Q.707 Imagine you own a call center. Use the following abstract class template to create three more classes, Respondent,

Manager, and Director that inherit this Employee Abstract Class. from abc import ABC, abstractmethod class Employee(ABC): @abstractmethod def receive_call(self): pass

@abstractmethod def end_call(self): pass @abstractmethod def is_free(self): pass

@abstractmethod def get*rank(self): pass 707 9 9 Sr. No. unit* numb er question_text answer_t ext mark s option1 (A) option2 (B) option3 (C) option4 (D) L.J Institute of Engineering and Technology, Ahmedabad. FCSP-1 Question Bank (SEM-III) Note : This question bank is only for reference purpose. LJU Test question paper may not be completely set from this question bank. Create a program using the instructions given below:

1. Create a constructor in all three classes (Respondent, Manager and Director) which takes the id and name as input and initializes two additional variables, rank and free. rank should be equal to 3 for Respondent, 2 for Manager and 1 for Director. free should be a boolean variable with value True initially. (1 mark)
2. Implement rest of the methods in all three classes in the following way: (2 marks) a. receive_call(): prints the message, "call received by (name of the employee)" and sets the free variable to False. b. end_call(): prints the message, "call ended" and sets the free variable to True. c. is_free(): returns the value of the free variable d. get_rank(): returns the value of the rank variable
3. Create a class Call, with a constructor that accepts id and name of the caller and initializes a variable called assigned to False. (0.5 marks)
4. Create a class CallHandler, with three lists, respondents, managers and directors as class variables. (0.5 marks)
5. Create an add_employee() method in CallHandler class that allows you to add an employee (an object of Respondent/Manager/Director) into one of the above lists according to their rank. (1 mark)
6. Create a dispatch_call() method in CallHandler class that takes a call object as a parameter. This method should find the first available employee starting from rank 3, then rank 2 and then rank 1. If a free employee is found, call its receive_call() function and change the call's assigned variable value to True. If no free employee is found, print the message: "Sorry! All employees are currently busy." (2 marks)
7. Create 3 Respondent objects, 2 Manager objects and 1 Director object and add them into the list of available employees using the CallHandler's add_employee() method. (1 mark)
8. Create a Call object and demonstrate how it is assigned to an employee. (1 mark)

In [22]:
```
from abc import ABC, abstractmethod

class Employee(ABC):
    def __init__(self, emp_id, name):
        self.emp_id = emp_id
        self.name = name
        self.rank = None
        self.free = True

    @abstractmethod
    def receive_call(self):
        pass
```

```
    @abstractmethod
    def end_call(self):
        pass

    def is_free(self):
        return self.free

    def get_rank(self):
        return self.rank

class Respondent(Employee):
    def __init__(self, emp_id, name):
        Employee.__init__(self, emp_id, name)
        self.rank = 3

    def receive_call(self):
        print(f"Call received by Respondent {self.name}")
        self.free = False

    def end_call(self):
        print("Call ended")
        self.free = True

class Manager(Employee):
    def __init__(self, emp_id, name):
        Employee.__init__(self, emp_id, name)
        self.rank = 2

    def receive_call(self):
        print(f"Call received by Manager {self.name}")
        self.free = False

    def end_call(self):
        print("Call ended")
        self.free = True

class Director(Employee):
    def __init__(self, emp_id, name):
        Employee.__init__(self, emp_id, name)
        self.rank = 1

    def receive_call(self):
        print(f"Call received by Director {self.name}")
        self.free = False

    def end_call(self):
        print("Call ended")
        self.free = True

class Call:
    def __init__(self, caller_id, caller_name):
        self.caller_id = caller_id
        self.caller_name = caller_name
        self.assigned = False

class CallHandler:
    respondents = []
    managers = []
    directors = []

    @classmethod
    def add_employee(cls, employee):
        if employee.rank == 3:
            cls.respondents.append(employee)
        elif employee.rank == 2:
            cls.managers.append(employee)
```

```
        elif employee.rank == 1:
            cls.directors.append(employee)

    @classmethod
    def dispatch_call(cls, call):
        for employee_list in [cls.respondents, cls.managers, cls.directors]:
            for employee in employee_list:
                if employee.is_free():
                    employee.receive_call()
                    call.assigned = True
                    return
        print("Sorry! All employees are currently busy.")

# Create employees and add them to the CallHandler
respondent1 = Respondent(1, "John")
respondent2 = Respondent(2, "Alice")
respondent3 = Respondent(3, "Bob")

manager1 = Manager(4, "Charlie")
manager2 = Manager(5, "David")

director1 = Director(6, "Eve")

CallHandler.add_employee(respondent1)
CallHandler.add_employee(respondent2)
CallHandler.add_employee(respondent3)
CallHandler.add_employee(manager1)
CallHandler.add_employee(manager2)
CallHandler.add_employee(director1)

# Create a Call object and dispatch it
incoming_call = Call(101, "Customer")
CallHandler.dispatch_call(incoming_call)
incoming_call_2 = Call(102, "Customer1")
CallHandler.dispatch_call(incoming_call_2)
incoming_call_3 = Call(103, "Customer2")
CallHandler.dispatch_call(incoming_call_3)
incoming_call_4 = Call(101, "Customer")
CallHandler.dispatch_call(incoming_call_4)
```

```
Call received by Respondent John
Call received by Respondent Alice
Call received by Respondent Bob
Call received by Manager Charlie
```

Q. 708 Write a python program to create a Bus child class that inherits from the Vehicle class.

In Vehicle class vehicle name, mileage and seatingcapacity as its data member. The default fare charge of any vehicle is seating capacity * 100. If Vehicle is Bus instance, we need to add an extra 10% on full fare as a maintenance charge. So total fare for bus instance will become the final amount = total fare + 10% of the total fare. Sample Output: The bus seating capacity is 50. so, the final fare amount should be 5000+500=5500. The car seating capacity is 5. so, the final fare amount should be 500

In [16]:
```
class Vehicle:
    def __init__(self, vehicle_name, mileage, seating_capacity):
        self.vehicle_name = vehicle_name
        self.mileage = mileage
        self.seating_capacity = seating_capacity

    def calculate_fare(self):
```

```
        fare = self.seating_capacity * 100
        return fare

class Bus(Vehicle):
    def __init__(self, vehicle_name, mileage, seating_capacity):
        Vehicle.__init__(self, vehicle_name, mileage, seating_capacity)

    def calculate_fare(self):
        base_fare = Vehicle.calculate_fare(self)
        maintenance_charge = 0.1 * base_fare
        total_fare = base_fare + maintenance_charge
        return total_fare

# Example usage
bus_instance = Bus("Bus", 10, 50)
car_instance = Vehicle("Car", 20, 5)

bus_fare = bus_instance.calculate_fare()
car_fare = car_instance.calculate_fare()

print(f"The bus seating capacity is {bus_instance.seating_capacity}. "
      f"So, the final fare amount should be {bus_fare:.2f}.")

print(f"The car seating capacity is {car_instance.seating_capacity}. "
      f"So, the final fare amount should be {car_fare:.2f}.")
```

```
The bus seating capacity is 50. So, the final fare amount should be 5500.00.
The car seating capacity is 5. So, the final fare amount should be 500.00.
```

Q. 709 Create an abstract class named Shape.

Create an abstract method named calculate_area for the Shape class. Create Two Classes named Rectangle and Circle which inherit Shape class. Create calculate_area method in Rectangle class. It should return the area of the rectangle object. (area of rectangle = (length * breadth)) Create calculate_area method in Circle class. It should return the area of the circle object. (area of circle =πr^2)) Create objects of Rectangle and Circle class. The python Program Should also check whether the area of one Rectangle object is greater than another rectangle object by overloading > operator. Execute the method resolution order of the Circle class.

In [17]:
```
from abc import ABC, abstractmethod
from math import pi


class Shape(ABC):
    @abstractmethod
    def calculate_area(self):
        pass


class Rectangle(Shape):
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def calculate_area(self):
        return self.length * self.breadth

    def __gt__(self, other):
        return self.calculate_area() > other.calculate_area()


class Circle(Shape):
    def __init__(self, radius):
```

```
        self.radius = radius

    def calculate_area(self):
        return pi * self.radius**2


# Creating objects
rectangle1 = Rectangle(5, 10)
rectangle2 = Rectangle(8, 6)

circle1 = Circle(7)
circle2 = Circle(4)

# Checking area comparison of rectangles
if rectangle1 > rectangle2:
    print("Area of Rectangle 1 is greater than Rectangle 2.")
else:
    print("Area of Rectangle 2 is greater than Rectangle 1.")

# Executing the method resolution order of the Circle class
print("Method Resolution Order (MRO) of Circle class:", Circle.mro())
```

```
Area of Rectangle 1 is greater than Rectangle 2.
Method Resolution Order (MRO) of Circle class: [<class '__main__.Circle'>, <class '__main__.Shap
e'>, <class 'abc.ABC'>, <class 'object'>]
```

Q. 711 Create a class called Matrix containing constructor that initialized the number of rows and number of columns of a new

Matrix object. The Matrix class has methods for each of the following:

1. get the number of rows
2. get the number of columns
3. set the elements of the matrix at given position (i,j)
4. adding two matrices. If the matrices are not addable, " Matrices cannot be added" will be displayed. (Overload the addition operation to perform this)
5. Multiplying the two matrices. If the matrices are not multiplied, " Matrices cannot be multiplied" will be displayed.(Overload the addition operation to perform this)

In [23]:
```
class Matrix:
    def __init__(self, rows, columns):
        self.rows = rows
        self.columns = columns
        self.matrix = [[0] * columns for _ in range(rows)]

    def get_num_rows(self):
        return self.rows

    def get_num_columns(self):
        return self.columns

    def set_element(self, i, j, value):
        if 0 <= i < self.rows and 0 <= j < self.columns:
            self.matrix[i][j] = value
        else:
            print("Invalid position for setting element.")

    def add_matrices(self, other_matrix):
```

```
        if self.rows != other_matrix.get_num_rows() or self.columns != other_matrix.get_num_colu
            print("Matrices cannot be added.")
            return None

        result_matrix = Matrix(self.rows, self.columns)

        for i in range(self.rows):
            for j in range(self.columns):
                result_matrix.set_element(i, j, self.matrix[i][j] + other_matrix.matrix[i][j])

        return result_matrix

    def multiply_matrices(self, other_matrix):
        if self.columns != other_matrix.get_num_rows():
            print("Matrices cannot be multiplied.")
            return None

        result_matrix = Matrix(self.rows, other_matrix.get_num_columns())

        for i in range(self.rows):
            for j in range(other_matrix.get_num_columns()):
                element_sum = 0
                for k in range(self.columns):
                    element_sum += self.matrix[i][k] * other_matrix.matrix[k][j]
                result_matrix.set_element(i, j, element_sum)

        return result_matrix

    def __add__(self, other_matrix):
        return self.add_matrices(other_matrix)

    def __mul__(self, other_matrix):
        return self.multiply_matrices(other_matrix)

    def display_matrix(self):
        for row in self.matrix:
            print(row)


# Example Usage:
matrix1 = Matrix(2, 3)
matrix2 = Matrix(2, 3)

matrix1.set_element(0, 0, 1)
matrix1.set_element(0, 1, 2)
matrix1.set_element(0, 2, 3)
matrix1.set_element(1, 0, 4)
matrix1.set_element(1, 1, 5)
matrix1.set_element(1, 2, 6)

matrix2.set_element(0, 0, 7)
matrix2.set_element(0, 1, 8)
matrix2.set_element(0, 2, 9)
matrix2.set_element(1, 0, 10)
matrix2.set_element(1, 1, 11)
matrix2.set_element(1, 2, 12)

print("Matrix 1:")
matrix1.display_matrix()

print("\nMatrix 2:")
matrix2.display_matrix()

# Addition
result_addition = matrix1 + matrix2
if result_addition:
```

```
    print("\nMatrix Addition:")
    result_addition.display_matrix()

# Multiplication
result_multiplication = matrix1 * matrix2
if result_multiplication:
    print("\nMatrix Multiplication:")
    result_multiplication.display_matrix()
```

```
Matrix 1:
[1, 2, 3]
[4, 5, 6]

Matrix 2:
[7, 8, 9]
[10, 11, 12]

Matrix Addition:
[8, 10, 12]
[14, 16, 18]
Matrices cannot be multiplied.
```

