Inspiring Excellence

Course Code:	**CSE111**
Course Title:	**Programming Language II**
Lab No:	**10**
Topic:	**OOP (Inheritance)**
Number of tasks:	**5 Classwork + 4 Homework**

*** You are not allowed to change any of the code of the tasks*

*** Use **Inheritance** to solve all problems*

Classwork Part

Task - 1

Given the following classes, write the code for the **BBA_Student** class so that the following output is printed:

```
class Student:
    def __init__(self, name='Just a student', dept='nothing'):
        self.__name = name
        self.__department = dept
    def set_department(self, dept):
        self.__department = dept
    def get_name(self):
        return self.__name
    def set_name(self,name):
        self.__name = name
    def detail(self):
        return 'Name: '+self.__name+' Department: '+self.__department

#write your code here

print(BBA_Student().detail())
print(BBA_Student('Humpty Dumpty').detail())
print(BBA_Student('Little Bo Peep').detail())
```

Output:

Name: default Department: BBA
Name: Humpty Dumpty Department: BBA
Name: Little Bo Peep Department: BBA

Task – 2

```
class Vehicle:
    def __init__(self):
        self.x = 0
        self.y = 0
    def moveUp(self):
        self.y += 1
    def moveDown(self):
        self.y -= 1
    def moveRight(self):
        self.x += 1
    def moveLeft(self):
        self.x -= 1
    def detail(self):
        return '('+str(self.x)+' , '+str(self.y)+')'
#write your code here

print('Part 1')
print('------')
car = Vehicle()
print(car.detail())
car.moveUp()
print(car.detail())
car.moveLeft()
print(car.detail())
car.moveDown()
print(car.detail())
car.moveRight()
print(car.detail())
print('------')
print('Part 2')
print('------')
car1 = Vehicle2010()
print(car1.detail())
car1.moveLowerLeft()
print(car1.detail())
car2 = Vehicle2010()
car2.moveLeft()
print(car1.equals(car2))
car2.moveDown()
print(car1.equals(car2))
```

OUTPUT:

```
Part 1
------
(0 , 0)
(0 , 1)
(-1 , 1)
(-1 , 0)
(0 , 0)
------
Part 2
------
(0 , 0)
(-1 , -1)
False
True
```

A vehicle assumes that the whole world is a 2-dimensional graph paper. It maintains its x and y coordinates (both are integers). The vehicle gets manufactured (constructed) at (0, 0) coordinate.

Subtasks:

1. Design a **Vehicle2010 class** that inherits movement methods from **Vehicle** and adds new methods called **move UpperRight, UpperLeft, LowerRight, LowerLeft.** Each of these diagonal move methods must re-use two inherited and appropriate move methods.
2. Write an "**equals"** method that tests if significant class properties are the same (in this case x and y).

Note: All moves are 1 step. That means a single call to any move method changes value of either x or y or both by 1.

Task - 3

Given the following classes, write the code for the **Cricket_Tournament** and the **Tennis_Tournment** class so that the following output is printed.

```
class Tournament:
    def __init__(self,name='Default'):
        self.__name = name
    def set_name(self,name):
        self.__name = name
    def get_name(self):
        return self.__name

#write your code here

ct1 = Cricket_Tournament()
print(ct1.detail())
print("-----------------------")
ct2 = Cricket_Tournament("IPL",10,"t20")
print(ct2.detail())
print("-----------------------")
tt = Tennis_Tournament("Roland Garros",128)
print(tt.detail())
```

OUTPUT:

```
Cricket Tournament Name: Default
Number of Teams: 0
Type: No type
-----------------------
Cricket Tournament Name: IPL
Number of Teams: 10
Type: t20
-----------------------
Tennis Tournament Name: Roland Garros
Number of Players: 128
```

