

Chap.- 9

Polymorphism

- using same to perform different methods

```
In [14]: 1 class A:
2         def display(self, x):
3             print("Display without parameter")
4
5         def display(self, x, y):
6             print("Display x :",x,"y :",y)
7
8 ob = A()
9 # ob.display()
10 #
11 ob.display(1,2) # Display x : 1 y : 2
12 ob.display(5, 1) # Display x : 5 y : 1
```

Display x : 1 y : 2
Display x : 5 y : 1

```
In [11]: 1 class A:
2         def display(self, x):
3             print("Display without parameter")
4
5         def display(self, x, y):
6             print("Display x :",x,"y :",y)
7
8 ob = A()
9 # ob.display()
10 #
11 ob.display(1,2) # Display x : 1 y : 2
12 ob.display(5) # TypeError: display() missing 1 required positional argu
```

Display x : 1 y : 2

```
-----
-
TypeError                                Traceback (most recent call las
t)
<ipython-input-11-8bbd0f12908e> in <module>
    10 #
    11 ob.display(1,2) # Display x : 1 y : 2
--> 12 ob.display(5) # Display x : 5 y : None

TypeError: display() missing 1 required positional argument: 'y'
```

```
In [15]: 1 class A:
2         def display(self):
3             print("Display without parameter")
4
5         def display(self, x, y=None):
6             print("Display x :",x,"y :",y)
7
8 ob = A()
9 ob.display(1,2) # Display x : 1 y : 2
10 ob.display(5) # Display x : 5 y : None
```

Display x : 1 y : 2
Display x : 5 y : None

```
In [24]: 1 class A:
2         def display(self, x, y=None):
3             if(y == None):
4                 print(x)
5             else:
6                 print(x, y)
7
8 ob = A()
9 # ob.display()
10 #
11 ob.display(1,2) # 1 2
12 ob.display(5) # 5
```

1 2
5

- Can't support overloading...
- In python we create illusion of using overloading.

```
In [19]: 1 class A:
2         def __init__(self, x, y):
3             self.x = x
4             self.y = y
5
6 p1 = A(2,3)
7 p2 = A(10,20)
8 print(p1 + p2)
9 # TypeError: unsupported operand type(s) for +: 'A' and 'A'
```

```
-----
-
TypeError                                Traceback (most recent call last)
<ipython-input-19-e58a3b6646e9> in <module>
      6 p1 = A(2,3)
      7 p2 = A(10,20)
----> 8 print(p1 + p2)
      9 # TypeError: unsupported operand type(s) for +: 'A' and 'A'
```

TypeError: unsupported operand type(s) for +: 'A' and 'A'

. Operator Overloading

In [22]:

```

1  class A:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def __add__(self, other):
7          x = self.x + other.x
8          y = self.y + other.y
9          return(x,y)
10
11 p1 = A(2,3)
12 p2 = A(10,20)
13 print(p1 + p2) # (12, 23)

```

(12, 23)

In [26]:

```

1  class A:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def __add__(self, other):
7          x = self.x + other.x
8          y = self.y + other.y
9          return(x,y)
10
11 p1 = A(2,3)
12 p2 = A(10,20)
13 print(p1 * p2) # TypeError: unsupported operand type(s) for *: 'A' and
14
15 # fun name je hoy te sign use karvi

```

```

-----
-
TypeError                                Traceback (most recent call las
t)
<ipython-input-26-6c8a3007c8d0> in <module>
      11 p1 = A(2,3)
      12 p2 = A(10,20)
----> 13 print(p1 * p2) # TypeError: unsupported operand type(s) for *: 'A'
and 'A'

```

TypeError: unsupported operand type(s) for *: 'A' and 'A'

```
In [28]: 1 class A:
2         def __init__(self, x, y):
3             self.x = x
4             self.y = y
5
6         def __add__(self, other):
7             x = self.x * other.x
8             y = self.y * other.y
9             return(x,y)
10
11 p1 = A(2,3)
12 p2 = A(10,20)
13 print(p1 + p2) # (20, 60)
```

(20, 60)

- **add**
- **sub**
- **mul**
- **truediv (/)**
- **floordiv (//)**
- **mod (%)**
- **pow (**)**
- **lt (<)**
- **gt (>)**
- **le (<=)**
- **ge (>=)**
- **ne (!=)**
- **eq (==)**

In [35]:

```
1 class A:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     def __add__(self, other):
7         x = self.x + other.x
8         y = self.y + other.y
9         return(x,y)
10    def __sub__(self, other):
11        x = self.x - other.x
12        y = self.y - other.y
13        return(x,y)
14    def __mul__(self, other):
15        x = self.x * other.x
16        y = self.y * other.y
17        return(x,y)
18    def __truediv__(self, other):
19        x = self.x / other.x
20        y = self.y / other.y
21        return(x,y)
22    def __floordiv__(self, other):
23        x = self.x // other.x
24        y = self.y // other.y
25        return(x,y)
26    def __mod__(self, other):
27        x = self.x % other.x
28        y = self.y % other.y
29        return(x,y)
30    def __pow__(self, other):
31        x = self.x ** other.x
32        y = self.y ** other.y
33        return(x,y)
34    def __lt__(self, other):
35        x = self.x < other.x
36        y = self.y < other.y
37        return(x,y)
38    def __gt__(self, other):
39        x = self.x > other.x
40        y = self.y > other.y
41        return(x,y)
42    def __le__(self, other):
43        x = self.x <= other.x
44        y = self.y <= other.y
45        return(x,y)
46    def __ge__(self, other):
47        x = self.x >= other.x
48        y = self.y >= other.y
49        return(x,y)
50    def __ne__(self, other):
51        x = self.x != other.x
52        y = self.y != other.y
53        return(x,y)
54    def __eq__(self, other):
55        x = self.x == other.x
56        y = self.y == other.y
57        return(x,y)
58
59
60 p1 = A(2,3)
61 p2 = A(10,20)
```

```
62 print(p1 + p2) # (12, 23)
63 print(p1 - p2) # (-8, -17)
64 print(p1 * p2) # (20, 60)
65 print(p1 / p2) # (0.2, 0.15)
66 print(p1 // p2) # (0, 0)
67 print(p1 % p2) # (2, 3)
68 print(p1 ** p2) # (1024, 3486784401)
69 print(p1 < p2) # (True, True)
70 print(p1 > p2) # (False, False)
71 print(p1 <= p2) # (True, True)
72 print(p1 >= p2) # (False, False)
73 print(p1 != p2) # (True, True)
74 print(p1 == p2) # (False, False)
```

```
(12, 23)
(-8, -17)
(20, 60)
(0.2, 0.15)
(0, 0)
(2, 3)
(1024, 3486784401)
(True, True)
(False, False)
(True, True)
(False, False)
(True, True)
(False, False)
```

P.b. = 680

```

In [45]: 1 class St:
2         def __init__(self, name, rn, age, marks):
3             self.n = name
4             self.r = rn
5             self.a = age
6             self.m = marks
7         def display(self):
8             print("Name :",self.n)
9             print("Roll No. :",self.r)
10            print("Age :",self.a)
11            print("Marks :",self.m)
12
13         def __eq__(self, other):
14             if(self.m == other.m):
15                 print("Both marks are Same :)")
16                 return True
17             else:
18                 print("Both marks are not Same")
19                 return False
20
21 s1 = St('Romil', 84, 18, 24)
22 s2 = St('Yash' , 94, 18, 25)
23 s3 = St('Rudra' , 90, 18, 24)
24
25 print(s1 == s2) # Both marks are not Same
26 print(s1 == s3) # Both marks are Same :)

```

Both marks are not Same

False

Both marks are Same :)

True

• Inheritance :

```

In [51]: 1 class A:
2         def demo(self):
3             print("Class A")
4         class B:
5             def dis(self):
6                 print("Class B")
7
8         ob = B()
9         ob.demo()
10        ob.dis() # AttributeError: 'B' object has no attribute 'demo'

```

```

-----
-
AttributeError                                Traceback (most recent call las
t)
<ipython-input-51-2092b482f89d> in <module>
7
8     ob = B()
----> 9     ob.demo()
10     ob.dis() # AttributeError: 'B' object has no attribute 'demo'

AttributeError: 'B' object has no attribute 'demo'

```


• Single (Simple) Inheritance

```
In [54]: 1 class A:
2         def demo(self):
3             print("Class A")
4         class B(A):
5             def dis(self):
6                 print("Class B")
7
8         ob = B()
9         ob.demo() # Class A
10        ob.dis() # Class B
```

Class A

Class B

```
In [56]: 1 class A:
2         def demo(self):
3             print("Class A")
4         class B(A):
5             def dis(self):
6                 print("Class B")
7
8         ob = A()
9         ob.demo() # Class A
10        ob.dis() # Class B
```

