Object Oriented Programming (CT-260) Lab 02

Introduction to Object Oriented Programming with C++

Objectives

The objective of this lab is to familiarize students with object-oriented programming. By the end of this lab, students will be able to understand the concepts of classes, objects, and access modifiers.

Tools Required

DevC++ IDE / Visual Studio / Visual Code

Course Coordinator –
Course Instructor –
Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

Object Oriented Programming

The fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data. Such a unit is called an object. An object's functions, called member functions in C++, typically provide the only way to access its data.

To read a data item in an object, member function in the object is called, which in turn will access the data and return the value. The data cannot be accessed directly. The data is hidden, so it is safe from accidental alteration. Data encapsulation and data hiding are key terms in object-oriented languages. This simplifies writing, debugging, and maintaining the program.

A C++ program typically consists of a number of objects, which communicate with each other by calling one another's member functions. Member functions are also called methods in some other object-oriented (OO) languages. Also, data items are referred to as attributes or instance variables, features. The major elements of object-oriented languages in general, and C++ in particular are:

- ✓ Objects
- ✓ Classes
- ✓ Inheritance.
- ✓ Reusability
- ✓ Polymorphism and Overloading.

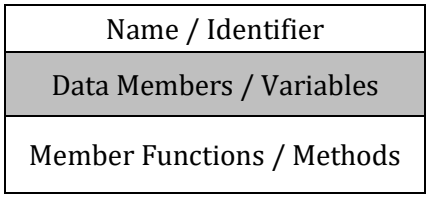
In C++, everything is linked to classes and objects, as well as their properties and functions. An automobile, for example, is an object in real life. The automobile has attributes like weight and color, as well as methods like drive and brake. Attributes and methods are essentially the class's variables and functions. These are commonly known as "class members."

Class

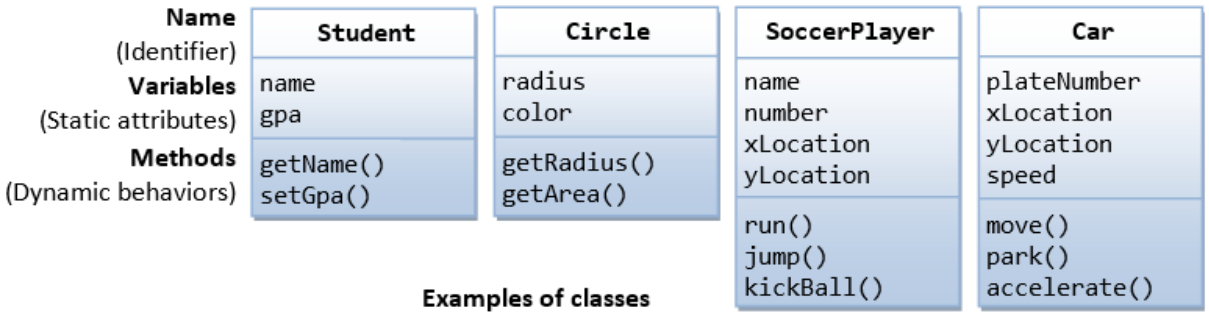
A class is a definition of objects of the same kind. In other words, a class is a blueprint, template, or prototype that defines and describes the static attributes and dynamic behaviors common to all objects of the same kind.

Instance

An instance is a realization of a particular item of a class. In other words, an instance is an instantiation of a class. All the instances of a class have similar properties, as described in the class definition. For example, you can define a class called "Student" and create three instances of the class "Student" for "Paul", "Peter" and "Pauline". The term "object" usually refers to instance. But it is often used quite loosely, which may refer to a class or an instance. A class can be visualized as a three-compartment box:

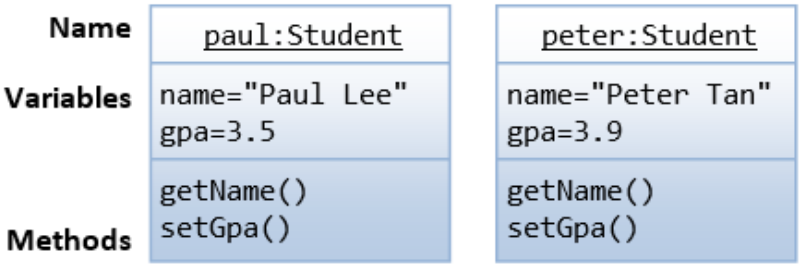


Example:



Examples of classes

The following figure shows two instances of the class Student, identified as "paul" and "peter".



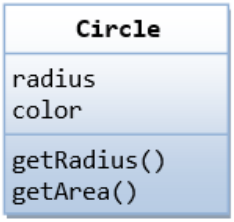
Two instances - paul and peter - of the class Student

Class Definition

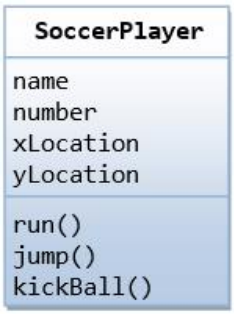
In C++, we use the keyword class to define a class. There are two sections in the class declaration:

- ✓ private
- ✓ public

```
class Circle                    // classname
{
private:
   double radius;               // Data members
   string color;
public:
    double getRadius( );        // Member functions
    double getArea( );
}
```




```
class SoccerPlayer              // classname
{
private:
   int number;                  // Data members
   string name;
   int x, y;
public:
   void run();                  // Member functions
   void kickBall();
}
```



Creating Instances of a Class

To create an instance of a class, you have to:

1. Declare an instance identifier (name) of a particular class.
2. Invoke a constructor to construct the instance (i.e., allocate storage for the instance and initialize the variables).

For examples, suppose that we have a class called Circle, we can create instances of Circle as follows:

Construct 3 instances of the class Circle: c1, c2, and c3	
`Circle c1(1.2, "red");`	// radius, color
`Circle c2(3.4);`	// radius, default color
`Circle c3;`	// default radius and color

Dot (.) Operator

To reference a member of an object (data member or member function), you must:

1. First identify the instance you are interested in, and then
2. Use the dot operator (.) to reference the member, in the form of *instanceName.memberName.*

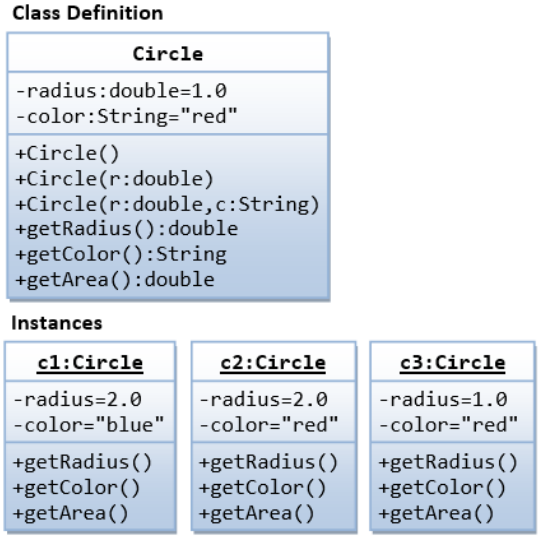
For example, suppose that we have a class called Circle, with two data members (radius and color) and two functions (getRadius( ) and getArea( )). We have created three instances of the class Circle, namely, c1, c2 and c3. To invoke the function getArea( ), you must first identity the instance of interest, say c2, then use the dot operator, in the form of c2.getArea( ), to invoke the getArea( ) function of instance c2.

Example:

```
//Declare and construct instances c1 and c2 of the class Circle
Circle c1(1.2, "blue");
Circle c2(3.4, "green");
//Invoke member function via dot operator
cout << c1.getArea() << endl;
cout << c2.getArea() << endl;
//Reference data members via dot operator
c1.radius = 5.5;
c2.radius = 6.6;
```

Calling getArea( ) without identifying the instance is meaningless, as the radius is unknown (there could be many instances of Circle - each maintaining its own radius).

Putting them together an OOP Example



A class called Circle is to be defined as illustrated in the class diagram. It contains two data members: radius (of type double) and color (of type String); and three member functions: getRadius(), getColor(), and getArea(). Three instances of Circles called c1, c2, and c3 shall then be constructed with their respective data members, as shown in the instance diagrams.

Public" vs. "Private" Access Control Modifiers

An access control modifier can be used to control the visibility of a data member or a member function within a class. We begin with the following two access control modifiers:

1. **public:** The member (data or function) is accessible and available to all in the system.
2. **private:** The member (data or function) is accessible and available within this class only.

For example, in the above Circle definition, the data member radius is declared private. As the result, radius is accessible inside the Circle class, but NOT outside the class.

In other words, you cannot use "c1.radius" to refer to c1's radius in main( ).

Try inserting the statement *"cout << c1.radius;"* in main( ) and observe the error message.

Encapsulation in OOP

A class encapsulates the static attributes and the dynamic behaviors into a "3-compartment box". Once a class is defined, you can seal up the "box" and put the "box" on the shelve for others to use and reuse. Anyone can pick up the "box" and use it in their application. This cannot be done in the traditional procedural-oriented language like C, as the static attributes (or variables) are scattered over the entire program and header files. You cannot "cut" out a portion of C program, plug into another program and expect the program to run without extensive changes.

Data member of a class are typically hidden from the outside world, with private access control modifier. Access to the private data members e.g., are provided via public assessor functions, getRadius( ) and getColor( ).

This follows the principle of information hiding. That is, objects communicate with each other using well-defined interfaces (public functions). Objects are not allowed to know the implementation details of

Getters and Setters

To allow other to read the value of a private data member says xxx, you shall provide a get function (or getter or accessor function) called getXxx( ). A getter need not expose the data in raw format. It can process the data and limit the view of the data others will see. Getters shall not modify the data member.

To allow other classes to modify the value of a private data member says xxx, you shall provide a set function (or setter or mutator function) called setXxx(). A setter could provide data validation (such as range checking), and transform the raw data into the internal representation.

For example, in our Circle class, the data member's radius and color are declared private. That is to say, they are only available within the Circle class and not visible outside the

Circle class - including main ( ). You cannot access the private data members radius and color from the main( ) directly – via says c1.radius or c1.color. The Circle class provides two public accessor functions, namely, getRadius( ) and getColor( ). These functions are declared public. The main( ) can invoke these public accessor functions to retrieve the radius and color of a Circle object, via says c1.getRadius( ) and c1.getColor( ).

There is no way you can change the radius or color of a Circle object, after it is constructed in main( ). You cannot issue statements such as c1.radius = 5.0 to change the radius of instance c1, as radius is declared as private in the Circle class and is not visible to other including main( ). If the designer of the Circle class permits the change the radius and color after a Circle object is constructed, he has to provide the appropriate setter, e.g.,

```
// Setter for color
void setColor(string c) {
     color = c;
}

// Setter for radius
void setRadius(double r) {
     radius = r;
}
```

Example:

```
#include <iostream>
using namespace std;
class firstprogram {
     private: // we declare a as private to hide it from outside
       int number1;
     public:
     void set(int input1){//set() function to set the value of a
        number1 = input1;
     }
     int get() { // get() function to return the value of a
       return number1;
     }
  };
// main function
int main() {
     firstprogram myInstance;
     myInstance.set(10);
     cout << myInstance.get() << endl;
     return 0;
}
```

Exercise

1. Write a program in which a class named student has member variables name, roll_no, semester and section. Create 4 objects of this class to store data of 4 different students, now display data of only those students who belong to section A.
2. You are a programmer for the ABC Bank assigned to develop a class that models the basic workings of a bank account. The class should perform the following tasks:
   - Save the account balance.
   - Save the number of transactions performed on the account.
   - Allow deposits to be made to the account.
   - Allow with drawls to be taken from the account.
   - Report the current account balance at any time.
   - Report the current number of transactions at any time.

   Menu
   1. Display the account balance
   2. Display the number of transactions
   3. Display interest earned for this period
   4. Make a deposit
   5. Make a withdrawal
   6. Exit the program
3. Create a class called Employee that includes three pieces of information as data members—a first name (type char* (DMA)), a last name (type string) and a monthly salary (type int). Your class should have a setter function that initializes the three data members. Provide a getter function for each data member. If the monthly salary is not positive, set it to 0. Write a test program that demonstrates class Employee's capabilities. Create two Employee objects and display each object's yearly salary. Then give each Employee a 10 percent raise and display each Employee's yearly salary again. Identify and add any other related functions to achieve the said goal.
4. Write C++ code to represent a hitting game by using OOP concept. The details are as follows: This game is being played between two teams (i.e. your team and the enemy team). The total number of players in your team is randomly generated and stored accordingly. The function generates a pair of numbers and matches each pair. If the numbers get matched, the following message is displayed: "Enemy got hit by your team!" Otherwise, the following message is displayed: "You got hit by the enemy team!" The number of hits should be equal to the number of players in your team. The program should tell the final result of your team by counting the hits of both the teams. Consider the following sample output: