**National University of Computer and Emerging Sciences**

Logo

Description automatically generated with medium confidence

**Lab Manual 10**

**Object Oriented Programming**

|  |  |
| --- | --- |
| Course Instructor | Miss Abeeda |
| Lab Instructor (s) | Siddiqua Nayyer  Dilawar Shabbir |
| Section | A |
| Semester | Spring 2021 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

**TASK 1** (More on account creation)

**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 as per 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: SavingAccount(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.

What should be done to avoid any such issues?

## **TASK 2: (Playing with Destructor)**

Write a program to practice memory management alongside polymorphism.

You are not allowed to change function prototypes

Implement following class structure. In addition to this you are to implement destructors in all classes below to ensure dynamically allocated memory is properly deleted.

|  |  |  |
| --- | --- | --- |
| Person (Base Class) | Employee (Derived) | Student (Derived) |
| //member variables  {  **Private:**  String \*fullName;  Int \*height;  **Public:**  **Person(string name,Int height)** //constructor  **Virtual void printInfo();(1)**  //this function is to print all private varaibles  //destructor to be implemented alongside type of class eg cout<<”person destructor” (5)    } | //member variables  {  **Private:**  String \*departement;  Int \*ID;  **Public:**  **Employee(string name,Int height,string departement,Int id) : Person( name, height)**  //constructor  **void printInfo();**  //this function is to print all private variables alongside type of class  //destructor to be implemented alongside type of class eg cout<<”employee destructor”  } | //member variables  {  **Private:**  String \*schoolName;  **Public:**  Student **(string name,Int height, string SchoolName) : Person( name, height)**  //constructor  **void printInfo();**  //this function is to print all private variables alongside type of class  //destructor to be implemented alongside type of class eg cout<<”student destructor”  } |

Main Program:

1. Create a array of base class pointers of size 2. (2d Array)
2. Initialize each of the base class pointer with employee and student object respectively.
3. Run a loop to call printInfo on the array created.
4. Call delete operator on the array of base class to test the memory management.
5. Show polymorphic behavior on non-member functions such as ostream operators.

**TASK 3**

***Base class:***

class IShape{

public:

// Pure virtual functions

virtual void Area() = 0;

virtual void Perimeter() = 0;

virtual void GetDiagonal() = 0;

virtual ~IShape();

};

***Derived class:***

class Rectengle: public IShape{

int length;

int width

public:

// Constructor (s)

virtual void printRectengle(){ ////// }

virtual ~A(){}

// Must implement all Pure virtual functions.

// What will happen if you skip implementation of any function ????? Write in comments

};

Complete the definition of classes and write and suitable main showing all the functionality of your class(es).