National University of Computer and Emerging Sciences

Lab Manual 12
Object Oriented Programming

Course Instructor	
Lab Instructor (s)	
Section	
Semester	Spring 2024

Department of Computer Science
FAST-NU, Lahore, Pakistan

Objectives:

- ✓ Use base class pointers to call derived class objects
- ✓ Virtual functions and destructors
- ✓ See polymorphism in action

Exercise 1:

- Create a class called `Account`.
- It has data member:
  - Account Number.
  - Account Balance.
- And suitable setter/getter for data.
- And Print(), Debit(float), Credit(float) as member functions (virtual).
  - override Debit and Credit functions according to derived classes.

Exercise 2:

- Create a class called `CurrentAccount` i-e: `CurrentAccount(is-a) Account`
- It has data member:
  - Service Charges (To be charged during credit if account balance is less than min balance)
  - Minimum Balance
- Override print() as created in above class which displays:
  - Account Number, Account Balance, Minimum Balance, Service Charges
- Modify the definition of the print() so that it displays a suitable message containing above info.
- Similarly override credit(float), debit(float) functions such that credit(float) simply add amount to the Account Balance and debit(float) checks if the amount to be debited is within the range of Account Balance, and further if the amount is account balance is less than min balance standard charges would also be deducted.
- Create a class called `SavingAccount` i-e: `CurrentAccount(is-a) Account`
- It has data member:
  - Interest Rate.
- Override print() as created in parent class which displays:
  - Account Number, Account Balance, Interest Rate
- Modify the definition of the print() so that it displays a suitable message containing above info.
- Similarly override credit(float), debit(float) functions such that credit(float) simply add amount to the Account Balance and debit(float) checks if the amount to be debited is within the range of Account Balance.
- Write a suitable main function of your program, in which you have to Call the functions (print, debit, credit) of `CurrentAccount` class according to the type of object. To accomplish this, we handle the keyword virtual to the declaration of the `print()` method in the `Base` class. Make sure that print function of the calling object is called.

Specifying a function as `virtual` makes sure that whenever we use a base class pointer pointing to an object of a derived class to call a function, the definition of the method declared in the derived class is used.

Exercise 3:

In the above exercises, we have seen a very simple implementation of Polymorphism. The real power of this feature is realized when we have a collection of objects of multiple derived classes and we use a pointer of the base class to call their respective overloaded methods. A `SavingAccount` is an `Account` too. Let's see how we can use an array of base class pointers to utilize the essence of polymorphism.

- Modify the `main()` function as shown below.
- Compile, execute and paste the output in the space given below.

```
//Array of base pointers
Account ** alist = new Account*[5];
alist[0] = new SavingAccount;
alist[1] = new CurrentAccount;
alist[2] = new Account;
….
//Print data of all accounts polymorphic behavior
for(int i=0; i<5 ;i++)
alist[i]->print();
//credit and debit polymorphic behavior
alist[0]->credit(50);
alist[2]->debit(333);
```

Exercise 4:

Although things seem to be fine on the surface, there is a problem in the program we just wrote. To observe this problem, we must add destructors for all classes. Paste the following inline definitions of the destructors in their corresponding classes, execute the program and paste the output below.

```
~Account()      { cout << "~Account Destroyed ."<<endl; }
```

```
~CurrentAccount() { cout << "~ CurrentAccount Destroyed."<<endl;
     }
```

```
~SAvingAccount()     { cout << "~ SAvingAccount
Destroyed."<<endl;        }
```

Can you see what went wrong? When using delete to deallocate memory, only the base class destructor is called where as the derived class destructor is not called at all. Although this is fine in the example we are using here but it will create memory leaks if there are any dynamically allocated variables in any of the derived class. To avoid this we declare the base class destructor as `virtual`. Doing this will make sure that the derived class destructor is called even if you are using a base class pointer to call the destructor. Now change the definition of the base class destructor to make it virtual, execute the program and observe the sequence of calling destructor. Make sure you can see the derived class destructors being called in the output. Copy the sequence of destructor called and write as comment(s) in your file.