Day 8

Date 15 June 2024

Daily Report

Today's training was based on Object Oriented Concept - Abstraction and Inheritence.

Today's topic

Abstraction

The ability to use something without having to know the detail of how it is working is called as abstraction.

Abstraction is needed to prevent access to sensitive data.

Data Access

- Public data access:- public data can be access and modify easily.
- Private data:- In pyhton using(__) in front of arrtribute, it become private. it doesnt means it can't access but not easily as Python convert attribute into _class_name__Attribute name

Getter and Setter

To have a error free way of accessing and updating private variables, we create specific methods for this.

- The methods which are meant to set a value to a private variable are called setter methods.
- The methods meant to access private variable values are called getter methods.

Inheritence

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows one class (child or subclass) to inherit the properties and methods of another class (parent or superclass). It promotes code reusability by enabling a new class to take on the attributes and behaviors of an existing class.

Key Concepts

1. Parent Class (Superclass):

- Also known as a base class or superclass.
- It's the class whose attributes and methods are inherited by another class.

2. Child Class (Subclass):

- Also known as a derived class or subclass.
- It inherits attributes and methods from its parent class and can also have its own additional attributes and methods.

3. Inheritance Syntax in Python:

- In Python, inheritance is declared by specifying the parent class(es) in the definition of a child class.
- Syntax: class ChildClassName(ParentClassName):

Types of Inheritance:

- Single Inheritance: A child class inherits from only one parent class.
- Multiple Inheritance: A child class inherits from multiple parent classes.
- Multilevel Inheritance: One class is derived from another, which is itself derived from another class.
- Hierarchical Inheritance: Multiple child classes inherit from the same parent class.
- Hybrid Inheritance: Combination of two or more types of inheritance.

some question for practice:- **Library Book Management System**

- Create a `Book` class with private attributes `title`, `author`, and `isbn`.
- Implement getter and setter methods for `title` and `author`.
- Implement a getter method for `isbn`.

```
class book:
  def __init__(self,title,author,isbn):
    self.__title = title
    self.__author = author
    self.__isbn = isbn
  def author(self):
    return self.__author
  def title(self):
    return self.__title
  def isbn(self):
    return self.__isbn
class library:
  def __init__(self):
    self.lis = []
  def add(self,b):
    self.lis.append(b)
  def remove(self,b):
    self.lis.remove(b)
  def display(self):
    for x in self.lis:
      print("BOOK:")
      print("Author: ",x.author())
      print("Title: ",x.title())
      print("ISBN: ",x.isbn(),"\n")
b1 = book("A","B",123)
b2 = book("c","D",456)
l = library()
l.add(b1)
l.add(b2)
l.display()
l.remove(b1)
l.display()
```

```
BOOK:
Author:  B
Title:  A
ISBN:  123

BOOK:
Author:  D
Title:  c
ISBN:  456

BOOK:
Author:  D
Title:  c
ISBN:  456
```

```
class Athlete:
    def __init__(self,name,gender):
        self.__name=name
        self.__gender=gender
    def running(self):
        if(self.__gender=="girl"):
            print("150mtr running")
        else:
            print("200mtr running")
a = Athlete("Maria","girl")
a.running()
```

```
150mtr running
```

Bank Account Management System

- Create a `BankAccount` class with private attributes `account_number`, `account_holder`, and `balance`.
- Implement getter and setter methods for `account_number` and `account_holder`.
- Implement a setter method for `balance` that checks if the balance being set is non-negative.

```
class BankAccount:
  def __init__(self,account_number, account_holder,balance):
    self.__an = account_number
    self.__ah = account_holder
    self.__bal = balance
```

```
    def set_attribute(self,ac_no,balan):
        if ac_no%100000==0:
            self.__an = ac_no
        else:
            print("Wrong Account number!!")
        if balan >100:
          self.__bal = balan
        else:
          print("TO start the account please deposit atleat 100$")

    def get_attribute(self):
        print("holder: ",self.__ah)
        print("ACC : ",self.__an)
        print("Balance : ",self.__bal)

    def deposit(self,rupp):
        self.__bal +=rupp
        print("Amount is deposit in account")

    def withdraw(self,rupp):
        w = self.__bal - rupp
        if w>0:
          self.__bal = w
        else:
          print("Do not have enough balance!!")
    def balance(self):
        print("Balance :",self.__bal)
ac = BankAccount(12345,"honey",20000)
ac.get_attribute()
ac.deposit(670)
ac.balance()
ac.withdraw(723)
ac.balance()
```

```
holder:  honey
ACC :  12345
Balance :  20000
Amount is deposit in account
Balance : 20670
Balance : 19947
```

