Slide Type Slide

1 ICS 104 - Introduction to Programming in Python and C

1.1 Objects and Classes - Lab

Slide Type Slide

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

Slide Type Slide

3 Worked Example

Slide Type Fragment

- 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.

Slide Type Slide

- **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.**

Slide Type Fragment

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

Slide Type Slide

- **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):`

Slide Type Fragment

- 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) :`

Slide Type Fragment

- 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 __int__ (self, initialBalance = 0.0) :`

Slide Type Slide

- **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
```

Slide Type Fragment

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

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

Slide Type Slide

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

Slide Type Fragment

- 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 [3]:

Slide Type: Slide

```
##
#  This module defines a class that models a bank ac
#

## A bank account has a balance that can be changed
#
class BankAccount :
   ## Constructs a bank account with a given balance
   #  @param initialBalance the initial account bala
   #
   def __init__(self, initialBalance = 0.0) :
      self._balance = initialBalance

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

   ## Makes a withdrawal from this account, or charg
   #  sufficient funds are not available.
   #  @param amount the amount of the withdrawal
   #
   def withdraw(self, amount) :
      PENALTY = 10.0
      if amount > self._balance :
         self._balance = self._balance - PENALTY
      else :
         self._balance = self._balance - amount

   ## Adds interest to this account.
   #  @param rate the interest rate in percent
   #
   def addInterest(self, rate) :
      amount = self._balance * rate / 100.0
      self._balance = self._balance + amount

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


```

In [6]:

Slide Type: Fragment

```
##
#  This program tests the BankAccount class.
#
# from bankaccount import BankAccount

harrysAccount = BankAccount(1000.0)
harrysAccount.deposit(500.0)  # Balance is now $1500
harrysAccount.withdraw(2000.0)  # Balance is now $14
harrysAccount.addInterest(1.0)  # Balance is now $14
print("%.2f" % harrysAccount.getBalance())
print("Expected: 1504.90")


```

```
1504.90
Expected: 1504.90
```

Slide Type: Slide

4 Exercises

Slide Type: Slide

- 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 [2]:

Slide Type: Fragment

```
# Exercise # 1 - Source Code
import math

class Point:
    def __init__(self,x=0.0 ,y=0.0):
        self._X = x
        self._Y = y

    def translate(self, dx,dy):
        self._X = self._X + dx
        self._Y = self._Y + dy

    def distanceTo (self, point2):
        return math.sqrt(((point2._X-self._X)**2) + ((point2._Y-self._Y)**2))

    def getX(self) :
        return self._X

    def getY(self):
        return self._Y

FirstPoint  = Point(3,5)
SecondPoint = Point(-10,30)

FirstPoint.translate(5.5,-12.5)

print("new coordinates of point1= ("+str(FirstPoint.getX())+" , "+str(FirstPoint.getY())+")")
print("Coordinates of point 2 = (" +str(SecondPoint.getX())+" , "+str(SecondPoint.getY())+")")
print("Distance between the 2 points = %.2f" % FirstPoint.distanceTo(SecondPoint))
```

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

Slide Type: Slide

- 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 [9]:

Slide Type Fragment

```python
# Exercise # 2 - Source Code
class Portfolio:
    def __init__(self):
        self._saving = BankAccount()
        self._checking = BankAccount()

    def deposit(self, amount, account):
        if account =="S":
            self._saving.deposit(amount)

        elif account == "C":
            self._checking.deposit(amount)

    def withdraw(self, amount, account):
        if account == "S":
            self._saving.withdraw(amount)
        elif account == "C":
            self._checking.withdraw(amount)

    def transfer (self, amount, account):
        if account == "S":
            self._saving.withdraw(amount)
            self._checking.deposit(amount)

        elif account == "C":
            self._checking.withdraw(amount)
            self._saving.deposit(amount)

    def  getBalance (self, account):
        if account == "S":
            return self._saving.getBalance()
        elif account =="C":
            return self._checking.getBalance()
#---------------------------END OF DEF -------------------------------------------------#
trial = Portfolio()
trial.deposit(10000,"C")
trial.transfer(5000,"C")
trial.withdraw(2500,"C")
print("Saving Balance = ",trial.getBalance("S"))
print("Checking Balance = ",trial.getBalance("C"))
```

```
Saving Balance =  5000.0
Checking Balance =  2500.0
```

In [ ]:

Slide Type

```python

```