Q. 712 Find the MRO of class Z of below program:

class A: pass class B: pass class C: pass class D:pass class E:pass class K1(C,A,B): pass class K3(A,D): pass class K2(B,D,E): pass class Z(K1,K3,K2): pass

In [25]:
```
class A:
    pass
class B:
    pass
class C:
    pass
class D:
    pass
class E:
    pass
class K1(C,A,B):
    pass
class K3(A,D):
    pass
class K2(B,D,E):
    pass
class Z( K1,K3,K2):
    pass
print(Z.__mro__)
```

```
(<class '__main__.Z'>, <class '__main__.K1'>, <class '__main__.C'>, <class '__main__.K3'>, <clas
s '__main__.A'>, <class '__main__.K2'>, <class '__main__.B'>, <class '__main__.D'>, <class '__ma
in__.E'>, <class 'object'>)
```

Q. 713 Write a Python Program to Find the Net Salary of Employee using Inheritance.

Create three Class Employee, Perks, NetSalary. Make an Employee class as an abstract class. Employee class should have methods for following tasks.

- To get employee details like employee id, name and salary from user.
- To print the Employee details.

- return Salary.
- An abstract method emp_id. Perks class should have methods for following tasks.
- To calculate DA, HRA, PF.
- To print the individual and total of Perks (DA+HRA-PF). Netsalary class should have methods for following tasks.
- Calculate the total Salary after Perks.
- Print employee detail also prints DA, HRA, PF and net salary. Note 1: DA-35%, HRA-17%, PF-12% Note 2: It is compulsory to create objects and demonstrating the methods with Correct output. Example: Employee ID: 1 Employee Name: John Employee Basic Salary: 25000 DA: 8750.0 HRA: 4250.0 PF: 3000.0 Total Salary: 35000.0

In [31]:
```
from abc import ABC, abstractmethod

class Employee(ABC):
    def __init__(self):
        self.emp_id = 0
        self.emp_name = ""
        self.emp_salary = 0

    def get_employee_details(self):
        self.emp_id = int(input("Enter Employee ID: "))
        self.emp_name = input("Enter Employee Name: ")
        self.emp_salary = float(input("Enter Employee Basic Salary: "))

    def print_employee_details(self):
        print("\nEmployee ID:", self.emp_id)
        print("Employee Name:", self.emp_name)
        print("Employee Basic Salary:", self.emp_salary)

    @abstractmethod
    def emp_id(self):
        pass

    def return_salary(self):
        return self.emp_salary

class Perks(Employee):
    def __init__(self):
        Employee.__init__(self)
        self.DA = 0
        self.HRA = 0
        self.PF = 0

    def calculate_perks(self):
        self.DA = 0.35 * self.emp_salary
        self.HRA = 0.17 * self.emp_salary
        self.PF = 0.12 * self.emp_salary

    def print_perks(self):
        print("\nDA:", self.DA)
        print("HRA:", self.HRA)
        print("PF:", self.PF)

    def total_perks(self):
        return self.DA + self.HRA - self.PF

class NetSalary(Perks):
    def __init__(self):
        Perks.__init__(self)

    def emp_id(self):  # Implementing the abstract method
        pass
```

```
    def calculate_total_salary(self):
        self.calculate_perks()
        total_salary = self.emp_salary + self.total_perks()
        return total_salary

    def print_salary_details(self):
        self.print_employee_details()
        self.print_perks()
        print("Total Salary:", self.calculate_total_salary())

# Example Usage:
employee = NetSalary()
employee.get_employee_details()
employee.print_salary_details()
```

```
Enter Employee ID: 1
Enter Employee Name: asd
Enter Employee Basic Salary: 5677

Employee ID: 1
Employee Name: asd
Employee Basic Salary: 5677.0

DA: 0
HRA: 0
PF: 0
Total Salary: 7947.8
```

In [ ]: