BRAC UNIVERSITY

Inspiring Excellence

| | |
|---|---|
| **Course Code:** | **CSE111** |
| **Course Title:** | **Programming Language II** |
| **Homework No:** | **11** |
| **Topic:** | **Inheritance** |
| **Submission Type:** | **Will be notified later on.** |
| **Resources:** | **1. Class lectures**<br>**2. BuX lectures**<br>**a. English:**<br>**i. Inheritance: here**<br>**b. Supplementary:**<br>**i. Inheritance: here** |

## Task 1

A multinational company has two special types of regular employees. One is Foreign employees and another one is Part time employees. Design the Employee (parent), Foreign_employee(child) and Parttime_employee(child) classes so that the following output is produced. The Foreign_employee and Parttime_employee classes should inherit the Employee class. Note that:

- Basic salary of a Regular, Foreign employee is 30,000 and for Part-time employees basic is 15,000.
- Regular employees get 10% increment on their salary and Foreign employees get 15% increment on their basic salary.
- Employees from the HR department will collect their work distribution load from the manager, and others will collect their work distribution load from the HR department.

| Driver Code | Output |
|---|---|
| print("1----------------------------------------")<br>emp1=Employee("Nawaz Ali", 102, "Marketing")<br>print("2----------------------------------------")<br>emp1.employeeDetails()<br>print("3----------------------------------------")<br>emp1.workDistribution("Marketing")<br>print("4----------------------------------------")<br>emp1.increment()<br>emp1.employeeDetails()<br>print("5----------------------------------------")<br>f_emp=Foreign_employee("Nadvi", 311, "Human Resource")<br>f_emp.employeeDetails()<br>print("6----------------------------------------")<br>f_emp.workDistribution("Human Resource")<br>print("7----------------------------------------")<br>f_emp.increment()<br>f_emp.employeeDetails()<br>print("8----------------------------------------")<br>p1_emp=Part_time_employee("Asif", 210, "Sales")<br>p2_emp=Part_time_employee("Olive", 223, "Accounts")<br>print("9----------------------------------------")<br>p1_emp.employeeDetails()<br>print("10----------------------------------------")<br>p1_emp.workDistribution("Sales")<br>print("11----------------------------------------")<br>p2_emp.increment() | 1----------------------------------------<br>2----------------------------------------<br>Name: Nawaz Ali, Dept Marketing<br>Employee id: 102, Salary: 30000<br>3----------------------------------------<br>Collect work distribution loads from the HR department.<br>4----------------------------------------<br>Name: Nawaz Ali, Dept Marketing<br>Employee id: 102, Salary: 33000.0<br>5----------------------------------------<br>Name: Nadvi, Dept Human Resource<br>Employee id: 311, Salary: 30000<br>Employee Type: Foreign<br>6----------------------------------------<br>Collect work distribution details from the manager.<br>7----------------------------------------<br>Name: Nadvi, Dept Human Resource<br>Employee id: 311, Salary: 34500.0<br>Employee Type: Foreign<br>8----------------------------------------<br>9----------------------------------------<br>Name: Asif, Dept Sales<br>Employee id: 210, Salary: 15000<br>Employee Type: Part Time<br>10----------------------------------------<br>Collect work distribution loads from the HR department.<br>11----------------------------------------<br>Sadly, there is no increment for the part time |

```
print("12-----------------------------------------")
p2_emp.employeeDetails()
```

```
employees!!
12-----------------------------------------
Name: Olive, Dept Accounts
Employee id: 223, Salary: 15000
Employee Type: Part Time
```

