Experiment 6

Title : Provide a solution to the problem using python object oriented concepts.

Aim : To understand use of various object oriented concepts.

Theory : There are five important features related to object oriented programming :

1) Classes & objects
2) Encapsulation
3) Abstraction
4) Inheritance
5) Polymorphism

1] Classes & objects

Classes - the classes is user defined data structure that binds the data members & methods into a single unit. classes is a blue print or code template for object creation.

Object : An object is an instance of classes. It is a collection of attributes & methods we use the object of classes to perform actions.

Syntax :

```
class class_name:
    # statements
```

Create object in python :

Object creation divided into two parts in object creation & object initialization

--new--() is the method that create the object.

using --init--() method we can implement constructor to initialize the object

Syntax : <objName> = <className>(<arglist>)
Example :

```
class person:
    def __init__(self, name, rollno):
        self.name = name
        self.rollno = rollno
    def printdata(self):
        print("Name :", self.name)
        print("Roll No.:", self.rollno)
p1 = person("Shubham Sawant", 146)
p1.printdata()
```

2] Encapsulation

Encapsulation is a mechanism where the data & the code that act on the data will bind together.

Example :

```
class Person:
    def __init__(self):
        self.rollno = 146
        self.name = "Shubham Sawant"
    def printdata(self):
        print("Name :", self.name)
        print("Roll No. :", self.rollno)
st = Person()
st.printdata()
```

3] Abstraction

There may be a lot of data, a class contains & the user does not need the entire data. the user requires only some part of the available data

Page No:.....
Roll No. :.....

In this case, we can hide the unnecessary data from the user & expose only that data that is of interest to the user this is called abstraction.

4] Inheritance

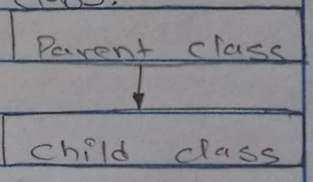
The process of inherting the properties of the parent class into a child class is called inheritance. The existing class is called a base class or parent class & new class is called a subclass or child class or derived class.

Syntax : class BaseClass:
    # statements for BaseClass
class DerivedClass(BaseClass):
    # statements for derived class.

Types of Inheritance :

1) Single inheritance

In single inheritance, a child class inherits from a single-parent class. Here is one child class & one parent class.



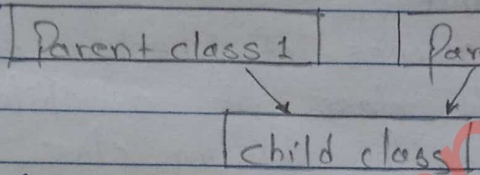
Example :

```
class A:
    def mA(self):
        print("A")
class B(A):
    def mB(self):
        print("B")
obj = B()
obj.mA()
obj.mB()
```

Page No:.....
Roll No. :.....

2) Multiple Inheritance

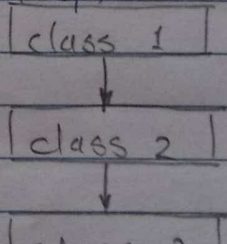
In multiple inheritance, one child class can inherit from multiple parent classes, so here is one child class & multiple parent classes.




```
Example: class A:
             def mA(self):
                 print("A")
         class B:
             def mB(self):
                 print("B")
         class C(A, B):
             def mC(self):
                 print("C")
         obj = C()
         obj.mA()
         obj.mB()
         obj.mC()
```

3) Multilevel Inheritance

Multilevel inheritance is a concept in object-oriented programming where a derived class inherits from a base class & another class is derived from that derived class. This creates a hierarchy of classes with each class inheriting characteristics & behavior from its immediate parent class.

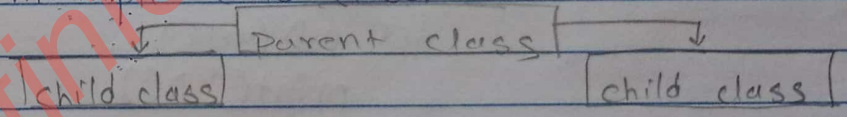


Page No:.....
Roll No. :.....

```
Example: class A:
             def mA(self):
                 print("A")
         class B(A):
             def mB(self):
                 print("B")
         class C(B):
             def mC(self):
                 print("C")
         obj=C()
         obj.mC()
         obj.mB()
         obj.mA()
```

4) Hierarchical Inheritance

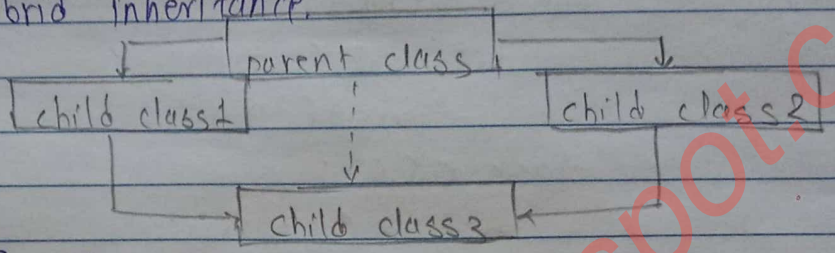
In hierarchical inheritance, more than one child class is derived from a single parent class in other words, we can say one parent class & multiple child class.




```
Example: class A:
             def mA(self):
                 print("A")
         class B(A):
             def mB(self):
                 print("B")
         class C(A):
             def mC(self)
                 print("C")
         obj = C()
         obj.mC()
         obj.mB()
```

5) Hybrid Inheritance

When inheritance is consisd of multiple types or a combination of different inheritance is called hybrid inheritance.



Example:

```
class A :
    def mA(self):
        print("A")
class B(A):
    def mB(self):
        print("B")
class C(A)
    def mC(self):
        print("C")
class D(C)
    def mD(self):
        print("D")
obj d= D()
d.mD()
d.mC()
d.mB()
d.mA()
```

5] Polymorphism

The word 'polymorphism' come from two Greek words "poly" meaning many & "morphos" meaning forms. Thus, polymorphism represents the ability to assume several different forms.

In programming, if an object or method is exhibiting different behavior in different contexts, it is called polymorphic nature. Polymorphism provides flexibility in writing programs in such a way that the programmer use is some method call to perform different operations depending on the requirement.

We can implement polymorphism using funcation overloading & operator overloading.

Example :

```
def join (x, y):
    return(x + y)
print(join(20, 14))
print(join("Shubham ", "Sawant"))
```

Conclusion : In object-oriented programming, the concept of inheritance allows classes to inherit properties & methods from other classes, promoting code reuse & hierarchical relationships. Encapsulation, on the other hand, enables bundling data & methods together, providing data protection & promoting modular & maintainable code.