11-10-2024

Training Day – 19

Multiple Inheritance: One Child Multiple Parents

```
# class C1:
#     def __init__(self):
#         self.a=1
#         self.b=2
#     # def showData(self):
#     #     print("I am in Class C1")
# class C2:
#     def __init__(self):
#         self.c=3
#         self.d=4
#     # def showData(self):
#     #     print("I am in Class C2")
# class C3:
#     def __init__(self):
#         self.e=5
#         self.f=6
#     def showData(self):
#         print("I am in Class C3")
# class C4(C1,C2,C3):
#     def __init__(self):
#         self.g=7
#         self.h=8
#         super().__init__()
#         C2.__init__(self)
#         C3.__init__(self)
#     # def showData(self):
#     #     print("I am in Class C4")
# obj=C4()       #object of C4. obj.__init__()
# obj.showData()
# print(obj.a,obj.b,obj.c,obj.d,obj.e,obj.f,obj.g,obj.h)


"""
```

In above program, if we have showData method in all 3 parents but not in child. Now if we call showData method using child object then it may create an ambiguity or in Java it creates ambiguity that which parent showData method will be called? So this ambiguity in Java is called Diamond Problem and there is no direct solution made for diamond problem in Java ie Java doesn't support Multiple Inheritance. But in Python this a solution is made for this ambiguity using Priority setting and making MRO (Method Resolution Order) If there are multiple parents of one child ie C1, C2, C3 then direct priority will be given from left to right as we mention parents name in child class.

Now in above program as we have multiple parents for one child
so to create variables of all parents, we need to call constructor
of all the parents in child class.

# Hybrid Inheritance:

```
# Case 1 as shared in painting:
# """
# class C1:
#     def __init__(self):
#         self.a=1
#         self.b=2
#     def showData(self):
#         print("I am in Class C1")
# class C2(C1):
#     def __init__(self):
#         self.c=3
#         self.d=4
#         super().__init__()
#     # def showData(self):
#     #     print("I am in Class C2")
# class C3(C1):
#     def __init__(self):
#         self.e=5
#         self.f=6
#         super().__init__()
#     # def showData(self):
#     #     print("I am in Class C3")
# class C4(C2,C3):
#     def __init__(self):
#         self.g=7
#         self.h=8
#         super().__init__()
#         C3.__init__(self)
#     # def showData(self):
#     #     print("I am in Class C4")
# obj=C4()
# print(obj.a,obj.b,obj.c,obj.d,obj.e,obj.f,obj.g,obj.h)
# obj.showData()
```

Polymorphism:

Overloading: is also called compile time polymorphism

Overriding: is also called run time polymorphism

Real definition of access specifiers as per OOPS concepts used in C++ or Java or other:

Public: Can be accessed anywhere in itself or child or outside class
Protected: Can be accessed only in itself or in child class
Private: Can be accessed only in itself.
.