In [1]:

```
# the class statement creates a new class definition
# the name of the class follows the keyword class and it is followed by a colon
class Employee:
    # common base class for all employees

    # the variable empCount is a class variable whose value
    # is shared among all the instances of this class
    empCount = 0

    # this below is the class constructor or initialization method
    # that Python calls when you create a new instance of this class
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    # the first argument to each method is self
    def displayCount(self):
        print ("Total Employee %d" % Employee.empCount)

    def displayEmployee(self):
        print ("Name: ", self.name,  ", Salary: ", self.salary)


#This would create first object of Employee class"
emp1 = Employee("Ana", 2000)
#This would create second object of Employee class"
emp2 = Employee("Alexandru", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

```
Name:  Ana , Salary:  2000
Name:  Alexandru , Salary:  5000
Total Employee 2
```

In [2]:

```
emp1.displayEmployee()
# add a new 'salary' attribute
emp1.salary = 7000
# modify 'name' attribute
emp1.name = 'Raisa'
emp1.displayEmployee()
```

```
Name:  Ana , Salary:  2000
Name:  Raisa , Salary:  7000
```

In [3]:

```
# Returns true if 'salary' attribute exists
print("Salary attribute exists: ", hasattr(emp1, 'salary'))
# Returns value of 'salary' attribute
print("Salary: ",getattr(emp1, 'salary'))
# Set attribute 'salary' at 6000
setattr(emp1, 'salary', 6000)
# Returns value of 'salary' attribute
print("New salary: ", getattr(emp1, 'salary'))
# Delete attribute 'salary'
# delattr(emp1, 'salary')
```

```
Salary attribute exists:  True
Salary:  7000
New salary:  6000
```

In [17]:

```
# Python deletes unneeded objects (built-in types or class instances)
# automatically to free the memory space
# the process by which Python periodically reclaims blocks of memory
# that no longer are in use is termed as Garbage Collection

class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    # the destructor is invoked when the instance
    # is about to be destroyed
    # this destructor below prints the class name of an
    # instance that is about to be destroyed
    def __del__(self):
        class_name = self.__class__.__name__
        print (class_name, "destroyed")

pt1 = Point()
pt2 = pt1
pt3 = pt1
# This below prints the ids of the objects
print (id(pt1), id(pt2), id(pt3))
del pt1
del pt2
del pt3
```

```
599696157888 599696157888 599696157888
Point destroyed
```

In [21]:

```
# Class inheritance

# define parent class
class Parent:
    parentAttr = 100

    def __init__(self):
        print ("Calling parent constructor")

    def parentMethod(self):
        print ('Calling parent method')

    def setAttr(self, attr):
        Parent.parentAttr = attr

    def getAttr(self):
        print ("Parent attribute :", Parent.parentAttr)

    # overriding methods
    def myMethod(self):
        print ('Calling parent method')

# define child class
class Child(Parent):
    def __init__(self):
        print ("Calling child constructor")

    def childMethod(self):
        print ('Calling child method')

    # overriding methods
    def myMethod(self):
        print ('Calling child method')

c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method

# overriding methods
print("---Override a method---")
c.myMethod()         # child calls overridden method
```

```
Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200
---Override a method---
Calling child method
```

In [22]:

```
# Data hiding
class JustCounter:
    # the attributes named with a double underscore prefix
    # will not be directly visible to outsiders
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print(self.__secretCount)

counter = JustCounter()
counter.count()
counter.count()
print(counter.__secretCount)
```

```
1
2

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-22-d7786fc99992> in <module>()
     10 counter.count()
     11 counter.count()
---> 12 print(counter.__secretCount)

AttributeError: 'JustCounter' object has no attribute '__secretCount'
```

In [23]:

```
# use this line instead
print(counter._JustCounter__secretCount)
```

```
2
```