Class A

```
-----
-
AttributeError                                Traceback (most recent call las
t)
<ipython-input-56-0d7de25aaab0> in <module>
      8 ob = A()
      9 ob.demo() # Class A
----> 10 ob.dis() # Class B

AttributeError: 'A' object has no attribute 'dis'
```

P.b. 671

In [63]:

```

1  class Book():
2      def __init__(self):
3          self.name = input("Enter Name of Book : ")
4          self.no = input("Enter No. of Book : ")
5          self.a = input("Enter Name of Author : ")
6          self.pub = input("Enter Name of Publisier : ")
7          self.isbn = input("Enter ISBN : ")
8          self.y = input("Enter Year : ")
9
10     def display(self):
11         print("Name :", self.name)
12         print("No. :", self.no)
13         print("Name of Author :", self.a)
14         print("Publisier :", self.pub)
15         print("ISBN :", self.isbn)
16         print("Year :", self.y)
17
18     class TextBook(Book):
19         def __init__(self):
20             super().__init__()
21             self.co = input("Enter Course : ")
22
23         def display(self):
24             super().display()
25             print("Course : ", self.co)
26
27     # -----
28     B1 = TextBook()
29     B1.display()

```

```

Enter Name of Book : sdsd
Enter No. of Book : 234
Enter Name of Author : dfh
Enter Name of Publisier : jhg
Enter ISBN : 5214
Enter Year : 2045
Enter Course : adfgad
Name : sdsd
No. : 234
Name of Author : dfh
Publisier : jhg
ISBN : 5214
Year : 2045
Course :  adfgad

```

• Types of Inheritance

1.) Single P-C

2.) Multiple 2P-C

3.) Multilevel

4.) Heirachical

5.) Hybrid

. Multiple Inheritance

```

In [69]: 1 class Person():
2         def __init__(self):
3             self.name = input("Enter Name : ")
4             self.age = input("Enter Age : ")
5
6         class Car():
7             def __init__(self):
8                 self.model = input("Enter Model : ")
9                 self.Color = input("Enter Color : ")
10
11        class Parking(Person, Car):
12            def __init__(self):
13                Person.__init__(self)
14                Car.__init__(self)
15                self.pn = input("Enter Parking No. : ")
16            def display(self):
17                print("-----Person Details-----")
18                print("Person Name :", self.name)
19                print("Person Age :", self.age)
20                print("-----Car Details-----")
21                print("Car Model :", self.model)
22                print("Car Color :", self.Color)
23                print("-----Parking Details-----")
24                print("Parking No. :", self.pn)
25
26        ob = Parking()
27        ob.display()
28
29        # Enter Name : Romil
30        # Enter Age : 18
31        # Enter Model : Mustang 1969
32        # Enter Color : Black
33        # Enter Parking No. : 8
34        # -----Person Details-----
35        # Person Name : Romil
36        # Person Age : 18
37        # -----Car Details-----
38        # Car Model : Mustang 1969
39        # Car Color : Black
40        # -----Parking Details-----
41        # Parking No. : 8

```

```

Enter Name : Romil
Enter Age : 18
Enter Model : Mustang 1969
Enter Color : Black
Enter Parking No. : 8
-----Person Details-----
Person Name : Romil
Person Age : 18
-----Car Details-----
Car Model : Mustang 1969
Car Color : Black
-----Parking Details-----
Parking No. : 8

```

- **Method Resolution Order (Left to Right)**
- **DFS Search (Depth First Search)**

Mcq. 636

```
In [5]: 1 class A:
2         def rk(self):
3             print(" In class A")
4 class B:
5         def rk(self):
6             print(" In class B")
7 class C(A, B):
8         def rk(self):
9             pass
10 r = C()
11 print(C.__mro__)
12 # (<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <c
13
14 print(B.__mro__) # (<class '__main__.B'>, <class 'object'>)
15
```

```
(<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class
'object'>)
(<class '__main__.B'>, <class 'object'>)
```

Mcq. 648

```
In [8]: 1 class P:
2         pass
3 class Q:
4         pass
5 class R(P,Q):
6         pass
7 class S(Q):
8         pass
9 class T(S,R):
10        pass
11 a=T()
12 T.__mro__
13 # (__main__.T, __main__.S, __main__.R, __main__.P, __main__.Q, object)
```

```
Out[8]: (__main__.T, __main__.S, __main__.R, __main__.P, __main__.Q, object)
```

Mcq. 640

```
In [15]: 1 class A:
2         pass
3 class B:
4         pass
5 class C:
6         pass
7 class X(A,B):
8         pass
9 class Y(C,A,B):
10        pass
11 class Z(A):
12        pass
13 class P(Z,Y,X):
14        pass
15
16 P.__mro__
17
18 # (__main__.P,
19 #  __main__.Z,
20 #  __main__.Y,
21 #  __main__.C,
22 #  __main__.X,
23 #  __main__.A,
24 #  __main__.B,
25 #  object)
```

```
Out[15]: (__main__.P,
__main__.Z,
__main__.Y,
__main__.C,
__main__.X,
__main__.A,
__main__.B,
object)
```

- **Also Check left to right in Sub-Parent (for all tree)**

In [20]:

```
1 class A:
2     pass
3 class B:
4     pass
5 class C:
6     pass
7 class X(A,B):
8     pass
9 class Y(A,B,C):
10    pass
11 class Z(A):
12    pass
13 class P(Z,Y,X):
14    pass
15
16 P.__mro__
17
18 # (__main__.P,
19 #  __main__.Z,
20 #  __main__.Y,
21 #  __main__.X,
22 #  __main__.A,
23 #  __main__.B,
24 #  __main__.C,
25 #  object)
```

```
Out[20]: (__main__.P,
__main__.Z,
__main__.Y,
__main__.X,
__main__.A,
__main__.B,
__main__.C,
object)
```

In [21]:

```
1 class A:
2     pass
3 class B:
4     pass
5 class C:
6     pass
7 class X(B,A):
8     pass
9 class Y(A,B,C):
10    pass
11 class Z(A):
12    pass
13 class P(Z,Y,X):
14    pass
15
16 P.__mro__
17
18
19 # TypeError: Cannot create a consistent method resolution
20 # order (MRO) for bases A, B
```

TypeError

Traceback (most recent call last)

t)

<ipython-input-21-40d2f551462d> in <module>

```
11 class Z(A):
12     pass
--> 13 class P(Z,Y,X):
14     pass
15
```

TypeError: Cannot create a consistent method resolution
order (MRO) for bases A, B

Mcq. 695

```
In [24]: 1 class A: pass
2 class B: pass
3 class C: pass
4 class D: pass
5 class E: pass
6 class K1(C,A,B): pass
7 class K3(A,D): pass
8 class K2(B,D,E): pass
9 class Z( K1,K3,K2): pass
10
11 Z.__mro__
12
13 # (__main__.Z,
14 # __main__.K1,
15 # __main__.C,
16 # __main__.K3,
17 # __main__.A,
18 # __main__.K2,
19 # __main__.B,
20 # __main__.D,
21 # __main__.E,
22 # object)
```

```
Out[24]: (__main__.Z,
__main__.K1,
__main__.C,
__main__.K3,
__main__.A,
__main__.K2,
__main__.B,
__main__.D,
__main__.E,
object)
```

P.b. 691

```
In [25]: 1 # Write a python program to create a Bus child class that inherits from
2 # In Vehicle class vehicle name, mileage and seatingcapacity as its dat
3 # seating capacity * 100. If Vehicle is Bus instance, we need to add an
4 # total fare for bus instance will become the final amount = total fare
5 # Sample Output:
6 # The bus seating capacity is 50. so, the final fare amount should be 5
7 # The car seating capacity is 5. so, the final fare amount should be 50
```

In [30]:

```
1 class Vehicle():
2     def __init__(self):
3         self.vn = input("Enter Vehicle Name : ")
4         self.m = int(input("Enter Mileage : "))
5         self.sc = int(input("Enter Seating Capacity : "))
6
7     def fare(self):
8         return self.sc*100
9
10 class Bus(Vehicle):
11     def __init__(self):
12         super().__init__()
13     def display(self):
14         print(self.name, self.mileage, self.sc)
15     def fare(self):
16         return super().fare()+super().fare()*0.1
17 class car(Vehicle):
18     def __init__(self):
19         super().__init__()
20     def fare(self):
21         return super().fare()
22 b=Bus()
23 print(b.fare())
24 c=car()
25 print(c.fare())
```

```
Enter Vehicle Name : abc
Enter Mileage : 50
Enter Seating Capacity : 50
5500.0
Enter Vehicle Name : sdf
Enter Mileage : 50
Enter Seating Capacity : 100
10000
```

