Inheritance

When to implement the inheritance?

Requirement:

I have to create a class with data/variables and methods

which are already existed in another class

but without re-creating those in my class I want inherit those into my class

from the existing class.

In this case, we can use "Inheritance".

What is inheritance?

=> Inheritance is the process of creating new class from the existing class by inherit the properties and methods from existing class into new class.

New class ==> Child class/Sub class/Derived class

Existing class ==> Parent Class/Super class/Base class

How to implement the inheritance?

Syntax:

Child-class-name(Parent-class-name)

Payment:

transactionId;

transactionDate;

amount;

CardPayment(Payment):

transactionId

transactionDate

amount

cardType

cardNumber

cvv

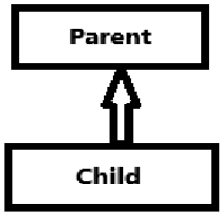
Types of Inheritance?

=> there are 5-types of inheritances:

1) Single Inheritance
2) Multi-level Inheritance
3) Multiple Inheritance
4) Hierarchical Inheritance
5) Hybrid Inheritance

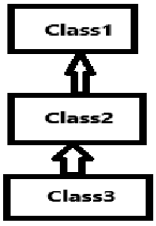
Single Inheritance

One to one
there are no extensions with other class

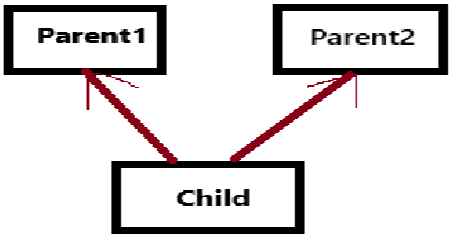


Multi-level Inheritance

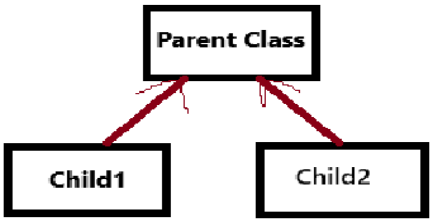
class ----- sub1 ---- sub11



Multiple Inheritance

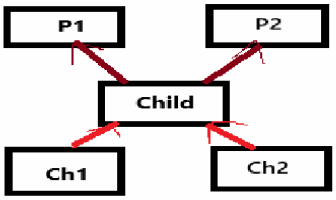


Hierarchical Inheritance



Hybrid Inheritance

Combination of Multiple Inheritance And Hierarchical Inheritance



Note:

---

to call the parent class constructor we need "super() call".

=> super() call is the default statement in every constructor.

If we cannot write the super() call in the constructor the PVM can automatically write it.

=> super() call can always use to invoke the parent class constructor.

=> super() call is always the first statement in the constructor.

=> super() call can use without parameters and with parameters also.

Single Inheritance:

```
# Employee management System

class Employee:
    def __init__(self,name,salary):
        super()
        self.name = name
        self.salary = salary

    def show_details(self):
        print("Employee Name = ",self.name)
        print("Employee Salary = ",self.salary)

class Developer(Employee):
    def __init__(self,name,salary,programming_language):
        super().__init__(name,salary)
        self.progrmming_language = programming_language

    def show_developer_details(self):
        self.show_details()
        print("Programming language = ",self.progrmming_language)
```

```
dev1 = Developer("Ravi", 97000, "Python")
dev1.show_developer_details()
```

Multi-level Inheritance:

```
# Vehicle Management System

class Vehicle:
    def __init__(self,brand, model):
        self.brand = brand
        self.model = model

    def show_info(self):
        print("Brand name = ",self.brand)
        print("Model = ",self.model)

class Car(Vehicle):
    def __init__(self,brand, model,fuel_type):
        super().__init__(brand,model)
        self.fuel_type = fuel_type

    def show_Car_Info(self):
        self.show_info()
        print("Fuel Type = ",self.fuel_type)
```

```
class ElectricCar(Car):
    def __init__(self,brand, model, battery_capacity):
        super().__init__(brand,model,"Electric")
        self.battery_capacity = battery_capacity

    def show_electric_car_info(self):
        self.show_Car_Info()
        print("Battery Capacity = ",self.battery_capacity)



tesla = ElectricCar("Tesla","Model 3",75)
tesla.show_electric_car_info()
```

Multiple Inheritance:

Two or more parent classes

and one child class

here:

child class can inherit all members (properties and behaviors) from all parent classes.

Parent classes should be independent together.

```
# Student Management System


# parent1 class
class Person:
    def __init__(self,name,age):
        self.name = name
```

```
        self.age = age

    def show_person(self):
        return f"Name : {self.name}, Age : {self.age}"

# Parent2 class
class Marks:
    def __init__(self,math,science, english):
        self.math = math
        self.science = science
        self.english = english

    def total_marks(self):
        return self.math + self.science + self.english

    def grade(self):
        avg = self.total_marks() / 3

        if avg >= 90:
            return "A+"
        elif avg >= 75:
            return "A"
        elif avg >= 50:
            return "B"
        else:
            return "C"
```

```
class Student(Person, Marks):
    def __init__(self,name,age, math, science, english):
        Person.__init__(self,name,age)
        Marks.__init__(self,math,science,english)

    def show_student(self):
        return self.show_person() + f"\nTotal Marks : {self.total_marks()}, Grade : {self.grade()}"

s1 = Student("Navin", 20, 95,85,90)
print(s1.show_student())

s2 = Student("Satish",19,89,79,99)
print(s2.show_student())
```

Assignment:

---

1) Write a logic to implement hierarchical inheritance

Parent ----------> ch1, ch2, ..

2) Write a logic to implement hybrid inheritance

Hierarchical Inheritance + Multiple Inheritance ===> Hybrid

Polymorphism:

Polymorphism is a word has taken from two Greek words:

poly ==> many

morphs ==> forms

class:

constructor

variables

methods

constructor ==> to initialize the data members

variables ==> store values

methods ==> perform task/functionality

=> In python, polymorphism can implement in three ways:

1) Operator overloading

2) Method overloading

3) Method overriding

Polymorphism with operator overloading:

```
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def __add__(self, other):
        return BankAccount(self.balance + other.balance)

    def __str__(self):
        return f"Balance : {self.balance}"

a1 = BankAccount(5000)
```

```
a2 = BankAccount(3000)

merged = a1 + a2
print(merged)
```

Polymorphism with method overloading:

```
class Notification:
    def send(self, *args):
        if len(args) == 1:
            print(f"Sending email:{args[0]}")
        elif len(args) == 2:
            print(f"Sending SMS to : {args[0]} : {args[1]}")
        else:
            print("Invalid Notification format")
n = Notification()
n.send("Welcome to our service!")
n.send("+91-9876543210","your OTP is 1234")
```

Polymorphism with method overriding:

```
class Animal:
    def speak(self):
        return "Some generic Sound"

class Dog(Animal):
    def speak(self):
```

```
            return "Bark"

class Cat(Animal):
    def speak(self):
            return "Meow"

animals = [Dog(), Cat(), Animal()]
for a in animals:
    print(a.speak())
```