Employee Management System

- Create an `Employee` class with private attributes `employee_id`, `name`, `position`, and `salary`.
- Implement getter and setter methods for `position`.
- Implement a setter method for `salary` that ensures the salary being set is positive.

```
class employee:
  def __init__(self,employee_id, name, position,salary):
    self.__id = employee_id
    self.__name = name
    self.__position = position
    self.__sal = salary
  def set_attribute(self,s):
    if(s>0):
      self.__sal = s
    else:
      print("negative salary can't be possible!!")
  def emp_id(self):
    return self.__id
  def name(self):
    return self.__name
  def pos(self):
    return self.__position
  def sal(self):
    return self.__sal
class department:
  def __init__(self):
    self.lis = []
  def add(self,a):
    self.lis.append(a)
    print("New Employee is added!!")
  def remove(self,id):
    for x in self.lis:
      if x.emp_id() == id:
        self.lis.remove(x)
  def display(self):
    for x in self.lis:
      print("Name: ",x.name())
      print("ID: ",x.emp_id())
      print("Position: ",x.pos())
      print("Salary : ",x.sal())

d = department()
d.add(employee(123,"john","manager",30000))
d.add(employee(124,"merry","reception",10000))
d.add(employee(125,"Raj",'manager',30000))
d.display()
d.remove(124)
d.remove(123)
print("after")
d.display()
```

```
New Employee is added!!
New Employee is added!!
New Employee is added!!
Name:  john
ID:  123
Position:  manager
Salary :  30000
Name:  merry
ID:  124
Position:  reception
Salary :  10000
Name:  Raj
ID:  125
Position:  manager
Salary :  30000
after
Name:  Raj
ID:  125
Position:  manager
Salary :  30000
```

Student Record System

- Create a `Student` class with private attributes `student_id`, `name`, `age`, and `grades` (a list of integers).
- Implement getter and setter methods for `name` and `age`.
- Implement a setter method for `grades` that ensures all grades are within a valid range (e.g., 0-100).

```
class student:
  def __init__(self,student_id, name, age,grades):
    self.__id = student_id
    self.__name = name
    self.__age = age
    self.__grade = grades
  def id(self):
    return self.__id
  def name(self):
    return self.__name
  def age(self):
    return self.__age
  def grade(self):
    return self.__grade
  def set_name(self,name):
    self.__name = name
  def set_age(self,age):
    self.__age = age
  def set_grade(self,grade):
    self.__grade.clear()
    for x in grade:
      if x>0 and x<100:
        self.__grade.append(x)
s1 = student(123,"John",17,[67,78,45,34])
s2 = student(124,"Merry",18,[78,89,56,78])
s1.set_grade([23,67,45,89])
s1.grade()
```

que 5. Online Shopping System

Create a Product class with private attributes product_id, name, and price. Implement getter and setter methods for product_id. Implement setter methods for name and price that perform validation (e.g., ensure name is not empty and price is positive).

```
class product:
  def __init__(self,product_id,name,price):
    self.__id = product_id
    self.__name = name
    self.__price = price
  def set_id(self,id):
    self.__id = id
  def id(self):
    return self.__id
  def set_name(self,name):
    if len(name)>0:
      self.__name = name
    else:
      print("Name is empty!!!")
  def name(self):
    return self.__name
  def set_price(self,price):
    if price>0:
      self.__price = price
    else:
      print("Price is Negative!!!")
  def price(self):
    return self.__price
```