P.b 694

```
In [5]: 1 class Matrix():
2         def __init__(self):
3             self.r = 3
4             self.c = 3
5
6         def getr(self):
7             print(self.r)
8
9         def getc(self):
10            print(self.c)
11
12        def set_m(self):
13            self.mat = []
14            for i in range(self.r):
15                l=[]
16                for j in range(self.c):
17                    n = int(input("Enter element : "))
18                    l.append(n)
19                self.mat.append(l)
20            print(self.mat)
21
22 m = Matrix()
23 m.getr()
24 m.getc()
25 m.set_m()
```

```
3
3
Enter element : 1
Enter element : 2
Enter element : 3
Enter element : 4
Enter element : 5
Enter element : 6
Enter element : 7
Enter element : 8
Enter element : 9
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Abstraction

- when inheriting class then class with restriction

```
In [ ]: 1 # from abc import ABC          \
2         #                             / } abstract class
3         # abc - abstraction base class /
```

```
In [6]: 1 from abc import ABC
2 class Abdemo(ABC):
3     def demo(self):
4         pass
5
6 class Base (Abdemo):
7     def display(self):
8         print("Base")
9
10 ob = Base()
11 ob.display()
```

Base

• use Decorator to create abstract method

```
In [12]: 1 from abc import ABC, abstractmethod
2 class AbDemo(ABC):
3     @abstractmethod
4     def demo(self):
5         pass
6
7 class Base (AbDemo):
8     def display(self):
9         print("Base")
10
11 ob = Base()
12 ob.display()
13
14 # TypeError: Can't instantiate abstract class Base with abstract method
```

```
-----
-
TypeError                                Traceback (most recent call last)
<ipython-input-12-f15ec5789ca0> in <module>
      9         print("Base")
     10
--> 11 ob = Base()
     12 ob.display()

TypeError: Can't instantiate abstract class Base with abstract methods demo
```

```
In [9]: 1 from abc import ABC, abstractmethod
2 class AbDemo(ABC):
3     @abstractmethod
4     def demo(self):
5         pass
6
7 class Base(Abdemo):
8     def demo(self):
9         print("Base")
10
11 ob = Base()
12 ob.demo()
13
14 # Base
```

Base

P.b.- 692 IMP

In [17]:

```

1  from abc import ABC, abstractmethod
2  class Shape(ABC):
3      @abstractmethod
4      def cal_area(self):
5          pass
6
7  class Rect(Shape):
8      def cal_area(self, l, b):
9          self.Rarea = l * b
10         return self.Rarea
11
12     def __gt__(self, other):
13         return self.Rarea > other.Rarea
14
15  class Circle(Shape):
16      def cal_area(self, r):
17          self.Carea = 3.14 * r**2
18          return self.Carea
19
20  r1 = Rect()
21  print(r1.cal_area(10,20))
22  r2 = Rect()
23  print(r2.cal_area(20,30))
24
25  c1 = Circle()
26  print(c1.cal_area(10))
27
28  print(r1 > r2)
29  print(r2 > r1)
30
31  print(Circle.__mro__)
32
33  # 200
34  # 600
35  # 314.0
36  # False
37  # True
38  # (<class '__main__.Circle'>, <class '__main__.Shape'>, <class 'abc.ABC'>, <class 'object'>)

```

200

600

314.0

False

True

```

(<class '__main__.Circle'>, <class '__main__.Shape'>, <class 'abc.ABC'>, <class 'object'>)

```

P.b.- 690 IMP

In [28]:

```
1 from abc import ABC, abstractmethod
2 class Employee(ABC):
3     @abstractmethod
4     def receive_call(self):
5         pass
6
7     @abstractmethod
8     def end_call(self):
9         pass
10    @abstractmethod
11    def is_free(self):
12        pass
13
14    @abstractmethod
15    def get_rank(self):
16        pass
17
18 class Respondent(Employee):
19     def __init__(self):
20         self.id = 101
21         self.name = 'abc'
22         self.rank = 3
23         self.free = True
24
25     def receive_call(self):
26         print("call received by", self.name)
27         self.free = False
28
29     def end_call(self):
30         print("call ended")
31         self.free = True
32
33     def is_free(self):
34         return self.free
35
36     def get_rank(self):
37         return self.rank
38
39 class Manager(Employee):
40     def __init__(self):
41         self.id = 103
42         self.name = 'lmn'
43         self.rank = 2
44         self.free = True
45
46     def receive_call(self):
47         print("call received by", self.name)
48         self.free = False
49
50     def end_call(self):
51         print("call ended")
52         self.free = True
53
54     def is_free(self):
55         return self.free
56
57     def get_rank(self):
58         return self.rank
59
60 class Director(Employee):
61     def __init__(self):
```



```
62         self.id = 104
63         self.name = 'xyz'
64         self.rank = 1
65         self.free = True
66
67     def receive_call(self):
68         print("call received by", self.name)
69         self.free = False
70
71     def end_call(self):
72         print("call ended")
73         self.free = True
74
75     def is_free(self):
76         return self.free
77
78     def get_rank(self):
79         return self.rank
80
81 class Call():
82     def __init__(self):
83         self.id = 102
84         self.name = "indipendent"
85         self.assigned = False
86
87 class CallHandler():
88     respondents = []
89     managers = []
90     directors = []
91
92     def add_employee(self, ob):
93         if ob.rank == 3:
94             CallHandler.respondents.append(ob)
95         elif ob.rank == 2:
96             CallHandler.managers.append(ob)
97         elif ob.rank == 1:
98             CallHandler.directors.append(ob)
99
100    def dispatch_call(self, call):
101        for employee in CallHandler.respondents:
102            if employee.is_free():
103                employee.receive_call()
104                call.assigned = True
105                print(f"Call assigned to {employee.name} (Respondent)")
106                return
107
108        for employee in CallHandler.managers:
109            if employee.is_free():
110                employee.receive_call()
111                call.assigned = True
112                print(f"Call assigned to {employee.name} (Manager)")
113                return
114
115        for employee in CallHandler.directors:
116            if employee.is_free():
117                employee.receive_call()
118                call.assigned = True
119                print(f"Call assigned to {employee.name} (Director)")
120                return
121
122        print("Sorry! All employees are currently busy.")
```

```

123
124
125 r1 = Respondent()
126 r2 = Respondent()
127 r3 = Respondent()
128 m1 = Manager()
129 d1 = Director()
130
131 ch = CallHandler()
132 ch.add_employee(r1)
133 ch.add_employee(r2)
134 ch.add_employee(r3)
135
136 ch.add_employee(m1)
137 ch.add_employee(d1)
138
139 call = Call()
140
141 ch.dispatch_call(call)
142 ch.dispatch_call(call)
143 ch.dispatch_call(call)
144 ch.dispatch_call(call)
145
146
147 # Output:
148
149 # call received by abc
150 # Call assigned to abc (Respondent)
151 # call received by abc
152 # Call assigned to abc (Respondent)
153 # call received by abc
154 # Call assigned to abc (Respondent)
155 # call received by lmn
156 # Call assigned to lmn (Manager)
157
158 # call received by abc
159 # Call assigned to abc (Respondent)
160
161 # If Respondent is Busy then :
162 # call received by lmn
163 # Call assigned to lmn (Manager)

```

```

call received by abc
Call assigned to abc (Respondent)
call received by abc
Call assigned to abc (Respondent)
call received by abc
Call assigned to abc (Respondent)
call received by lmn
Call assigned to lmn (Manager)

```

P.b.- 696

```
In [*]: 1 from abc import ABC, abstractmethod
2
3 class Employee(ABC):
4     @abstractmethod
5     def data(self):
6         self.id = 101
7         self.name = 'abc'
8         self.salary = 25000
9
10    def display(self):
11        print(f"ID : {self.id} Name : {self.name} Salary : {self.salary}")
12
13    def get_sal(self):
14        return self.salary
15
16    @abstractmethod
17    def emp_id(self):
18        pass
19
20    class Perks(Employee):
21        def cal_perk(self):
22            self.da = self.salary * 0.35
23            self.hra = self.salary * 0.17
24            self.pf = self.salary * 0.12
25            self.total = self.da + self.hra + self.pf
26            print("DA : ",self.da)
27            print("HRA : ",self.hra)
28            print("PF : ",self.pf)
29            print("Total : ", self.total)
30
31    class NetSalary(Perks):
32        def cal_nets(self):
33            self.final = self.salary + self.total
34            print("Net Salary : ", self.final)
35
36    n = NetSalary()
37
38    n.cal_perk()
39    n.cal_nets()
```

```
In [ ]: 1
```