Task - 4

Given the following classes, write the code for the **Book** and the **CD** class so that the following output is printed.

```
class Product:
    def __init__(self,id, title, price):
        self.__id = id
        self.__title = title
        self.__price = price
    def get_id_title_price(self):
        return "ID: "+str(self.__id)+" Title:"+self.__title+
"Price: "+str(self.__price)

#write your code here

book = Book(1,"The Alchemist",500,"97806","HarperCollins")
print(book.printDetail())
print("----------------------")
cd = CD(2,"Shotto",300,"Warfaze",50,"Hard Rock")
print(cd.printDetail())
```

OUTPUT:

```
ID: 1 Title: The Alchemist Price: 500
ISBN: 97806 Publisher: HarperCollins
----------------------
ID: 2 Title: Shotto Price: 300
Band: Warfaze Duration: 50 minutes
Genre: Hard Rock
```

Task - 5

Given the following classes, write the code for the **Dog** and the **Cat** class so that the following output is printed.

```
class Animal:
    def __init__(self,sound):
        self.__sound = sound

    def makeSound(self):
        return self.__sound


class Printer:
    def printSound(self, a):
        print(a.makeSound())

#write your code here

d1 = Dog('bark')
c1 = Cat('meow')
a1 = Animal('Animal does not make sound')
pr = Printer()
pr.printSound(a1)
pr.printSound(c1)
pr.printSound(d1)
```

OUTPUT:

Animal does not make sound
meow
bark

Homework Part

Task - 1

Given the following classes, write the code for the **Triangle** and the **Trapezoid** class so that the following output is printed.

```
class Shape:

  def __init__(self, name='Default', height=0, base=0):
    self.area = 0
    self.name = name
    self.height = height
    self.base = base

  def get_height_base(self):
    return "Height: "+str(self.height)+", Base: "+str(self.base)

#write your code here

tri_default = triangle()
tri_default.calcArea()
print(tri_default.printDetail())
print('--------------------------')
tri = triangle('Triangle', 10, 5)
tri.calcArea()
print(tri.printDetail())
print('--------------------------')
trap = trapezoid('Trapezoid', 10, 6, 4)
trap.calcArea()
print(trap.printDetail())
```

OUTPUT:

```
Shape name: Default
Height: 0, Base: 0
Area: 0.0
--------------------------
Shape name: Triangle
Height: 10, Base: 5
Area: 25.0
--------------------------
Shape name: Trapezoid
Height: 10, Base: 6, Side_A: 4
Area: 50.0
```

Task - 2

Given the following classes, write the code for the **Player** and the **Manager** class so that the following output is printed. To calculate the match earnings use the following formula:

1. Player: (total_goal * 1000) + (total_match * 10)
2. Manager: match_win * 1000

```
class SportsPerson:

  def __init__(self, team_name, name, role):
    self.__team = team_name
    self.__name = name
    self.role = role
    self.earning_per_match = 0

  def get_name_team(self):
    return 'Name: '+self.__name+', Team Name: ' +self.__team

#write your code here

player_one = Player('Al-Nassr', 'Ronaldo', 'Striker', 25, 32)
player_one.calculate_ratio()
player_one.print_details()
print('----------------------------------------')
manager_one = Manager('Real Madrid', 'Zidane', 'Manager', 25)
manager_one.print_details()
```

OUTPUT:

```
Name: Ronaldo, Team Name: Al-Nassr
Team Role: Striker
Total Goal: 25, Total Played: 32
Goal Ratio: 0.78125
Match Earning: 25320K
---------------------------------
Name: Zidane, Team Name: Real Madrid
Team Role: Manager
Total Win: 25
Match Earning: 25000K
```

Task - 3

The tea company **Kazi and Kazi (KK)** has decided to produce a new line of flavored teas. Design the **KK_tea (parent) and KK_flavoured_tea (child)** classes so that the following output is produced. The KK_flavoured_tea class should inherit KK_tea. Note that:

- An object of either class represents a **single box of teabags**.
- Each tea bag **weighs 2 grams**.
- The **status** of an object refers to whether it is sold or not

