Lecture:10 Classes I

Department of Computer Science and Engineering, ITER
Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar, Odisha, India.

Contents

CLASSES I

- Class is a template that provides logical grouping grouping of data and methods that operate on them.
- In OOP we define/ create objects(instances of class).
- Objects used are derived using classes.
- Related code and data are packaged together using classes.
- We can have user defined classes and there are in-built classes also.
- Data and attributes with a class are collectively known as class attributes.
- We can assign instances to objects of class.

Classes and Objects

- We have various in-built class in python like str, int, float etc.
- 'Raman','CBSE' are instances of class str that are assigned to variable name and board.
- Various methods of class can act on objects of class.
- Method lower() can be used on object to return all the lowercase letters of string.
- methods are invoked by the object name followed by dot and the attribute name.
- Also, we can invoke class methods by class name followed by dot operator, method name and object name passed as an argument for method.

Person-An example of class

Syntax of defining class

```
class ClassName:
        classbody
```

```
class Person:
        count=0

def __init__(self, name, DOB, address):
        self.name=name
        self.DOB=DOB
        self.address=address
        Person.count+=1
```

About Person

```
def getName(self):
        return self.name

def getDOB(self):
        return self.DOB

def getAddress(self):
        return self.address

def setName(self,name):
        self.name=name

def setDOB(self,DOB):
        self.DOB=DOB
```

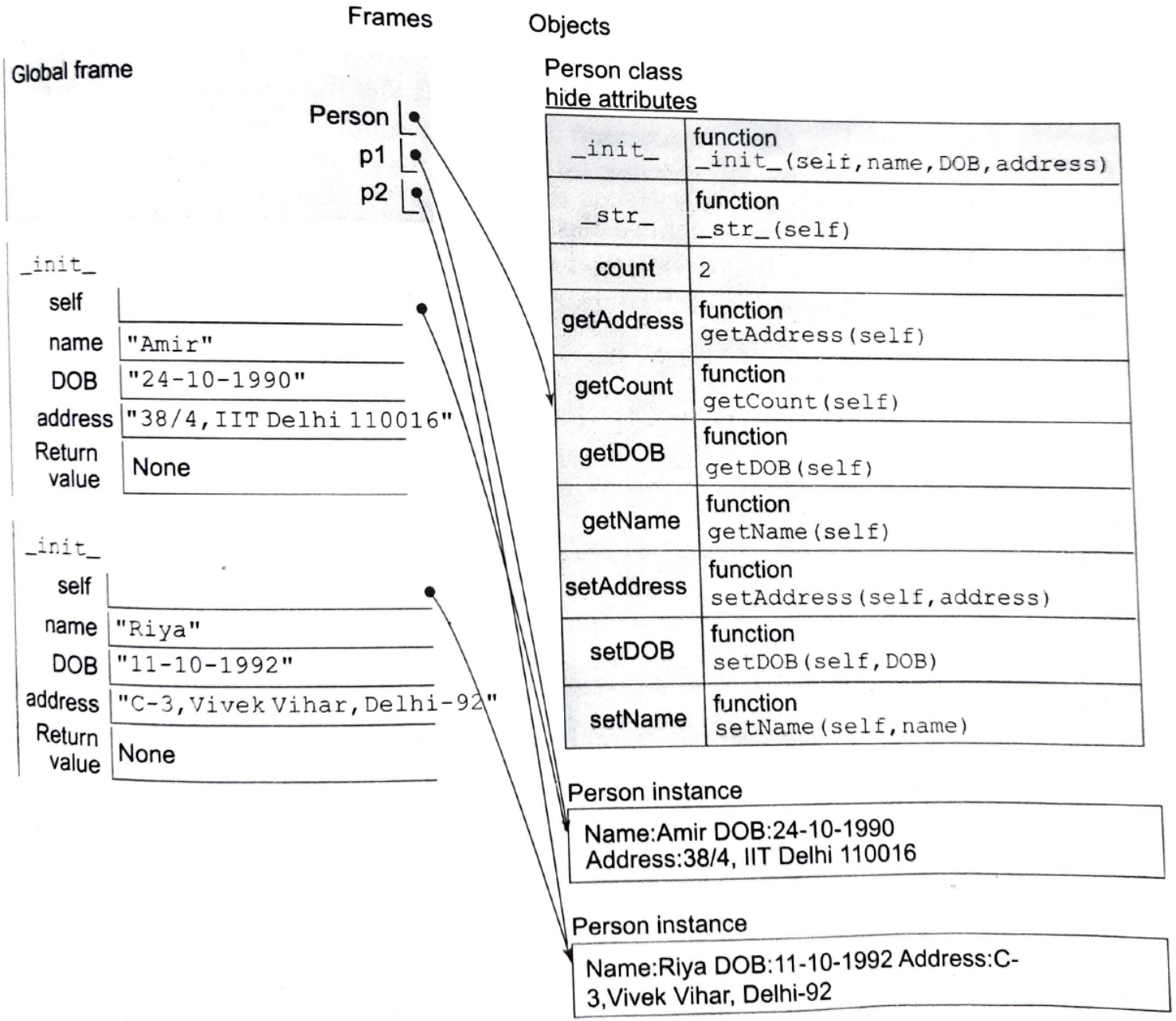
Create Class

```
def setAddress(self,address):
        self.address=address

def getCount(self):
        return Person.count

def __str__(self):
        return 'Name:'+self.name+':'+str(self.DOB)+':'+self.address
```

Visualisation



Constructor

- Constructors are generally used for instantiating an object.
- In Python the __init__() method is called the constructor and is always called when an object is created.
- The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created.

