Object Oriented Programming

- Object Oriented is an approach to software development that models application around real world objects.
- The main aim of OOP is to bind together the data and the functions that operates on the given application using class.

Object: Objects are basic run-time entities in an object oriented system.

Objects are instances of a class.

They are defined user defined data types.

Class is an extensible program-code-template for creating objects, providing initial values for state and implementations of behavior for member functions.

Features of the class

- A class can have subclasses that can inherit all or some of the characteristics from the superclass.
- Class can define it’s own methods and variables.
- The structure of a class and its subclasses is called the class hierarchy.

Features of Object Oriented Programming

- Encapsulation
- Polymorphism
- Inheritance

Encapsulation -

- Wrapping up of data and functions into a single unit is known as encapsulation.
- The Object implementation and state are hidden behind well-defined boundaries of the class.

Polymorphism means ability to take more than one form.

An operation may exhibit different behaviors in different instances.

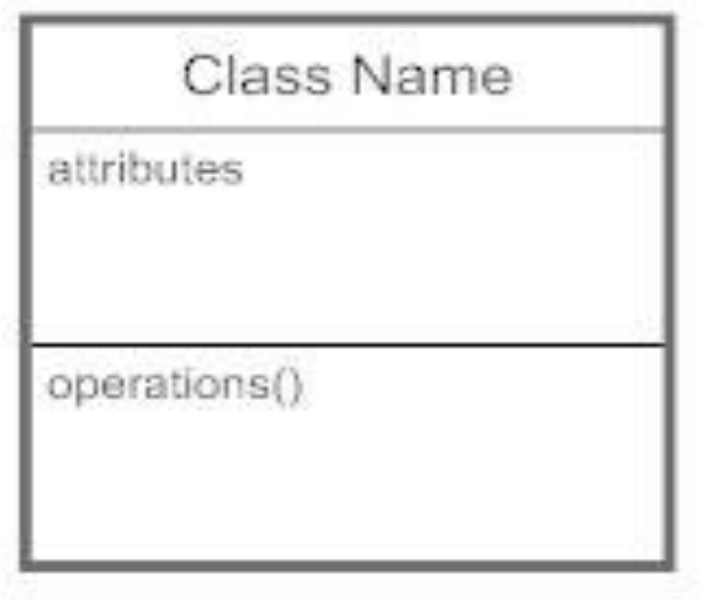
The behavior depends upon the types of data used in the operation.

- Inheritance: inheritance is the process by which objects of one class acquire the properties of objects of another class.

- Inheritance provides re usability.

UML

Unified Modeling Language UML is a technique used to design and document object oriented systems.



Active class

- The Upper box contains the class name
- The middle box contains the class variables
- The lower box contains the class methods
- The minus (-) sign means private scope
- The plus (+) sign means public scope

- The hash (#) sign means protected scope

Access Specifier of the Class

- Public: Public method or variable can be accessible from anywhere in an application.
- Private: Method or property with private visibility can only be accessible inside the class.
- Protected: Method or variable with protected visibility can only be access in the derived class. Protected will be used in the process of inheritance.

Examples of Inheritance –

Ex1-

```
class MyClass:

    a = 10

print(MyClass.a)
```

Ex2-

```
class MyClass:

    a = 100

    b=200

    c=300

print(MyClass.a)

print(MyClass.b)

print(MyClass.c)
```

Ex3-

```
class MyClass:
    a = 100
    b="Hello world"
    c={"aaa","bbb","ccc"}
print(MyClass.a)
print(MyClass.b)
print(MyClass.c)
```

Ex4-

```
class MyClass:
  def function(self):
      print("This is a message inside the class.")
objectx = MyClass()
objectx.function()
```

Ex5-

```
class MyClass1:
  def function1(self):
      print("This is a message inside the class1.")
objectx = MyClass1()
objectx.function1()
class MyClass2:
  def function2(self):
        print("This is a message inside the class2.")
objecty = MyClass2()
objecty.function2()
class MyClass3:
    def function3(self):
        print("This is a message inside the class3.")
objectz = MyClass3()
objectz.function3()
```

Ex6-

```
class Employee:
    def __init__(self, name):
        self.name = name
    def displayEmployee(self):
        print "Name : ", self.name
emp1 = Employee("abc")
emp1.displayEmployee()
```

