|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\Admin\Desktop\download.jpg | USMAN INSTITUTE OF TECHNOLOGY | | | | | |  |
|  |  |  |  |  |  |  |  |
|  | Department of Computer Science  CS121 Object Oriented Programming | | | | | |  |
|  |  | Lab # 07  Data Hiding | | | |  |  |
|  | Objective:  This experiment introduces the students to the concept of Data Hiding using Python as a Programming Language. The concept of public and private data members and the role of accessor and mutator is explained | | | | | |  |
|  | **Name of Student:**  **Roll No: Sec.**  **Date of Experiment:** | | | | | |  |
|  | **Marks Obtained/Remarks:**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |  |

**Lab 07: Data Hiding**

We can access data fields via instance variables directly from an object.

*Example 1*

* import math
* class Circle:
* def \_\_init\_\_(self, radius=1):
* self.radius = radius
* def getPerimeter(self):
* return 2 \* self.radius \* math.pi
* def getArea(self):
* return self.radius \* self.radius \* math.pi
* c = Circle(5)
* c.radius = 5.4
* print(c.radius)

This may lead to the following two problems

1. Data may be tempered with
2. Class becomes difficult to maintain and vulnerable to bugs

The solution lies in not letting the client directly access the data fields. This concept is known as Data Hiding. It is implemented by defining data fields to be private. Private data fields and methods can be accessed within a class and cannot be accessed outside the class

**Private Data Fields in Python**

In Python private data fields and methods are defined with two leading underscore

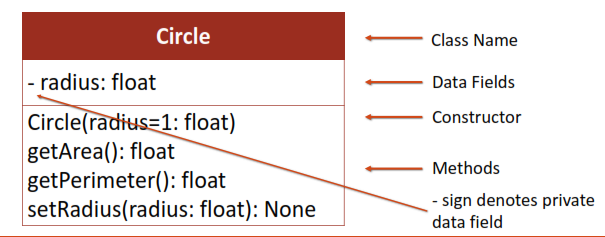
**Accessor and Mutator**

To make a data field accessible for the client, we provide a ***get*** method to return its value. And to enable a data field to be modified, we provide a ***set*** method to set a new value. ***get*** method is referred to as a getter or an accessor and ***set*** method is referred to as a setter or a mutator.

*Example 2*

* import math
* class Circle:
* def \_\_init\_\_(self, radius=1):
* **self.\_\_radius** = radius
* def getRadius(self):
* return self.\_\_radius
* def getPerimeter(self):
* return 2 \* self.\_\_radius \* math.pi
* def getArea(self):
* return self.\_\_radius \* self.radius \* math.pi
* c = Circle(5)
* print(c.\_\_radius)
* Traceback (most recent call last): File "main.py", line 16, in <module>
* print(c.\_\_radius)
* AttributeError: 'Circle' object has no attribute '\_\_radius'

**Private Fields in UML**



# Student Exercise

The **Fan** class

Design a class named Fan to represent a fan. The class contains:

■ Three constants named SLOW, MEDIUM, and FAST with the values 1, 2, and 3 to denote the fan speed.

■ A private int data field named speed that specifies the speed of the fan.

■ A private bool data field named on that specifies whether the fan is on (the default is False).

■ A private float data field named radius that specifies the radius of the fan.

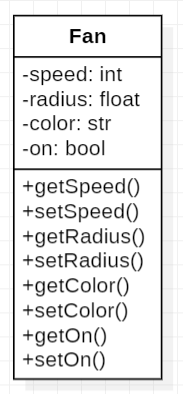
■ A private string data field named color that specifies the color of the fan.

■ The accessor and mutator methods for all four data fields.

■ A constructor that creates a fan with the specified speed (default SLOW), radius (default 5), color (default blue), and on (default False).

Exercise 1

Draw the UML diagram for the class and then implement the class.



**UML Diagram**

Exercise 2

Write a test program that creates two Fan objects. For the first object, assign the maximum speed, radius 10, color yellow, and turn it on. Assign medium speed, radius 5, color blue, and turn it off for the second object. Display each object’s speed, radius, color, and on properties.

