

1 ICS 104 - Introduction to Programming in Python and C

1.1 Objects and Classes - Lab

2 Lab Objectives

- To understand the concepts of classes, objects and encapsulation
- To implement instance variables, methods and constructors
- To be able to design, implement and test your own classes

3 Worked Example

- **Problem Statement:** Your task is to write a class that simulates a bank account. Customers can deposit and withdraw funds. If sufficient funds are not available for withdrawal, a \$10 overdraft penalty is charged. At the end of the month, interest is added to the account. The interest rate can vary every month.

- **Step 1:** Get an informal list of the responsibilities of your objects.
- The following responsibilities are mentioned in the problem statement:
 - **Deposit funds.**
 - **Withdraw funds.**
 - **Add interest.**

- There is a hidden responsibility as well. We need to be able to find out how much money is in the account.
 - **Get balance.**

- **Step 2:** Specify the public interface.
 - To deposit or withdraw money, one needs to know the amount of the deposit or withdrawal:
 - `def deposit (self, amount):`

```
▪ def withdraw (self, amount):
```

- To add interest, one needs to know the interest rate that is to be applied:
 - `def addInterest (self, rate) :`
- Finally, we have
 - `def getBalance (self) :`

- Now we move to the constructor. The constructor should accept the initial balance of the account.
- It can be useful to allow for an initial zero balance using a default argument.
 - `def __init__ (self, initialBalance = 0.0) :`

- **Step 3:** Document the public interface:

```
## A bank account has a balance that can be changed by deposits and withdrawals.
#
class BankAccount :
    ## Constructs a bank account with a given balance.
    # @param initialBalance the initial account balance (default = 0.0)
    #
    def __init__(self, initialBalance = 0.0) :

    ## Deposits money into this account.
    # @param amount the amount to deposit
    #
    def deposit(self, amount) :

    ## Makes a withdrawal from this account, or charges a penalty if
    # sufficient funds are not available.
    # @param amount the amount of the withdrawal
    #
    def withdraw(self, amount) :

    ## Adds interest to this account.
    # @param rate the interest rate in percent
```

```
#
def addInterest(self, rate) :

## Gets the current balance of this account.
# @return the current balance
#
def getBalance(self) :
```

- **Step 4:** Determine instance variables.
- We need to store the bank balance:
 - `self._balance = initialBalance`

- Do we need to store the interest rate?
 - No — it varies every month, and is supplied as an argument to `addInterest`.
- What about the withdrawal penalty?
 - The problem description states that it is a fixed \$10, so we need not store it.
- If the penalty could vary over time, as is the case with most real bank accounts, we would need to store it somewhere (perhaps in a Bank object), but it is not our job to model every aspect of the real world.

In []:

```
1  ##
2  # This module defines a class that models a bank account.
3  #
4
5  ## A bank account has a balance that can be changed by deposits and withdrawals.
6  #
7  class BankAccount :
8      ## Constructs a bank account with a given balance.
9      # @param initialBalance the initial account balance (default = 0.0)
10     #
11     def __init__(self, initialBalance = 0.0) :
12         self._balance = initialBalance
13
14     ## Deposits money into this account.
15     # @param amount the amount to deposit
16     #
17     def deposit(self, amount) :
18         self._balance = self._balance + amount
19
20     ## Makes a withdrawal from this account, or charges a penalty if
21     # sufficient funds are not available.
22     # @param amount the amount of the withdrawal
23     #
24     def withdraw(self, amount) :
25         PENALTY = 10.0
26         if amount > self._balance :
27             self._balance = self._balance - PENALTY
28         else :
29             self._balance = self._balance - amount
30
31     ## Adds interest to this account.
32     # @param rate the interest rate in percent
33     #
34     def addInterest(self, rate) :
35         amount = self._balance * rate / 100.0
36         self._balance = self._balance + amount
37
38     ## Gets the current balance of this account.
39     # @return the current balance
40     #
41     def getBalance(self) :
42         return self._balance
```

43
44

In []:

```
1  ##
2  # This program tests the BankAccount class.
3  #
4  # from bankaccount import BankAccount
5
6  harrysAccount = BankAccount(1000.0)
7  harrysAccount.deposit(500.0) # Balance is now $1500
8  harrysAccount.withdraw(2000.0) # Balance is now $1490
9  harrysAccount.addInterest(1.0) # Balance is now $1490 + 14.90
10 print("%.2f" % harrysAccount.getBalance())
11 print("Expected: 1504.90")
12
13
```

4 Exercises

4.1 Exercise # 1

Define a class `Point` that represents a point in 2 – D plane. The point has x and y coordinates. Define the following:

