:

    def___init__(self, name):
        self.name = name
        self.tricks = []                    #It will create seperate list for seperate dogs

    def add_trick(self, trick):
        self.tricks.append(trick)


d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')

print("Fido tricks : ", d.tricks)
print("Buddy tricks: ", e.tricks)
```

Output :-



```
Fido tricks :  ['roll over']
Buddy tricks:  ['play dead']
```

Assignment 8

```
class Employee:
    @classmethod
    def from_string(cls,emp_str):
    <Write logic here>

    def__init__(self,first,last,pay):
        self.firstname = first
        self.lastname = last
        self.pay = pay

emp_1_str = 'John-Abraham-50000'
emp_1 = Employee.from_string(emp_1_str)        #stmt1
```

Write logic for from_stringmethod such that it becomes alternative constructor; meaning it should create an object of Employee at #stmt1 with first name last name and pay values from emp_1_str string

Solution

```
class Employee:
    @classmethod
    def from_string(cls,emp_str):
        first, last, pay = emp_str.split('-')
        return cls(first, last, pay)
    def___init__(self,first,last,pay):
        self.firstname = first
        self.lastname = last
        self.pay = pay

emp_1_str = 'John-Abraham-50000'
emp_1 = Employee.from_string(emp_1_str)        #stmt1
```

Assignment 9

```
class Store:
    __item_count = 100

    #adds to count to___item_count
    def addItem

    #subtracts count from___item_count
    def issueItem

    #returns___item_count
    def getItemCount
```

***Requirement**: Both the counters (counter1 & counter2) in following code, access the same___item_count from Store. User can get the number of items in store by calling getItemCount method.*

```
counter1 = Store()
counter2 = Store()
#add 2 items to store from counter1
#issue 1 item at counter1
#getItemCount in the Store
```

Q1: Provide body for the 3 methods. Logic is as follows:

1) ***addItem (count):__*** *item_count += count*
2) ***issueItem (count):__*** *item_count -= count*
3) ***getItemCount(): returns__*** *item_count*

Q2: Justify method type (instance or static or class) for each method. Test your logic for above requirement

Solution

1)
```
@classmethod
def addItem(cls,count):
    cls.__item_count += count

@classmethod
def issueItem(cls,count):
    cls.__item_count -= count

@classmethod
def getItemCount(cls):
    return cls.__item_count
```

2) All three will be class method since they are accessing class variables __item_count

Assignment 10

```
class C:
    y = 5
    def__init__(self,a):
        self.x = a
    def m1(self):
        self.m2()        #No Error
        print(self.y)

    @classmethod
    def m2(cls):
        m1()             #Error
        self.x           #Error
```

Why objects can access class methods and class variables but class methods can not access instance methods or variables?

Solution :-

Because objects shares class variables among them and is same for every object thus can access class methods and variables
whereas
Instance variables and methods are seperate for seperate instances or objects and are meaningless without them thus class cannot access instance methods or variables

Assignment 11

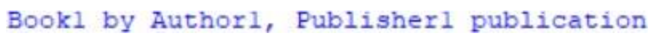
The title, author and publisher information make key representations for a book.

- *print(b) should print these key information about "b" which is an object of a Book class. Create class Book.*

Solution

```
class Book:

    def___init__(self, title, author, publisher):
        self.title = title
        self.author = author
        self.publisher = publisher

    def___str__(self):
        return self.title + " by " + self.author + ", " + self.publisher + "
publication"


b = Book("Book1", "Author1", "Publisher1")
print(b)
```

<u>Output :-</u>



```
Book1 by Author1, Publisher1 publication
```

Assignment 12

- *A class Calculatorhas two methods*
  - *getNextPrime() returns next primary number every time the function is called. For example, when first time invoked it returns 2 then 3,5,7... so on.*
  - *isPrime(num) returns true if the argument "num" is prime.*
- *Create the class with above functions.*
- *Test your code with a loop that calls the getNextPrimeand prints first 50 prime numbers.*

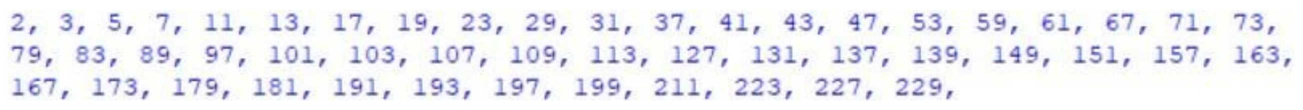
Hint :
1.Create a variable "self.lastprime" inside___init___method to store last prime delivered.
2.First write isPrime function, test if it works fine then write logic for

getNextPrime
3. Make call to isPrime from within getNextPrime function to avoid code duplication

Solution

```
class Calculator:
    last_prime = 1

    @staticmethod
    def isPrime(num):
        if(num <= 1):
            return False
        for i in range(2,num):
           if(num % i == 0):
                return False
        return True

