Introduction to Information & Communication Technologies

CL-1000

Lab 10

Introduction to Object-Oriented Programming | OOP

Part - 01

National University of Computer & Emerging Sciences – NUCES – Karachi

National University of Computer & Emerging Sciences – NUCES – Karachi

Course Code: CL-1000	Introduction to Information & Communication Technologies

Contents

1. Classes & Objects

1.1 Classes

A **class** is a **programmer-defined data type** that describes what an object of the class will look like when it is created. It consists of a set of **variables** and a **set of functions**.

We can think of **class** as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. **House** is the **object**.

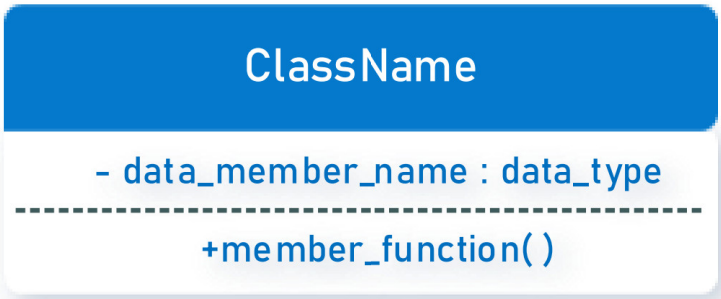
As, many houses can be made from the same description, we can create many **objects** from a **class**.

Classes are created using the keyword **class**. A **class** declaration defines a new type that links code and data. This new type is then used to declare **objects** of that **class**.

A **class** is a user defined data-type which has data members and member functions. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behaviour of the objects in a **class**.

A **class** member can be defined as **public**, **private** or **protected**. By default, members would be assumed as **private**.

In the **UML**, a class icon can be subdivided into compartments. The top compartment is for the name of the **class**, the second is for the variables of the **class**, and the third is for the methods of the **class**.



1.2 Class Name

By convention, the name of a user-defined class begins with a capital letter and, for readability, each subsequent word in the class name begins with a capital letter.

1.3 Data Members

Consider the attributes of some real-world objects:

- `RADIO` – station setting, volume setting.
- `CAR` – speedometer readings, amount of gas in its tank and what gear it is in.

These attributes form the data in our program. The values that these attributes take (the blue colour of the petals, for example) form the state of the object.

1.4 Member Functions

Consider the operations of some real-world objects:

- **RADIO** – setting its station and volume (invoked by the person adjusting the radio's controls)
- **CAR** – accelerating (invoked by the driver), decelerating, turning and shifting gears.

These operations form the functions in program. Member functions define the class's behaviors.

1.5 Objects

In **C++,** when we define a variable of a **class**, we call it **instantiating** the **class**. The variable itself is called an **instance** of the **class**. A variable of a **class** type is also called an **object**. Instantiating a variable allocates memory for the **object**.

Syntax to define Object in C++:

```
className objectVariableName;
```

```
Radio r;
Car c;
```

1.6 Accessing Public Data Members

The **public** data members of objects of a **class** can be accessed using the direct member access operator (**.**). However, the **private** data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

1.7 Accessing Private Data Members

To access, use and initialize the **private** data member you need to create **getter** and **setter** functions, to get and set the value of the data member.

The **setter** function will set the value passed as argument to the **private** data member, and the **getter** function will return the value of the **private** data member to be used. Both **getter** and **setter** function must be defined **public**.

```
#include "iostream"
using namespace std;

class Student{

    private:
        int rollNo; // private data member

    public:
        // public function to get value of rollNo - getter
        int getRollNo(){
            return rollNo;
        }
        // public function to set value for rollNo - setter
        void setRollNo(int i){
            rollNo = i;
```

```
        }
};

int main(){
    Student A;
    A.rollNo = 1;           // compile time error
    cout << A.rollNo;       // compile time error

    A.setRollNo(1);         // rollNo initialized to 1
    cout << A.getRollNo();  // output will be 1
}
```

```
#include<iostream>
using namespace std;

class Shape{

// as private so object.height and object.width is inaccessible

    int height, width;
    public:

        //setters
        void setHeight(int h){
            height = h;
        }

        void setWidth(int w){
            width = w;
        }

        //getters
        int getHeight(){
            return height;
        }

        int getWidth(){
            return width;
        }
};

int main(){
    Shape shape;

    // setters
    shape.setHeight(5);
    shape.setWidth(2);

    // getters
    cout << "The Height is : " << shape.getHeight() << endl;
```

```
    cout << "The Width is : " << shape.getWidth() << endl;
    // shape.height or shape.width wont work as they are private

    return 0;
}
```

Sample Run

```
The Height is : 5
The Width is : 2
```

2. Member Functions in Classes

There are 2 ways to define a member function:

1. Inside Class Definition
2. Outside Class Definition

2.1 Inside class definition

With an inline function, the compiler tries to expand the code in the body of the function in place of a call to the function.

Note that all the member functions defined inside the class definition are by default inline, but you can also make any non-class function inline by using keyword **`inline`** with them. Inline functions are actual functions, which are copied everywhere during compilation, like preprocessor macro, so the overhead of function calling is reduced.

2.2 Outside Class Definition

To define a member function outside the class definition we have to use the scope resolution **`::`** operator along with class name and function name.

```
#include <iostream>
using namespace std;

class car{
    private:
        int car_number;
        char car_model[10];
    public:
        void getdata(); //function declaration
        void showdata();
};

//Outside class function definition

void car::getdata(){

    cout<< "Enter car number: ";
    cin>>car_number;
    cout<< "Enter car model: ";
    cin>>car_model;
}

//Outside class function definition
```

```
void car::showdata(){

    cout<<"Car number is "<<car_number << endl;
    cout<<"Car model is "<<car_model;
}

// main function starts
int main(){

    car c1;
    c1.getdata();
    c1.showdata();

    return 0;
}
```

Sample Run	Enter car number: 9999 Enter car model: Sedan Car number is 9999 Car model is Sedan

3. Structure vs Classes

By default, all structure fields are **public**, or available to functions (like the **main()** function) that are outside the structure. Conversely, all class fields are **private**. That means they are not available for use outside the class. When you create a **class**, you can declare some fields to be **private** and some to be **public**. For example, in the real world, you might want your **name** to be **public** knowledge but your **Social Security Number**, **salary**, or **age** to be **private**.

4. Transformation from Procedural to Object-Oriented Programming

4.1 Procedural Approach

```
#include <iostream>
using namespace std;

double calculateBMI (double w, double h){

    return w/(h*h)*703;

}
string findStatus (double bmi){

    string status;
    if (bmi < 18.5)
    {
        status = "underweight";
    }
    else if (bmi < 25.0)
    {
        status = "normal"; // so on
    }
```

```
    return status;
}

int main(){

    double bmi, weight, height;
    string status;

    cout << "Enter weight in pounds: ";
    cin >> weight;
    cout << "Enter height in inches: ";
    cin >> height;

    bmi = calculateBMI(weight, height);

    cout << "Your BMI is " << bmi << endl;
    cout << "Your Status is " << findStatus(bmi);

    return 0;
}
```

Sample Run	Enter weight in pounds: 143 Enter height in inches: 69 Your BMI is 21.1151 Your Status is normal

4.2 Object-Oriented Approach

```
#include <iostream>
using namespace std;

class BMI
{
    double weight, height, bmi;
    string status;

public:
    void getInput()
    {
        // input weight in kilograms with validation
        cout << "Enter weight in kilograms: ";
        cin >> weight;
        if (weight <= 0)
        {
            cout << "Invalid weight. Please enter a positive value." <<
endl;
            return;
        }

        // input height in inches with validation
        cout << "Enter height in inches (e.g., 68 for 5 feet 8 inches): ";
        cin >> height;
        if (height <= 0)
        {
```

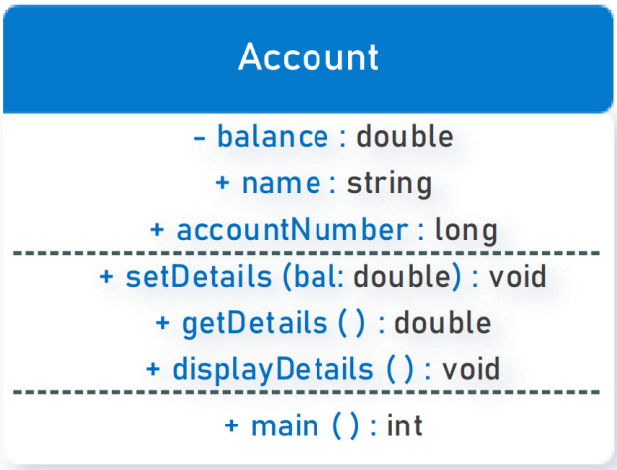
```
            cout << "Invalid height. Please enter a positive value." <<
endl;
            return;
        }
    }

    double calculateBMI()
    {
        // convert height from inches to meters
        double heightInMeters = height * 0.0254;

        // calculate BMI if both weight and height are valid
        if (weight > 0 && height > 0)
        {
            bmi = weight / (heightInMeters * heightInMeters);
            return bmi;
        }
        return -1; // Return -1 if inputs are invalid
    }

    string findStatus()
    {
        if (bmi <= 0)
        {
            status = "Invalid BMI calculation. Check your inputs.";
        }
        else if (bmi < 18.5)
        {
            status = "Underweight";
        }
        else if (bmi >= 18.5 && bmi < 25.0)
        {
            status = "Normal weight";
        }
        else if (bmi >= 25.0 && bmi < 30.0)
        {
            status = "Overweight";
        }
        else
        {
            status = "Obese";
        }
        return status;
    }

    void printStatus()
    {
        bmi = calculateBMI();

        if (bmi > 0)
        {
            cout << "Your BMI is " << bmi << endl;
```

```
            cout << "Your Status is " << findStatus() << endl;
        }
        else
        {
            cout << "Unable to calculate BMI due to invalid input." <<
endl;
        }
    }
};

int main()
{
    BMI bmi;
    bmi.getInput();
    bmi.printStatus();

    return 0;
}
```

Sample Run

```
Enter weight in kilograms: 60
Enter height in inches (e.g., 68 for 5 feet 8 inches): 68
Your BMI is 20.1125
Your Status is Normal weight
```

4.3 Object-Oriented Approach (another example with UML Diagram)




```
#include <iostream>
using namespace std;

class Account{
    private:

        double balance; // Account Balance

    public:

        string name; // Account Holder
        long accountNumber; // Account Number
```

```
    void setDetails(double bal){

        balance = bal;
    }

    double getDetails(){

        return balance;
    }

    void displayDetails(){

        cout << "Details are: " << endl;
        cout << "Account Holder: " << name << endl;
        cout << "Account Number: " << accountNumber << endl;
        cout << "Account Balance: " << getDetails();
    }
};

int main(){

    double accBal;

    Account currentAccount;
    currentAccount.getDetails();

    cout << "Please enter the details" << endl;

    cout << "Enter Name: ";
    getline(cin, currentAccount.name);

    cout << "Enter Account Number: ";
    cin >> currentAccount.accountNumber;

    cout << "Enter Account Balance: ";
    cin >> accBal;

    currentAccount.setDetails(accBal);

    currentAccount.displayDetails();

    return 0;
}
```

Set and Get Functions to manipulate Private Data Members

Publicly Available Data
Assigning values from.

Private Data
Assigning private data using member function.

Sample Run

```
Please enter the details
Enter Name: ABC
Enter Account Number: 102225111
Enter Account Balance: 50000
Details are:
Account Holder: ABC
Account Number: 102225111
Account Balance: 50000
```