- A constructor to initialize the x , y coordinates.
- A method `translate(self, dx, dy)` to translate the point object dx , and dy units in x and y directions, respectively.
- A method `distanceTo (self, point2)` to return the distance between the point referenced by `self` and `point2`.
- `getX(self)` to return the value of x coordinate.
- `getY(self)` to return the value of y coordinate

Test the above class by:

- Creating 2 point objects; one with (3,5) as x,y coordinates; the second with (-10,30) as x,y coordinates.
- Move the first point 5.5 units in x direction and -12.5 units in y direction using `translate` method
- Find the distance between the 2 points in their current location using `distanceTo` method

A Sample output resulting from running the above test class is shown below

```
new coordinates of point1= (8.5 , -7.5)
```

```
Coordinates of point 2 = (-10.0 , 30.0)
```

```
Distance between the 2 points = 41.82
```

In [23]:

```
1  # YOUR CODE HERE
2
3  from math import *
4
5  def main():
6      point1 = Point(3,5)
7      point2 = Point(-10,30)
8      updatedCoordinates = point1.translate(5.5,-12.5)
9      print("New coordinates of point1 = ", updatedCoordinates)
10     distance = point1.distanceTo(point2)
11     print("Coordinates of point 2 = " , point2._point)
12     print("Distance between the 2 points = %.2f" %distance)
13
14     class Point():
15         def __init__(self,initialx = 0.0, initially = 0.0):
16             self._point = (initialx,initialy)
17
18         def translate(self, dx,dy):
19             self._point = (self._point[0] + dx, self._point[1] + dy)
20             return self._point
21
22         def distanceTo(self, point2):
23             distance = sqrt((self._point[0]-point2._point[0])**2 + (self._point[1] - point2._point[1])**2)
24             return distance
25
26         def getX(self):
27             return self._point[0]
28
29         def getY(self):
30             return self._point[1]
31
32     main()
33
```

New coordinates of point1 = (8.5, -7.5)
Coordinates of point 2 = (-10, 30)
Distance between the 2 points = 41.82

4.2 Exercise # 2

Implement a class Portfolio. This class has two objects, checking and saving, of the type `bankAccount` that was developed in the worked example. Initialize the 2 bank accounts with 0 initial balance.

- Implement four methods
 - `def deposit (self, amount, account)`
 - `def withdraw (self, amount, account)`
 - `def transfer (self, amount, account)`
 - `def getBalance (self, account)`
- Here the `account` string is "S" or "C" for Saving and Checking, respectively. For the `deposit` or `withdraw`, it indicates which account is affected. For a `transfer`, it indicates the account from which the money is taken; the money is automatically transferred to the other account.
- To test your class:
 - create one Portfolio object
 - deposit 10000 in its checking account
 - transfer 5000 from checking account to saving account
 - withdraw 2500 from checking account
 - display the balance of both accounts
- A run for the above test program will result in the following output

```
Saving balance = 5000.0  
Checking balance = 2500.0
```

In [24]:

```
1 def main():
2     myAccount = portfolio()
3     myAccount.deposit(10000, "C")
4     myAccount.transfer(5000, "C")
5     myAccount.withdraw(2500, "C")
6     print("Saving balance = ", myAccount.getBalance("S"))
7     print("Checking Balance = ", myAccount.getBalance("C"))
8
9
10 class BankAccount :
11
12     def __init__(self, initialBalance = 0.0) :
13         self._balance = initialBalance
14
15     def deposit(self, amount) :
16         self._balance = self._balance + amount
17
18     def withdraw(self, amount) :
19         PENALTY = 10.0
20         if amount > self._balance :
21             self._balance = self._balance - PENALTY
22         else :
23             self._balance = self._balance - amount
24
25
26     def addInterest(self, rate) :
27         amount = self._balance * rate / 100.0
28         self._balance = self._balance + amount
29
30
31     def getBalance(self) :
32         return self._balance
33
34 class portfolio:
35
36     def __init__(self):
37         self._savebalance = BankAccount()
38         self._checkbalance = BankAccount()
39
40     def deposit(self, amount, account):
41         if account == "S":
42             self._savebalance.deposit(amount)
```

```
43         if account == "C":
44             self._checkbalance.deposit(amount)
45
46     def withdraw(self, amount, account):
47         if account == "S":
48             self._savebalance.withdraw(amount)
49         if account == "C":
50             self._checkbalance.withdraw(amount)
51
52     def transfer(self, amount, account):
53         if account == "S":
54             self._savebalance.withdraw(amount)
55             self._checkbalance.deposit(amount)
56         if account == "C":
57             self._checkbalance.withdraw(amount)
58             self._savebalance.deposit(amount)
59
60     def getBalance(self, account):
61         if account == "S":
62             return self._savebalance.getBalance()
63         if account == "C":
64             return self._checkbalance.getBalance()
65
66     main()
```

Saving balance = 5000.0

Checking Balance = 2500.0