    @classmethod
    def getNextPrime(cls):
        i = cls.last_prime + 1
        while(not cls.isPrime(i)):
            i += 1
        cls.last_prime = i
        return i

for i in range (0, 50):             # Will run for i = 0 to 49 i.e 50 times
    print(Calculator.getNextPrime(), end=", ")
```

<u>*Output :-*</u>



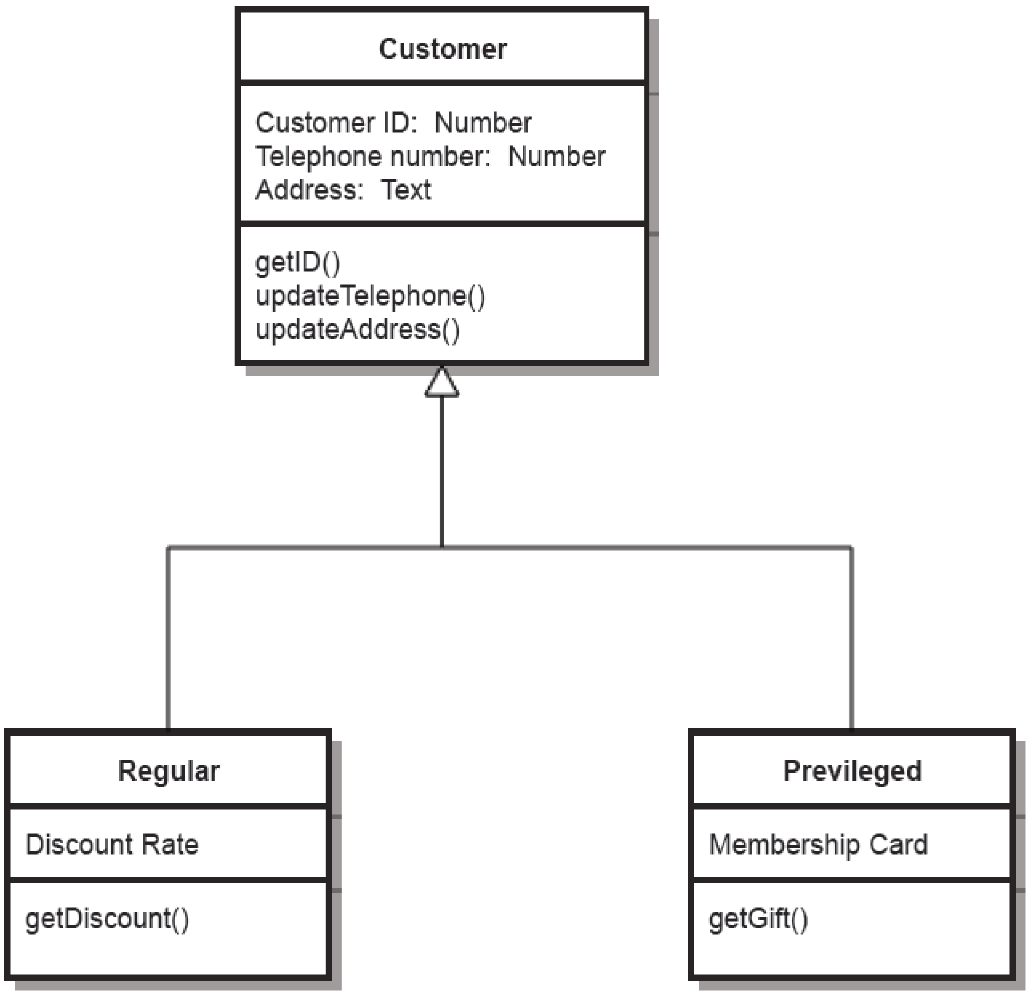
```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,
167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
```

Assignment 13

Identify classes, and relationship among them in following retail scenario:

- *All customers have Customer Id, Name, Telephone Number and Address*
- *The regular customer in addition is given discounts*
- *The privileged customer gets a membership card based on which gifts are given*

Solution



Assignment 14

class Box:

```
    def getVolume(self):
        vol= self.length*self.breadth* self.height
        return vol
    def___init__(self, length, breadth,height):
        self.length = length
        self.breadth = breadth
        self.height = height
class BigBox(Box):
    def___init__(self, length, breadth,height):
        Box.__init__(self,length,breadth,height)

    def getCapacity(self, sBox):
<<write logic here>>
```

•*Requirement:*

- •*A big box can contain small boxes*

.Q: Write logic for function getCapacity(self,sBox) in BigBox that returns capacity(number of small boxes it can contain).

- ➢*Formula for Capacity = BigBox volume / Small box volume.*

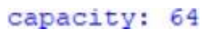
•*Test your code with following:*

```
smallBox = Box(1,1,1)
bigBox = BigBox (4,4,4)
capacity = bigBox.getCapacity(smallBox)
print("capacity:",capacity)
```

Solution

```
class Box:
    def getVolume(self):
        vol= self.length *self.breadth * self.height
        return vol

    def___init__(self, length, breadth, height):
        self.length = length
        self.breadth = breadth
        self.height = height

class BigBox(Box):
    def___init__(self, length, breadth,height):
        Box.__init__(self,length,breadth,height)

    def getCapacity(self, sBox):
        return int(self.getVolume()/sBox.getVolume())

smallBox = Box(1,1,1)
bigBox = BigBox (4,4,4)
capacity = bigBox.getCapacity(smallBox)
print("capacity:",capacity)
```

<u>Output :-</u>



```
capacity: 64
```

Assignment 15

•An animation film is in making. Following human actors are created in the animation film:(1) TennisPlayer (2) Professor (3) ShopKeeper (4) Carpenter
•This animation is coded as software. These actors are modeled as class entities.
•The film was so far silent, however a new method "talk" needs to be added to these entities.
Question : Which class(es) will be appropriate to add talk() method so that all actors can talk? Consider following before arriving at optimal solution:
•this functionality may undergo changes several times before final implementation.
•"sing" is next method to be added to these entities in near future.
•A Principal and a Pilot are two human actors to be added to the software

model in next version.

Solution

It will be best to create an actor class (if not already created) as superclass for TennisPlayer, Professor, Shopkeeper, Carpenter.
This way

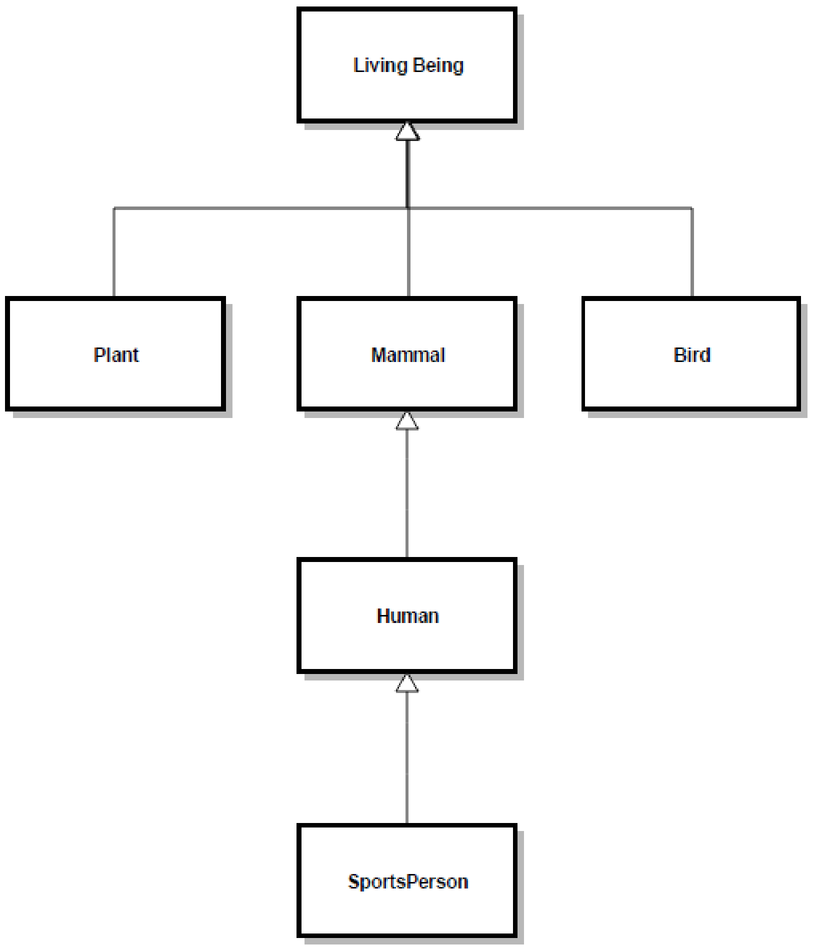
- Changes can be made several times without any problems
- Sing method can also be added to actors later
- Principal and Pilot can added as an subclass whenever required easily without any problem

Assignment 16

Identify relationship between following entities. Depict the same with UML class diagram

•*SportsPerson*
•*Human*
•*Mammal*
•*LivingBeing*
•*Plant*
•*Bird*

Solution



Assignment 17

•*Q1: Identify classes and their relationships; depict with UML class diagrams for following :*

•*External examiners and visiting faculties are visitors.*

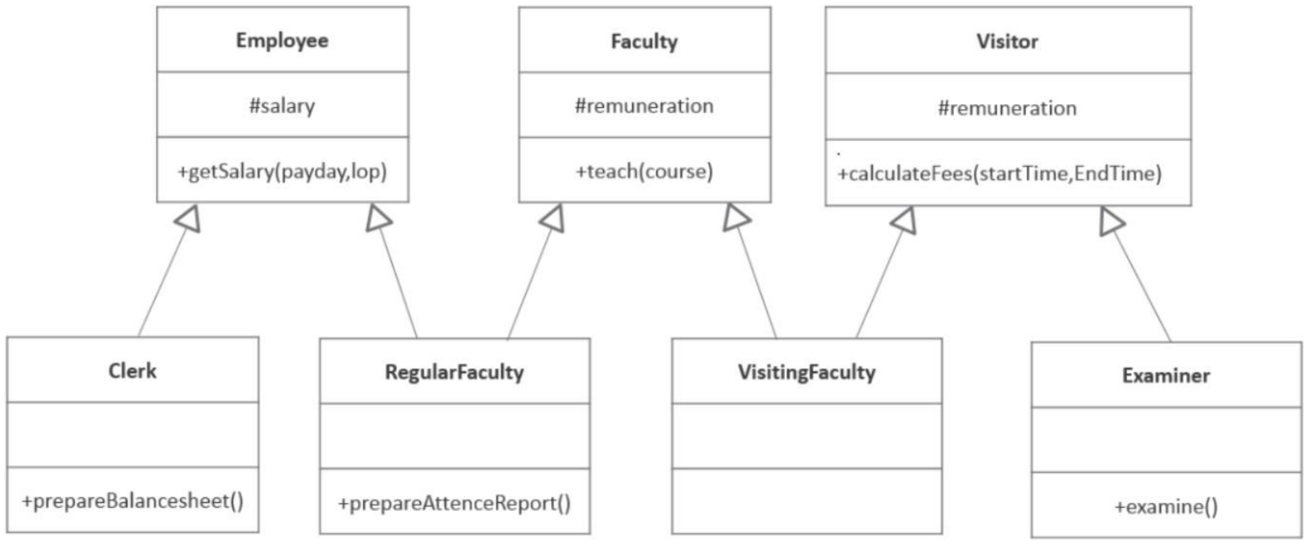
•*All visitors are paid remuneration. The fees is calculated based on number of hours (end time – start time)*

•*Examiners examine.*

•*Clerks and regular faculties are the employees working on payroll of institution (salary).*

•All employees get salary which is calculated based on pay days (calendar days - LOP(loss of pay))
•Clerks prepare balance sheet.
•A Faculty can be a Regular faculty from the same institute or a visiting faculty.
•Faculties teach one or more courses.
•Regular faculties prepare attendance report.
•Q2: Create classes in Python
•You may write "pass" for method bodies implementation, since no logic is provided

Solution



Code :-

```
class Employee:

    def___init__(self, salary):
        self.salary = salary

    def getSalary(payday, lop):
        pass

class Faculty:

    def___init__(self, remuneration):
        self.remuneration = remuneration

    def teach(course):
        pass

class Visitor:

    def___init__(self, remuneration):
        self.remuneration = remuneration

    def calculateFees(startTime, EndTime):
        pass

class Clerk(Employee):

    def preparedBalancesheet():
        pass

class RegularFaculty(Employee, Faculty):

    def prepareAttendanceReport():
        pass

class VisitingFaculty(Faculty, Visitor):
    pass

class Examiner(Visitor):

    def examine():
        pass
```

Assignment 18:

•In a retail outlet there are two modes of bill Payment
•Cash : Calculation includes VAT(15%)
Total Amount = Purchase amount + VAT
•Credit card: Calculation includes processing charge and VAT
Total Amount = Purchase amount + VAT(15%)+ Processing charge(2%)

The act of bill payment is same but the formula used for calculation of total amount differs as per the mode of payment.

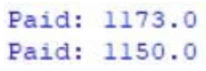
Q: Can the Payment maker simply call a method and that method dynamically selects the formula for the total amount? Demonstrate this Polymorphic behavior with code

Solution

```
from abc import ABC, abstractmethod

class Payment(ABC):

    VAT = 1.15

    @abstractmethod
    def getTotalAmount(self):
        pass

class CreditCardPayment(Payment):

    processingCharges = 1.02
    def getTotalAmount(self,purchaseAmt):
        amt = purchaseAmt * self.VAT #stmt1
        amt = amt * self.processingCharges
        return amt
```

```
class CashPayment(Payment):
    def getTotalAmount(self,purchaseAmt):
        return (purchaseAmt * self.VAT) #stmt2

class Bill:
    def___init__(self,purchaseAmount):
        self.__purchaseAmount = purchaseAmount

    def makePayment(self,mode):
        #Ensure that it is a valid mode of payment
        if (isinstance(mode, Payment)):
            #actual behavior is selected dynamically
            amount= mode.getTotalAmount(self.__purchaseAmount)
            print("Paid:",amount)

#create a bill with
#purchaseAmount=1000
bill = Bill(1000)
cc = CreditCardPayment()
bill.makePayment(cc)
cash = CashPayment()
bill.makePayment(cash)
```

Output :-



```
Paid: 1173.0
Paid: 1150.0
```

Assignment 19

What kind of relationship exists between car and its components given below:
•Engine
•Tool kit
•DVD Player
•DVD

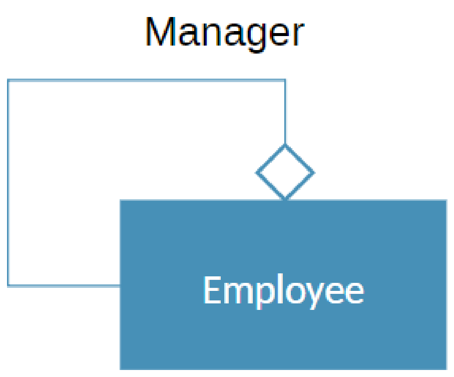
Solution

Car <has-a:composition> Engine
Car <uses-a> Toolkit
Car <has-a:aggregation> DVD Player
Car <uses-a> DVD

Assignment 20

•In an organization for an employee to be a manager, he/she must be an employee of the same organization.
•Model this relationship in UML class diagram

Solution

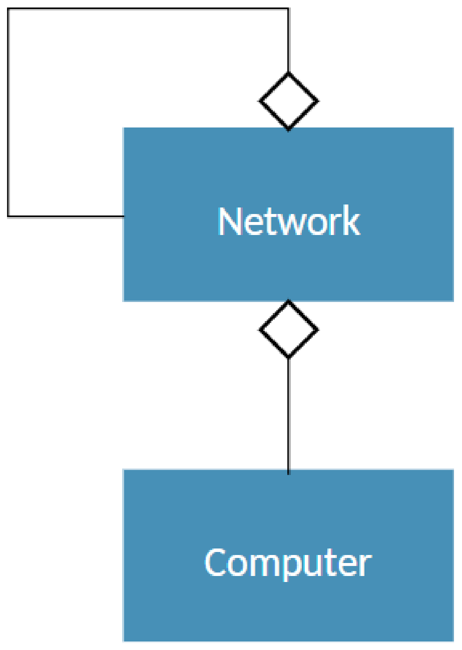


Assignment 21

•Internet is network of networks and computers
Q - Model Internet using UML diagram

Solution

Internet ->



Assignment 22

Q1: Identify classes and their relationships; depict relationship with UML class diagrams for following :

•An institution has multiple departments and many employees.•Every department offer multiple courses.

•All the regular faculties are the employees who belong to respective departments.

•Regular faculty teach one or more course.

•Visitors (Experts) are invited for a certain event.

Q2: Consider following details, improvise the class diagram and write Python code:

•The institution has a registered name. Every department has a unique name.

•Courses are identified by course name. A department can add new course anytime.

•Visitors are given an auto-generated visitor id at the time of their entrance.

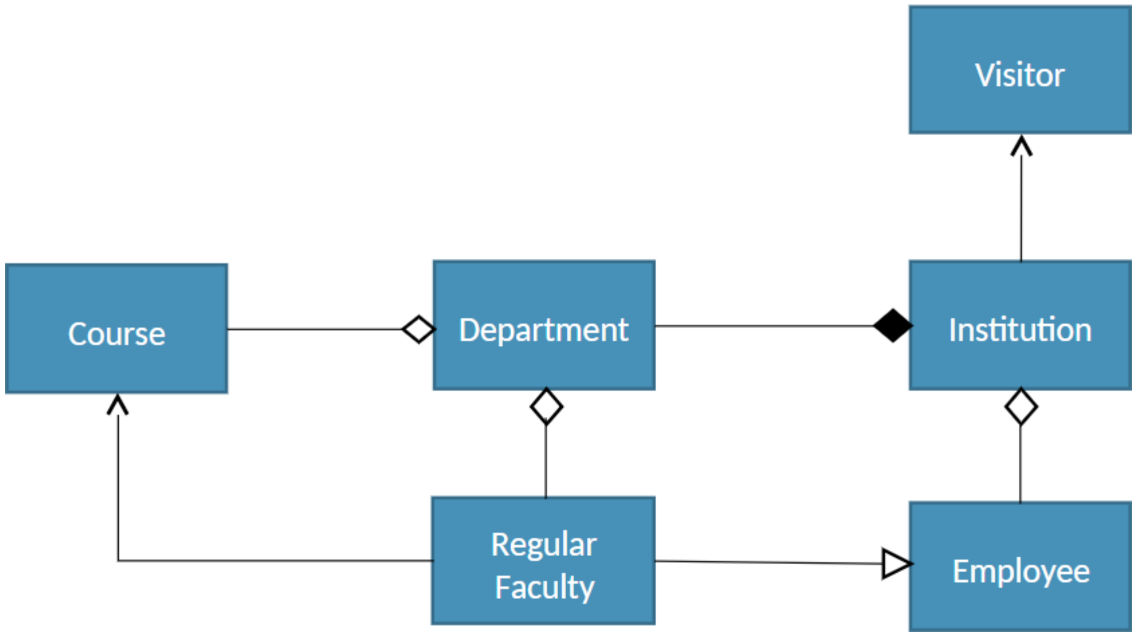
•New employees can be recruited in the institution. Name & Designation are recorded and a unique id is generated for the employee at the time of joining. The designation may change during service tenure.

•You may add getter (getXXX e.g. getName , getId), setter (setDesignation, ...),___str___(to print string equivalent of object) methods to the classes.

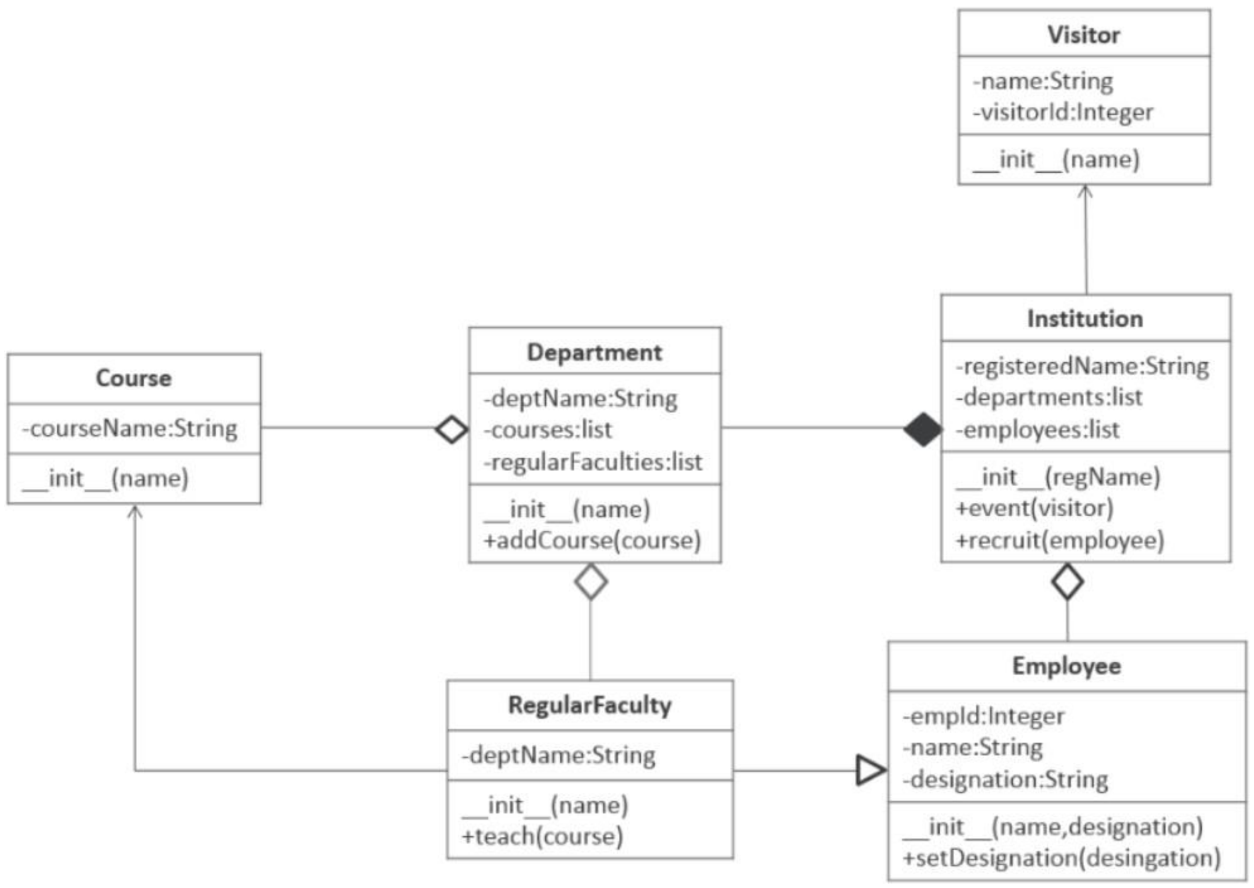
•Detailed method logic is not expected at this moment , however make sure that appropriate input parameters are written in the method signature.. You may write "pass" in the body of the method in place of logic.

Solution

1)


Visitor
Course
Department
Institution
Regular Faculty
Employee

2)



Code :

```
class Course:

    def___init__(self, courseName):
        self.courseName = courseName


class Visitor:

    visitorId = 1

    def___init__(self, name):
        self.__visitorId = Visitor.visitorId
        Visitor.visitorId += 1
        self.__name = name
```

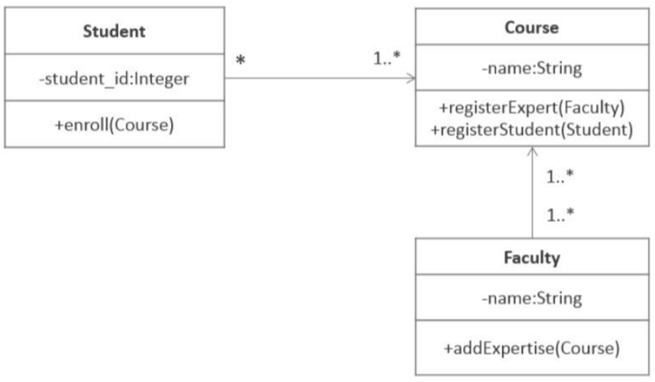
```
class Departmaent:

    def___init__(self, name):
        self.__deptName = name
        self.__courses = []
        self.__regularFaculties = []

    def addCourse(self, course):
        self.__courses.append(course)


class Employee:

    empId = 100

    def___init__(self, name, designation):
        Employee.empId += 1
        self.__empId = Employee.empId
        self.__name = name
        self.__designation = designation

    def setdesignation(self, designation):
        self.__designation = designation


class RegularFaculty(Employee):

    def___init__(self, name):
        self.__deptName = name

    def teach(course)
        pass

class Institution:
```

```
def___init__(regName):
    self.__registeredName = regName
    self.__departments = []
    self.__employees = []

def event(visitor):
    pass

def recruit(employee):
    self.__employees.append(employee)
```

Assignment 23

Q:(1) Model following scenario using class diagram and (2) Create Python classes for the same.
•The students can enroll for multiple courses.
•A course can be taught by one or more faculties.
•Every faculty has expertise to teach one or more courses. A faculty can acquire new expertise.
•Student is identified by student id, course is identified by course name and a faculty is identified by faculty name.

Solution

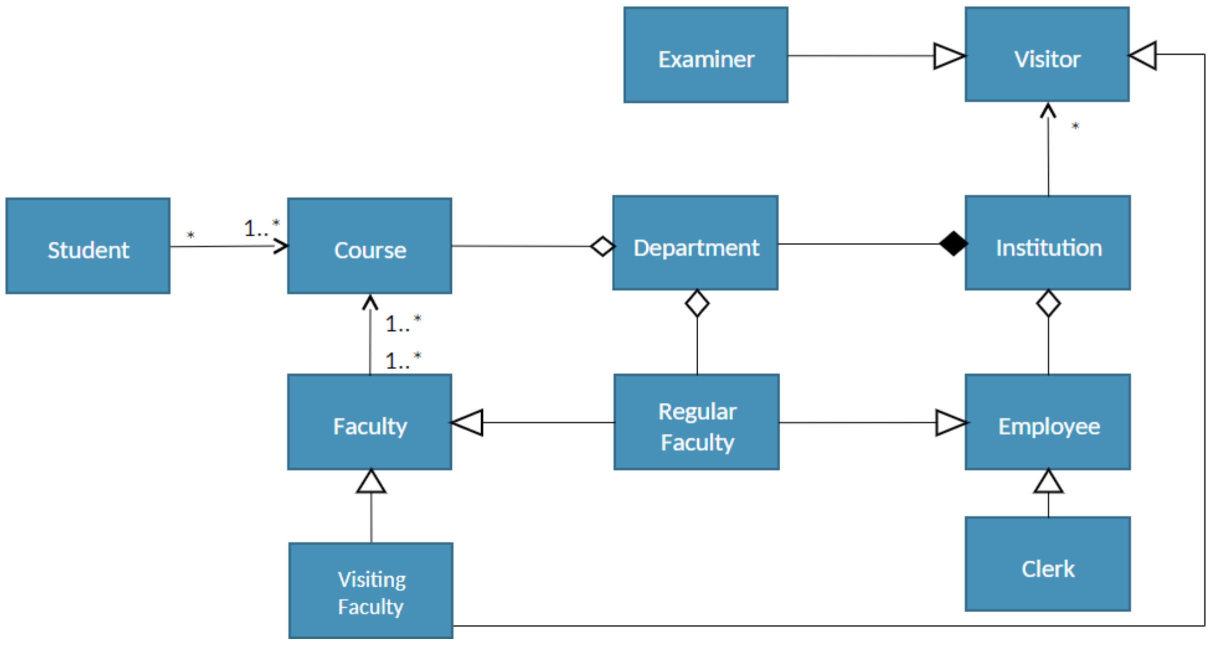


```
class Faculty:
    def addExpertise(self,course):
        self.__coursesExpertise.append(course)

    def___init__(self,name):
        self.__name = name
        self.__coursesExpertise=[]

class Student:
    def enroll(self,course):
        self.__enrolledCourses.append(course)

    def___init__(self,student_id):
        self.__id = student_id
        self.__enrolledCourses=[]

class Course:
    def registerStudent(self,student):
        self.__registeredStudents.append(student)

    def registerExpert(self,faculty):
        self.__expertFaculties.append(faculty)

    def___init__(self,name):
        self.__name=name
        self.__registeredStudents=[]
        self.__expertFaculties=[]
```

Assignment 24

Q: Identify entities/classes and their relationships; depict with UML class diagrams.
•An institution has multiple departments and many employees.
•Every department offer multiple courses.
•Students can enroll for multiple courses.
•Every faculty has expertise to teach one or more courses.
•All regular faculties of the institution belong to respective departments
•A course can be taught by one or more faculties
•External examiners and visiting faculties are the visitors who are invited for a certain event.
•Clerks, regular faculties are employees working on payroll of institution.
•A Faculty can be a Regular faculty from the same institute or a visiting faculty

Solution



Assignment 25

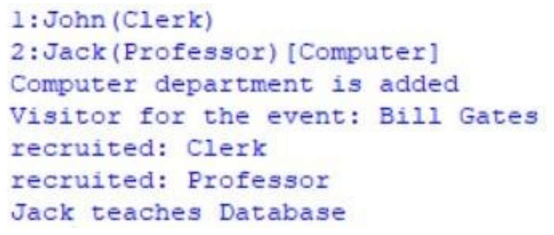
Write implementation logic for methods created in Assignment 22.
Test the same for following expected input/output:
1.d1 = Department("Computer") #create "computer" department
2.e1 = Employee("John", "Clerk") #create an employee "John" with designation "Clerk"
3.# Employee.__str__() should return:"<employee id>:<name>(<designation>)"
4.print(e1) #expected output=> "1:John(Clerk)"
5.#create regular faculty "Jack", a professor in "computer" department.
6.e2 = RegularFaculty("Jack",d1,"Professor")
7.#RegularFaculty.__str__() should return: "<employee id>:<name>(<designation>)[<department>]"
8.print(e2) # expected output=> "2:Jack(Professor)[Computer]"
9.v1 = Visitor("Bill Gates") #create a visitor with name "Bill Gates"
11.#create an Institute with registered name "Institute of Technology"
12.inst = Institution("Institute of Technology")
13.#Institution.addDepartment(department)method should print: "<department name> department is added"
14.inst.addDepartment(d1) # expected output=> "Computer department is added"
15.#Institution.event(visitor) method should print: "Visitor for the event: <visitor name>"
16.inst.event(v1) #expected output => "Visitor for the event: Bill Gates"
17.#Institution.recruit(employee)method should print=> "recruited: <employee designation>"
18.inst.recruit(e1) # expected output => "recruited: Clerk"
19.inst.recruit(e2) # expected output => "recruited: Professor"
20.c = Course("Database") #create a course for course name "Database"
21.#RegularFaculty.teach(Course) method should print "<faculty name> teaches <course name>"
22.e2.teach(c) # expected output => "Jack teaches Database"

Solution

```
class Course:

    def __init__(self, courseName):
        self.courseName = courseName

    def __str__(self):
        return self.courseName

class Visitor:

    visitorId = 1

    def __init__(self, name):
        self.__visitorId = Visitor.visitorId
        Visitor.visitorId += 1
        self.__name = name

    def __str__(self):
        return self.__name


class Department:

    def __init__(self, name):
        self.__deptName = name
        self.__courses = []
        self.__regularFaculties = []

    def addCourse(self, course):
        self.__courses.append(course)

    def getName(self):
        return self.__deptName
    def__str__(self):
        return self.__deptName


class Employee:
    empId = 0

    def__init__(self, name, designation):
        Employee.empId += 1
        self.__empId = Employee.empId
        self.name = name
        self.designation = designation

    def setDesignation(self, designation):
        self.designation = designation

    def getDesignation(self):
        return self.designation

    def__str__(self):
        return str(self.__empId) + ":" + self.name + "(" + self.designation+")"


class RegularFaculty(Employee):

    def__init__(self, name, department, designation):
        self.__deptName = department.getName()
        Employee.__init__(self, name, designation)

    def teach(self, course):
        print(self.name + " teaches " + str(course))

    def__str__(self):
        return str(self.empId) + ":" + self.name + "(" + self.designation+")[" +
self.__deptName + "]"
```

```
class Institution:

    def __init__(self, regName):
        self.__registeredName = regName
        self.__departments = []
        self.__employees = []

    def event(self, visitor):
        print("Visitor for the event: " + str(visitor))

    def recruit(self, employee):
        self.__employees.append(employee)
        print ("recruited: " + employee.getDesignation())

    def addDepartment(self, department):
        self.__departments.append(department)
        print(str(department) + " department is added")



d1 = Department("Computer")
e1 = Employee("John", "Clerk")
print(e1)
e2 = RegularFaculty("Jack", d1, "Professor")
print(e2)
v1 = Visitor("Bill Gates")
inst = Institution("Institute of Technology")
inst.addDepartment(d1)
inst.event(v1)
inst.recruit(e1)
inst.recruit(e2)
c = Course("Database")
e2.teach(c)
```

Output :-


```
1:John(Clerk)
2:Jack(Professor)[Computer]
Computer department is added
Visitor for the event: Bill Gates
recruited: Clerk
recruited: Professor
Jack teaches Database
```

Assignment 26

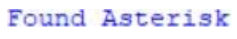
*Create function asteriskChecker(myString)such that the method raises user defined InvalidStringexception if an asterisk (*) is found. The #statement1 should print "Found Asterisk" if the input string contains *.*

```
mymessage="abcde*fz"
try:
    asteriskChecker(mymessage)
except InvalidString as e :
    print(e) #statement1
else:
    print("String has no Asterisk ")
```

Solution

```
class InvalidString(Exception):
    def__init__(self):
        Exception.__init__(self,"Found Asterisk")

def asteriskChecker(myString):
    for i in myString:
        if(i=="*"):
            raise InvalidString() #statement1
```

Output :-


