Aug 20, 2015

COMPUTER PROGRAMMING

  FAST NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Lab 2

Abdul aziz

|  |  |
| --- | --- |
| **Lab Instructor** | **Mr. Abdul Aziz** |
| **Course** | **Computer Programming Lab** |
| **Duration** | **2hrs** |

**Objectives:**

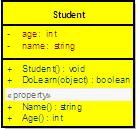
* In this lab, following topics will be covered
* Basics of OOP
* Class and Object
* Encapsulation
* Inheritance
* Polymorphism and virtual functions

# Basics of OOP

It stands for Object Oriented Programming. It’s a design philosophy. **O**bject-**O**riented **P**rogramming (OOP) uses a different set of programming languages than old procedural programming languages (C, Pascal*,* etc.). Everything in OOP is grouped as self-sustainable "objects". Hence, you gain reusability by means of main object-oriented programming concepts.

# Class

A class is simply a representation of a type of object. It is the blueprint, or plan, or template that describes the details of an object. In class the individual objects are created. Class is composed of three things: a name, attributes, and operations.



class classname

{

*Access - Specifier:  
 Member Varibale Declaration;  
 Member Function Declaration;*

}

# Object

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. An object can be considered as a "thing" that can perform a set of related activities. The set of activities that the object performs defines the object's behavior. For example, Students (object) can give their name or address.

In pure OOP terms an object is an instance of a class.

/\* Simple Class Example Program with Object Creation In C++

\*/

#include <iostream>

#include<conio.h>

using namespace std;

// Class Declaration

class person

{

//Access - Specifier

public:

//Varibale Declaration

  string name;

  int number;

};

//Main Function

int main()

{

    // Object Creation For Class

    person obj;

    //Get Input Values For Object Varibales

    cout<<"Enter the Name :";

    cin>>obj.name;

    cout<<"Enter the Number :";

    cin>>obj.number;

    //Show the Output

    cout << obj.name << ": " << obj.number << endl;

       return 0;

}

# Encapsulation

Encapsulation is the packing of data and functions into a single component. In programming languages Encapsulation means that a class publishes only what is needed for others to use it, and no more. This is called information hiding and it means classes can totally change their internals without having an effect on any of their users. In general, encapsulation is the inclusion of one thing within another thing so that the included thing is not apparent. This is to say, if a function or a method inside this class is private, only objects of this class can access the method.

\*/ A simple program using the concept of Encapsulation \*/

#include <iostream>

using namespace std;

class Adder

{

private: int total; // hidden data from outside world

public:

Adder(int i = 0) // constructor

{

total = i;

}

// interface to outside world

void addNum(int number)

{

total += number;

}

// interface to outside world

int getTotal()

{

return total;

};

};

//Now Main Class Starts

int main( )

{

Adder a; // Object Creation

a.addNum(10);

a.addNum(20);

a.addNum(30);

cout << "Total is " << a.getTotal() <<endl;

return 0;

}

# Inheritance

The concept of inheritance is one of the key features of an object-oriented programming language. Inheritance allows a programmer to define a general class, and then later define more specific classes that share or inherit all of the properties of the more general class. This allows the programmer to save time and energy that might otherwise be spent writing duplicate code.

In [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) (OOP), inheritance is when an [object](https://en.wikipedia.org/wiki/Object_(computer_science)) or [class](https://en.wikipedia.org/wiki/Class_(computer_programming)) is based on another object ([prototypal inheritance](https://en.wikipedia.org/wiki/Prototype-based_programming)) or class ([class-based inheritance](https://en.wikipedia.org/wiki/Class-based_programming)), using the same implementation (inheriting from an object or class) specifying implementation to maintain the same behavior (realizing an interface; inheriting behavior). It is a mechanism for [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and to allow independent extensions of the original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a [hierarchy](https://en.wikipedia.org/wiki/Hierarchy).

\*/ A simple program using the concept of Inheritance \*/

#include <iostream>

using namespace std;

// Base class

class Shape

{

protected:

int width;

int height;

public:

void setWidth(int w)

{

width = w;

}

void setHeight(int h)

{

height = h;

}

};

// Derived class

class Rectangle: public Shape

{

public:

int getArea()

{

return (width \* height);

}

};

// Main Class

int main(void)

{

Rectangle Rect; // Object Creation

Rect.setWidth(5);

Rect.setHeight(7);

// Print the area of the object.

cout << "Total area: " << Rect.getArea() << endl;

return 0;

}

**5.1 Access Control and Inheritance:**

A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

We can summarize the different access types according to who can access them in the following way:

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **public** | **protected** | **private** |
| Same class | yes | yes | yes |
| Derived classes | yes | yes | no |
| Outside classes | yes | no | no |

A derived class inherits all base class methods with the following exceptions:

* Constructors, destructors and copy constructors of the base class.
* Overloaded operators of the base class.
* The friend functions of the base class.

# Polymorphism

Polymorphism is an object-oriented programming concept that refers to the ability of a variable, function or object to take on multiple forms Polymorphism is a generic term that means 'many shapes'. More precisely Polymorphisms mean the ability to request that the same operations be performed by a wide range of different types of things.

For example, integers and floats are implicitly polymorphic since you can add, subtract, multiply and so on, irrespective of the fact that the types are different. They're rarely considered as objects in the usual term.

\*/ A simple program using the concept of Polymorphism \*/

#include<iostream>

using namespace std;

class Base

{

public:

virtual void show() { cout<<" In Base \n"; }

};

class Derived: public Base

{

public:

void show() { cout<<"In Derived \n"; }

};

class Derived2: public Base

{

public:

void show() { cout<<"In Derived2 \n"; }

};

int main(void)

{

Base \*bp = new Derived;

bp->show(); // RUN-TIME POLYMORPHISM

bp = new Derived2;

bp->show(); // RUN-TIME POLYMORPHISM

return 0;

}

**Exercise**

1. Write a program to calculate square of a number entered by user using encapsulation.
2. Write a program to implement classes and objects for the student data

* Define a constructor to initialize the data members
* Create object for the class to access the member function
* Take marks of three subjects
* Calculate the average
* Display the result

1. Write a C++ program to create inheritance. Create classes A1, A2, A3. Having following data members.

* Class A: Name and Father’s Name
* Class B: Age and Height
* Class C: Department