Ex7-

```
class EmployeeName:
    def __init__(self, name):
        self.name = name
    def displayEmployee(self):
        print "Name : ", self.name
emp1 = EmployeeName("abc")
```

```
emp1.displayEmployee()
class EmployeeAge:
  def __init__(self, age):
    self.age = age
  def displayAge(self):
    print "Age : ", self.age
emp2 = EmployeeAge(30)
emp2.displayAge()
class EmployeeSalary:
  def __init__(self, salary):
    self.salary = salary
  def displaySalary(self):
    print "Salary : ", self.salary
emp3 = EmployeeSalary(30275.75)
emp3.displaySalary()
```

Ex8-

```
class Employee:
  empCount = 0
  def __init__(self, name, salary):
    self.name = name
    self.salary = salary
    Employee.empCount += 1

  def displayEmployee(self):
    print "Name : ", self.name,  ", Salary: ", self.salary
emp1 = Employee("abc", 2000)
emp2 = Employee("xyz", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee is", Employee.empCount
```

Inheritance Examples

Ex1-

```
class Person:
    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last
    def Name(self):
        return self.firstname + " " + self.lastname
class Employee(Person):
    def __init__(self, first, last, staffnum):
        Person.__init__(self,first, last)
        self.staffnumber = staffnum
    def GetEmployee(self):
        return self.Name() + ", " +  self.staffnumber
x = Person("mahesh", "patil")
y = Employee("ravi", "k", "2007")
print(x.Name())
print(y.GetEmployee())
```

Ex2-

```
#Line:1, definition of the superclass starts here
class Car:
    #defining constructor
    def __init__(self, carName, carModel):
        self.name = carName
        self.model = carModel
      #defining class methods
    def showName(self):
        print(self.name)
      def showModel(self):
        print(self.model)
      #Line: 19, end of superclass definition
  #definition of subclass starts here
class CarInfo(Car):
    carRegNo = ""
      def __init__(self, carName, carModel, carRegNo):
        Car.__init__(self, carName, carModel)
        #Line: 26, Calling the superclass constructor
        self.carRegNo = carRegNo
      def showCarNo(self):
        print(self.carRegNo)
#end of subclass definition
# Create an object of the superclass
```

```
car1 = Car("Hyundai i10", "Magna")

#call member methods of the objects

car1.showName()

car1.showModel()

car2=CarInfo("Hyundai i10","Magna","KA25 Z 1000")

car2.showCarNo()
```

Ex3-

```
#Line:1, definition of the superclass starts here

class Car:

    #initializing the variables

    name = ""

    cost = 0

      #defining constructor

    def __init__(self, carName, carModel):

        self.name = carName

        self.model = carModel

      #defining class methods

    def showName(self):

        print(self.name)

      def showModel(self):

        print(self.model)

      #Line: 19, end of superclass definition

  #definition of subclass starts here

class CarInfo(Car):
    carRegNo = ""
     def __init__(self, carName, carModel, carRegNo):
        Car.__init__(self, carName, carModel)  #Line: 26, Calling the superclass constructor and sending values of attributes.
        self.carRegNo = carRegNo


    def showCarNo(self):
        print(self.carRegNo)
#end of subclass definition
# Create an object of the superclass
car1 = Car("Hyundai i10", "Magna")  #Line: 35
#call member methods of the objects
car1.showName()
car1.showModel()
car2=CarInfo("Hyundai i10","Magna","KA25 Z 1000")
car2.showCarNo()
```

Ex4-

```
#Line:1, definition of the superclass starts here
class Person:
    #initializing the variables
    name = ""
    age = 0
     #defining constructor
    def __init__(self, personName, personAge):
        self.name = personName

        self.age = personAge

      #defining class methods

     def showName(self):

        print("The name is ",self.name)

      def showAge(self):

        print(self.age)

      #Line: 19, end of superclass definition

  #definition of subclass starts here

class Student(Person):

    #Line: 22, Person is the  superclass and Student is the subclass

    studentId = ""

      def __init__(self, studentName, studentAge, studentId):

        Person.__init__(self, studentName, studentAge)

        #Line: 26, Calling the superclass constructor and sending values of attributes.

        self.studentId = studentId

      def getId(self):

        return self.studentId

    #returns the value of student id

#end of subclass definition

  # Create an object of the superclass

person1 = Person("Richard", 23)

#call member methods of the objects

person1.showAge()

# Create an object of the subclass
```

```
student1 = Student("Martin", 22, "102")

print(student1.getId())

student1.showName()
```

Ex5-

```
class Animal:

    def eat(self):

      print 'Eating...'

class Dog(Animal):

   def bark(self):

      print 'Barking...'

class BabyDog(Dog):

    def weep(self):

        print 'Weeping...'

d=BabyDog()

d.eat()

d.bark()

d.weep()
```