Destructor

Default Constructor

- The default constructor is a simple constructor which doesn't accept any arguments.
- Its definition has only one argument which is a reference to the instance being constructed.

Parameterized constructor

- Constructor with parameters is known as parameterized constructor.
- The parameterized constructor takes its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

Destructor

- We use **del** statement to remove no more required objects.
- Execution of del *reduces the reference count by one*.
- __del__ method is invoked when count is zero.
- del is actually called the destructor because everytime class object reduces by one and so we reduce the count by one.

Destructor Cont.

```
def __del__(self):
        '''
        Objective: To be invoked on deletion
        of an instance of the class person.
        Input parameter:
            self(implicit parameter)-object of
            type Person
        Return Value:None
        '''
        print('Deleted !!')
        Person.count-=1
```

Class as Abstract Datatype

- Python introduces an abstract data type i.e., not having any concrete existence.
- Objects of a class can be created, modified and destroyed.
- Class defined can be saved as independent module that can be imported for use in any module.
- Class attributes can also be imported after changing the directory using sys module.

Cont.

```
import sys
import os
sys.path.append('F:\Pythoncode\Ch10')
from person import person
def main():
    print('*****dir(Person):\n',dir(Person))
    print('*****Person.__doc__:\n',Person.__doc__)
    print('*****Person.__module__:\n',Person.__module__)

    p1=Person('Amir','24-10-1991','38/4, IIT Delhi 110016')
    print('*****Person.count:\n',Person.count)
    print('*****p1.getcount():\n',p1.getcount())
    print('*****p1.__doc__:\n',p1.__doc__)
    print('*****p1.__module__:\n',p1.__module__)
    print('*****p1:\n',p1)
    print('*****dir(p1):\n',dir(p1))
```

Cont.

```
p2=Person('Riya','11-10-1992','C-3, Vivek Vihar, Delhi-92')
    print('*****Person.count:\n',Person.count)
    print('*****p2.getcount():\n',p2.getcount())
    print('*****p1.getcount():\n',p1.getcount())
    print('*****p2.__doc__:\n',p2.__doc__)
    print('*****p2.__module__:\n',p2.__module__)
    print('*****p2:\n',p2)
    print('*****dir(p2):\n',dir(p2))

    print('\n***** id:Person.__doc__:\n',id(Person.__doc__))
    print('*****id:Person.__module__:\n',id(Person.__module__))
    print('*****id:p1.__doc__:\n',id(p1.__doc__))
    print('*****id:p1.__module__:\n',id(p1.__module__))
    print('*****id:p2.__doc__:\n',id(p2.__doc__))
    print('*****id:p2.__module__:\n',id(p2.__module__))
    print('*****id:dir(Person):\n',id(dir(Person)))
    print('*****id:dir(p1):\n',id(dir(p1)))
    print('*****id:dir(p2):\n',id(dir(p2)))
```

Cont.

```
    print('\n*****Person.__dict__\n',Person.__dict__)
    print('\n*****p1.__dict__\n',p1.__dict__)
    print('\n*****p2.__dict__\n',p2.__dict__)

if __name__=='__main__':
    main()
```

Date Class

- Today and defaultDate are two objects of class MyDate.
- Arguments passed while invoking constructors may be invalid.
- method checkDay is defined in order to check the arguments whether valid or not.
- print statement of above file returns the output in the string format.

Date Class Contd.

```
import sys
class MyDate:
    def__init__(self,day=1,month=1,year=2000):
        if not(type(day)==int and type(month)==int \\
            and type(year)==int):
            print('Invalid␣data␣provided␣for␣date')
            sys.exit()
        if month>0 and month <=12:
            self.month=month
        else:
            print('Invalid␣value␣for␣month')
            sys.exit()
        if year>1900:
            self.year=year
        else:
            print('Invalid␣value␣for␣year.␣Year␣\\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣should␣be␣greater␣than␣1900.')
            sys.exit()
        self.day=self.checkday(day)
```

Date Class Contd.

```
 def checkday(self,day):
        if self.year%400==0 or \\
         (self.year%100!=0 and self.year%4==0):
            currentYear=[31,29,31,30,31,30,31,31,30,31,30,31]
        else:
            currenYear=[31,28,31,30,31,30,31,31,30,31,30,31]
        if (day>0 and day<=currentYear[self.month-1]):
            return day
        else:
            print('Invalid value for day')
            sys.exit()
def __str__(self):
        if self.day<=9:
            day='0'+str(self.day)
        else:
            day=str(self.day)
        if self.month<=9:
            month='0'+str(self.month)
        else:
            month=str(self.month)
        return day+'-'+month+'-'+str(self.year)
```

Date Class Contd.

```
def main():
        today=MyDate(3,9,2014)
        print(today)
        defaultDate=MyDate()
        print(defaultDate)
    if __name__=='__main__':
        main()
```

Thank you

Any questions?