```
Found Asterisk
```

Assignment 27

•*Write a function that*
  •*opens a file*
  •*Reads and prints the existing content.*
  •*Write a statement "hello" in the file.*

•*After the file is successfully opened, if the file read/ write operation results in an exception for any reason (like file is read only and user is not authorized to write, some OS error etc.), then ensure that the file is closed before the exit from the function.*

Solution

```
file = open("somefile.txt", "w")
try:
    lines = file.readlines()
    for line in lines:
        print(line)
    file.write("hello")
except IOError as e:
    print(e):
finally:
    file.close()
```

Assignment 28

•*Take email id, mobile number and age as inputs from user*
•*Validate each and raise user defined exceptions accordingly*
•*Note:*
  -Email id :
    •*there must be only one @*

•*At least one "."*

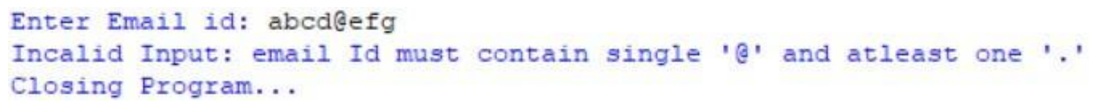
-*Mobile number must be 10 digits and it can start with + symbol*

-*Age must be a positive number less than 101*

Solution

```
class InvalidInput(Exception):
    def__init__(self,msg):
        self.msg = msg

    def__str__(self):
        return ("Incalid Input: " + self.msg)

try:
    email_id = input("Enter Email id: ")
    if(email_id.count("@") != 1 or email_id.count(".")==0):
        raise InvalidInput("email Id must contain single '@' and atleast one '.'")
    mob_num = input("Enter Mobile Number: ")
    if(not mob_num.isnumeric()):
        raise InvalidInput("Mobile Number must be integer")
    if(int(mob_num) < 1000000000 or int(mob_num) > 9999999999):
        raise InvalidInput("Invalid mobile number must be 10 digit")
    age = input("Enter age: ")
    if(not age.isnumeric()):
        raise InvalidInput("Invalid age must be integer")
    if(int(age) <= 0 or int(age) >= 101):
        raise InvalidInput("Invalid age must be positve and less than 101")
except InvalidInput as e:
    print(e)
else :
    print("All Valid Input")
finally:
    print("Closing Program...")
```

__Output :-__



```
Enter Email id: abcd@efg
Incalid Input: email Id must contain single '@' and atleast one '.'
Closing Program...
```

Assignment 29

Requirement from a function getDiscount(age) is given bellow. This function is supposed to return the percentage discount figure for the given input age value of customer in years.

•If the age of customer is less than 60, No discount..

•If age is 60 to 70 years then the discount should be 15%.

•Discount should be 30% for age greater than and equal to 70 years.

Complete the following table for boundary value test case.

Solution

Test Cases	Description	Expected discount
#1	Customer age less than 60	0
#2	Customer age equal to 60	15
#3	Customer age is 61	15
#4	Customer age is 69	15
#5	Customer age equal to 70	30
#6	Customer age above 70	30

Assignment 30

```
def getDiscount(age):
    discount = 0
    if age > 60 and age < 70:
        discount = 15
    elif age > 70:
        discount = 30
    return discount

myAge = int(input("Enter Age:"))
myDiscount = getDiscount(myAge)
print("Discount percent=",myDiscount)
```

•A programmer has written this code for getDiscount(age) function for the requirement given in previous assignment:
1.Write and Execute test cases. Use the below given table to note the details.
2."Actual discount": Write actual discount you get after execution of every test case.
3.Compare the actual discount with expected discount. If they match, write "pass" else "fail" for every test case in Result column
4.Execute the unit test cases for getDiscount using PyUnit

Solution

Test Case	Description/ Input Values	Expected Discount	Actual Discount	Result (Pass/Fail)
#1	age < 60	0	0	Pass
#2	age = 60	15	0	Fail
#3	age = 61	15	15	Pass
#4	age = 69	15	15	Pass
#5	age = 70	30	0	Fail
#6	age>70	30	30	Pass

Assignment 31

```
def getDiscount(age,gender):
    discount = 0
    if age >= 60:
        if gender == 'F':
            discount = 25
        discount = 20
    elif gender == 'F':
        discount = 15
    return discount
age = int(input("Enter age:"))
gender = input("Enter Gender as M or F:")
discount = getDiscount(age, gender)
print("discount",discount)
```

•A programmer has written this code for function getDiscount(age,gender), for the requirement given below :

•Non Senior (age<60) female = 15% discount;

•Senior citizen: Female=25%, Male=20%

1.Write and Execute test cases which covers all the paths of execution. Use the below given table to note the details.

2.Compare the actual discount with expected discount. If they match, write "pass" else "fail" for every test case in Result column

3.Execute the unit test cases for getDiscountfunction using PyUnit

Solution

Test Case	**Description/ Input Values**	**Expected Discount**	**Actual Discount**	**Result (Pass/Fail)**
#1	age < 60 gender = F	15	15	Pass

#2	age = 60
	gender = F
		25	25	Pass

#3	age > 60 gender = F	25	25	Pass
#4	age <60 gender = M	0	0	Pass

#5	age = 60
	gender = M
		20	20	Pass

#6	age > 60 gender = M	20	20	Pass

Assignment 32

Q: Write and Execute test cases for following algorithm

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)
```

Solution

Test Case	Description/ Input Values	Expected Output	Actual Output	Result (Pass/Fail)
#1	Mixed	Sorted in ascending order	Sorted in ascending order	Pass
#2	Descending order	Sorted in ascending order	Sorted in ascending order	Pass
#3	Ascending Order	Sorted in ascending order	Sorted in ascending order	Pass