## Task 2

Write the **ScienceExam** class so that the following code generates the output below:

```
class Exam:
    def __init__(self,marks):
        self.marks = marks
        self.time = 60

    def examSyllabus(self):
        return "Maths , English"
    def examParts(self):
        return "Part 1 - Maths\nPart 2 - English\n"


engineering = ScienceExam(100,90,"Physics","HigherMaths")
print(engineering)
print('----------------------------------')
print(engineering.examSyllabus())
print(engineering.examParts())
print('==================================')
architecture =
ScienceExam(100,120,"Physics","HigherMaths","Drawing")
print(architecture)
print('----------------------------------')
print(architecture.examSyllabus())
print(architecture.examParts())
```

```
OUTPUT:
Marks: 100 Time: 90 minutes Number
of Parts: 4
----------------------------------
Maths , English , Physics ,
HigherMaths
Part 1 - Maths
Part 2 - English
Part 3 - Physics
Part 4 - HigherMaths
==================================
Marks: 100 Time: 120 minutes Number
of Parts: 5
----------------------------------
Maths , English , Physics ,
HigherMaths , Drawing
Part 1 - Maths
Part 2 - English
Part 3 - Physics
Part 4 - HigherMaths
Part 5 - Drawing
```

## Task 3

Write the **PokemonExtra** class so that the following code generates the output below:

```
class PokemonBasic:

  def __init__(self, name = 'Default', hp = 0,
weakness = 'None', type = 'Unknown'):
    self.name = name
    self.hit_point = hp
    self.weakness = weakness
    self.type = type

  def get_type(self):
    return 'Main type: ' + self.type

  def get_move(self):
    return 'Basic move: ' + 'Quick Attack'

  def __str__(self):
    return "Name: " + self.name + ", HP: " +
str(self.hit_point) + ", Weakness: " + self.weakness

print('\n------------Basic Info:--------------')
pk = PokemonBasic()
print(pk)
print(pk.get_type())
print(pk.get_move())

print('\n------------Pokemon 1 Info:-------------')
charmander = PokemonExtra('Charmander', 39, 'Water',
'Fire')
print(charmander)
print(charmander.get_type())
print(charmander.get_move())

print('\n------------Pokemon 2 Info:-------------')
charizard = PokemonExtra('Charizard', 78, 'Water',
'Fire', 'Flying', ('Fire Spin', 'Fire Blaze'))
print(charizard)
print(charizard.get_type())
print(charizard.get_move())
```

***OUTPUT:***

```
------------Basic Info:--------------
Name: Default, HP: 0, Weakness: None
Main type: Unknown
Basic move: Quick Attack

------------Pokemon 1 Info:--------------
Name: Charmander, HP: 39, Weakness: Water
Main type: Fire
Basic move: Quick Attack

------------Pokemon 2 Info:--------------
Name: Charizard, HP: 78, Weakness: Water
Main type: Fire, Secondary type: Flying
Basic move: Quick Attack
Other move: Fire Spin, Fire Blaze
```

## Task 4

A renowned Bakery shop recently launched cheesecakes into their cakes menu. Cheesecakes will have all the general attributes of the regular cakes but it has some special features. Design the **Cakes** (parent) and **Cheese_Cakes** (child) classes so that the following output is produced. Note that:

- 1kg regular cake price is 1200 Taka and 1 kg cheese-cake price is 1500 Taka
- As cheese-cakes are newly launched, they need user feedback. For this reason, if a customer gives feedback on cheese-cakes he'll get 10% discounts on his next purchase.

Write the classes **Cakes** and **Cheese_Cakes** to generate the following output.

Driver Code:

```
order_1=Cakes("Chocolate",500)
order_2=Cakes("Vanilla",800)
print("(1)**********************************")
order_1.cake_details()
print("(1.1)**********************************")
print(Cakes.order_list)
print("(2)**********************************")
order_2.add_customization("Zero","Butter Cream")
order_2.cake_details()
print("(3)**********************************")
Cakes.give_feedbacks("Chocolate Cake","Very Delicious")
Cakes.give_feedbacks("Chocolate Cake","Yummy")
print("(4)**********************************")
Cakes.show_feedbacks()
print("(5)**********************************")
ch_order1=Cheese_Cakes("Red velvet",700)
ch_order1.cake_details()
print("(6)**********************************")
ch_order1.add_customization()
print("(7)**********************************")
ch_order2=Cheese_Cakes("Blue Berry",900,"No Bake")
ch_order2.cake_details()
print("(8)**********************************")
print(Cakes.order_list)
print("(9)**********************************")
Cheese_Cakes.give_feedbacks("Red velvet Cheese Cake","Average")
Cheese_Cakes.give_feedbacks("Blue Berry Cheese Cake","Liked it")
print("(10)**********************************")
Cakes.show_feedbacks()
```

Output:

```
(1)**********************************
Flavor: Chocolate Cake, Weight: 500 gm
Sweetness: Moderate sugar, Cream Type: Whipped Cream
Price: 600.0 Taka
(1.1)**********************************
{'Chocolate Cake 500gm': 1, 'Vanilla Cake 800gm': 1}
(2)**********************************
Flavor: Vanilla Cake, Weight: 800 gm
Sweetness: Zero sugar, Cream Type: Butter Cream
Price: 960.0 Taka
(3)**********************************
Thanks for your valuable feedback!
Thanks for your valuable feedback!
(4)**********************************
{'Chocolate Cake': ['Very Delicious', 'Yummy']}
(5)**********************************
Flavor: Red velvet Cheese Cake, Weight: 700 gm
Sweetness: Moderate sugar, Cream Type: Cream Cheese
Cake Type:baked, Price: 1050.0 Taka
(6)**********************************
Sorry! No customization available for cheese cakes.
(7)**********************************
Flavor: Blue Berry Cheese Cake, Weight: 900 gm
Sweetness: Moderate sugar, Cream Type: Cream Cheese
Cake Type:No Bake, Price: 1350.0 Taka
(8)**********************************
{'Chocolate Cake 500gm': 1, 'Vanilla Cake 800gm': 1, 'Red velvet Cheese Cake 700gm': 1, 'Blue Berry Cheese Cake 900gm': 1}
(9)**********************************
Thanks for your valuable feedback!
You will get 10% discount for your next purchase!
```

| | |
|---|---|
| | Thanks for your valuable feedback!<br>You will get 10% discount for your next purchase!<br>(10)**********************************<br>{'Chocolate Cake': ['Very Delicious', 'Yummy'], 'Red velvet Cheese Cake': ['Average'], 'Blue Berry Cheese Cake': ['Liked it']} |

## Task 5

```python
class A:
  temp = 3
  def __init__(self):
    self.sum = 0
    self.y = 0
    self.y = A.temp - 1
    self.sum = A.temp + 2
    A.temp -= 2

  def methodA(self, m, n):
    x = 0
    n[0] += 1
    self.y = self.y + m + A.temp
    A.temp += 1
    x = x + 2 + n[0]
    n[0] = self.sum + 2
    print(f"{x} {self.y} {self.sum}")

```

```python
class B(A):
  x = 1
  def __init__(self, b = None):
    super().__init__()
    self.sum = 2
    if b == None:
      self.y = self.temp + 1
      B.x = 3 + A.temp + self.x
      A.temp -= 2
    else:
      self.sum = self.sum + self.sum
      B.x = b.x + self.x
  def methodB(self, m, n):
    y = [0]
    self.y = y[0] + self.y + m
    B.x = self.y + 2 +  self.temp - n
    self.methodA(self.x, y)
    self.sum = self.x + y[0] + self.sum
    print(f"{self.x} {y[0]} {self.sum}")
```

Write the output of the following code:

| | | | |
|---|---|---|---|
| x = [23]<br>a1 = A()<br>b1 = B() | **Output:** | | |
| | x | y | sum |

| | | | |
|---|---|---|---|
| `b2 = B(b1)` | | | |
| `a1.methodA(1, x)` | | | |
| `b2.methodB(3, 2)` | | | |
| `a1.methodA(1, x)` | | | |
| | | | |