Hint: you should use class methods/variables

```
t1 = KK_tea(250)
print("-----------------1-----------------")
t1.product_detail()
print("-----------------2-----------------")
KK_tea.total_sales()
print("-----------------3-----------------")
t2 = KK_tea(470, 100)
t3 = KK_tea(360, 75)
KK_tea.update_sold_status_regular(t1, t2, t3)
print("-----------------4-----------------")
t3.product_detail()
print("-----------------5-----------------")
KK_tea.total_sales()
print("-----------------6-----------------")
t4 = KK_flavoured_tea("Jasmine", 260, 50)
t5 = KK_flavoured_tea("Honey Lemon", 270, 45)
t6 = KK_flavoured_tea("Honey Lemon", 270, 45)
print("-----------------7-----------------")
t4.product_detail()
print("-----------------8-----------------")
t6.product_detail()
print("-----------------9-----------------")
KK_flavoured_tea.update_sold_status_flavoured(t4,
t5, t6)
print("-----------------10----------------")
KK_tea.total_sales()
```

OUTPUT:

```
-----------------1-----------------
Name: KK Regular Tea, Weight: 100
Tea Bags: 50, Price: 250
Status: False
-----------------2-----------------
Total sales: {'KK Regular Tea': 0}
-----------------3-----------------
-----------------4-----------------
Name: KK Regular Tea, Weight: 150
Tea Bags: 75, Price: 360
Status: True
-----------------5-----------------
Total sales: {'KK Regular Tea': 3}
-----------------6-----------------
-----------------7-----------------
Name: KK Jasmine Tea, Weight: 100
Tea Bags: 50, Price: 260
Status: False
-----------------8-----------------
Name: KK Honey Lemon Tea, Weight: 90
Tea Bags: 45, Price: 270
Status: False
-----------------9-----------------
-----------------10----------------
Total sales: {'KK Regular Tea': 3, 'KK
Jasmine Tea': 1, 'KK Honey Lemon Tea': 2}
```

Task - 4

Given a **TwoDVector** class, design the **ThreeDVector** class that inherits 2D vector. You need to implement the following features:

- Similar to X and Y of 2D vector, there will be Z of 3D vector.
- Write a method **add3DVectors()** that adds 3D vectors. It **must reuse** the **add2DVectors()** function and be written with the same parameters. The only difference is that, in 3D vectors, the Z components are added as well.
- Write a **multiplyScalar()** method that takes an integer as parameter and multiplies it with all 3 components separately (scalar multiplication). Keep in mind that the X and Y variables are private.
- Write a **calculateLength()** that returns the length of the 3D vector using the following formula:
  - $\sqrt{X^2 + Y^2 + Z^2}$
- Write a **print3DVector()** similar to the **print2DVector()** method.
- 2D vector: Xi + Yj
  3D vector: Xi + Yj + Zk

```
class TwoDVector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def add2DVectors(self, *vectors):
        for i in vectors:
            self.x += i.x
            self.y += i.y
    def print2DVector(self):
        if self.y >= 0:
            y = "+ "+str(self.y)
        else:
            y = str(self.y)
        print(f"{self.x}i {y}j")

TwoDV1 = TwoDVector(5, 6)
TwoDV2 = TwoDVector(3, 7)
TwoDV3 = TwoDVector(1, 8)
print("===============")
TwoDV1.add2DVectors(TwoDV2, TwoDV3)
TwoDV1.print2DVector()
print("===============")
ThreeDV1 = ThreeDVector(5, 6, 1)
ThreeDV2 = ThreeDVector(1, 9, -7)
```

OUTPUT:

```
===============
9i + 21j
===============
===============
14i + 17j -2k
===============
42i + 51j -6k
===============
66.34003316248794
```

`` ThreeDV3 = ThreeDVector(8, 2, 4) print("===============") ThreeDV1.add3DVectors(ThreeDV2,ThreeDV3) ThreeDV1.print3DVector() print("===============") ThreeDV1.multiplyScalar(3) ThreeDV1.print3DVector() print("===============") print(ThreeDV1.calculateLength()) ```	