**Code:**

class Fan:

SLOW = 1

MEDIUM = 2

FAST = 3

def \_\_init\_\_(self, speed = SLOW, radius = 5, color = "blue", on = False):

self.\_\_speed = speed

self.\_\_radius = radius

self.\_\_color = color

self.\_\_on = on

def getSpeed(self):

return self.\_\_speed

def setSpeed(self, speed):

self.\_\_speed = speed

def getRadius(self):

return self.\_\_radius

def setRadius(self, radius):

self.\_\_radius = radius

def getColor(self):

return self.\_\_color

def setColor(self, color):

self.\_\_color = color

def getOn(self):

return self.\_\_on

def setOn(self, on):

self.\_\_on = on

fan1 = Fan(Fan.FAST, 10, "yellow", True)

fan2 = Fan(Fan.MEDIUM,5 , "blue", False)

print("fan1's speed is", fan1.getSpeed(), "radius is", fan1.getRadius(), "color is", fan1.getColor(), "and on is", fan1.getOn())

print("fan2's speed is", fan2.getSpeed(), "radius is", fan2.getRadius(), "color is", fan2.getColor(), "and on is", fan2.getOn())

**Output:**

****

The **Account** class

Design a class named Account that contains:

■ A private int data field named id for the account.

■ A private float data field named balance for the account.

■ A private float data field named annualInterestRate that stores the current interest rate.

■ A constructor that creates an account with the specified id (default 0), initial balance (default 100), and annual interest rate (default 0).

■ The accessor and mutator methods for id, balance, and annualInterestRate.

■ A method named getMonthlyInterestRate() that returns the monthly interest rate.

■ A method named getMonthlyInterest() that returns the monthly interest.

■ A method named withdraw that withdraws a specified amount from the account.

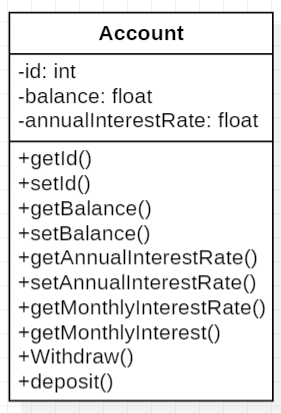
■ A method named deposit that deposits a specified amount to the account.

Exercise 3

Draw the UML diagram for the class, and then implement the class.

(Hint: The method getMonthlyInterest() is to return the monthly interest amount, not the interest rate. Use this formula to calculate the monthly interest: balance \* monthlyInterestRate. monthlyInterestRate is annualInterestRate / 12. Note that annualInterestRate is a percent (like 4.5%). You need to divide it by 100.)

**UML Diagram:**



**Code:**

class Account:

def \_\_init\_\_(self, id = 0, balance = 100, annualInterestRate = 0):

self.\_\_id = id

self.\_\_balance = balance

self.\_\_annualInterestRate = annualInterestRate

def getId(self):

return self.\_\_id

def setId(self, id):

self.\_\_id = id

def getBalance(self):

return self.\_\_balance

def setBalance(self, balance):

self.\_\_balance = balance

def getAnnualInterestRate(self):

return self.\_\_annualInterestRate

def setAnnualInterestRate(self, annualInterestRate):

self.\_\_annualInterestRate = annualInterestRate

def getMonthlyInterestRate(self):

return self.\_\_annualInterestRate / 12

def getMonthlyInterest(self):

return self.\_\_balance \* self.getMonthlyInterestRate()

def withdraw(self, amount):

self.\_\_balance -= amount

def deposit(self,amount):

self.\_\_balance += amount

Exercise 4

Write a test program that creates an Account object with an account id of 1122, a balance of $20,000, and an annual interest rate of 4.5%. Use the withdraw method to withdraw $2,500, use the deposit method to deposit $3,000, and print the id, balance, monthly interest rate, and monthly interest.

**Code:**

account = Account(1122, 20000, 4.5)

account.withdraw(2500)

account.deposit(3000)

print("id is", account.getId(), "balance is", account.getBalance(), "monthly interest rate is", account.getMonthlyInterestRate(), "monthly interest is", account.getMonthlyInterest())

**Output:**

****