Contents

Maths teacher
Talk()
Walk()
TeachMaths()
Footballer
Talk()
Walk()
PlayFootball()
Businessman
Talk()
Walk()
RunBusiness()

C++ Object and Class

C++ is a multi-paradigm programming language. Meaning, it supports different programming styles.

One of the popular ways to solve a programming problem is by creating objects, known as object-oriented style of programming.

C++ supports object-oriented (OO) style of programming which allows you to divide complex problems into smaller sets by creating objects.

Object is simply a collection of data and functions that act on those data.

C++ Class

Before you create an object in C++, you need to define a class.

A class is a blueprint for the object.

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.

How to define a class in C++?

A class is defined in C++ using keyword `class` followed by the name of class.

The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

```
class className

   {
```

```
   // some data

   // some functions

   };
```

Example: Class in C++

```
class Test
{
    private:
        int data1;
        float data2;

    public:
        void function1()
        {   data1 = 2;  }

        float function2()
        {
            data2 = 3.5;
            return data2;
        }
   };
```

Here, we defined a class named `Test`.

This class has two data members: data1 and data2 and two member functions:

Keywords: private and public

You may have noticed two keywords: private and public in the above example.

The private keyword makes data and functions private. Private data and functions can be accessed only from inside the same class.

The public keyword makes data and functions public. Public data and functions can be accessed out of the class.

Here, `data1` and `data2` are private members where as `function1()` and `function2() are public members`.

If you try to access private data from outside of the class, compiler throws error. This feature in OOP is known as data hiding.

`function1()` and `function2()`.

C++ Objects

When class is defined, only the specification for the object is defined; no memory or storage is allocated.

To use the data and access functions defined in the class, you need to create objects.

Syntax to Define Object in C++

```
className objectVariableName;
```

You can create objects of `Test` class (defined in above example) as follows:

```
class Test
{
    private:
        int data1;
        float data2;

    public:
        void function1()
        {   data1 = 2;  }

        float function2()
        {
            data2 = 3.5;
            return data2;
        }
   };

int main()
{
    Test o1, o2;
}
```

Here, two objects `o1` and `o2` of `Test` class are created.

In the above class `Test`, `data1` and `data2` are data members and `function1()` and`function2()` are member functions.

How to access data member and member function in C++?

You can access the data members and member functions by using a . (dot) operator. For example,

```
o2.function1();
```

This will call the function1() function inside the Test class for objects o2.

Similarly, the data member can be accessed as:

```
o1.data2 = 5.5;
```

It is important to note that, the private members can be accessed only from inside the class.

So, you can use `o2.function1();` from any function or class in the above example. However, the code `o1.data2 = 5.5;` should always be inside the class `Test`.

C++ Constructors

Constructors are special type of member functions that initializes an object automatically when it is created.

Compiler identifies a given member function is a constructor or not by its name and the return type.

Constructor has the same name as that of the class and it does not have any return type. Also, the constructor is always be public.

```
... .. ...

class temporary
```

```
{

private:

        int x;

        float y;

public:

        // Constructor

        temporary(): x(5), y(5.5)

        {

                // Body of constructor

        }

        ... ..  ...

};



int main()

{

        Temporary t1;

        ... .. ...

}
```

Above program shows a constructor is defined without a return type and the same name as the class.

How constructor works?

In the above pseudo code, `temporary()` is a constructor.

When an object of class `temporary` is created, the constructor is called automatically, and `x` is initialized to 5 and `y` is initialized to 5.5.

You can also initialize the data members inside the constructor's body as below. However, this method is not preferred.

```
temporary()

{

   x = 5;

   y = 5.5;

}

// This method is not preferred.
```

Use of Constructor in C++

Suppose you are working on 100's of `Person` objects and the default value of a data member `age` is 0. Initializing all objects manually will be a very tedious task.

Instead, you can define a constructor that initializes `age` to 0. Then, all you have to do is create a `Person` object and the constructor will automatically initialize the `age`.

These situations arise frequently while handling array of objects.

Also, if you want to execute some code immediately after an object is created, you can place the code inside the body of the constructor.

Access	**Public**	**protected**	**private**
Same class	Yes	Yes	yes
Derived classes	Yes	Yes	no
Outside classes	Yes	No	No

Inheritance :

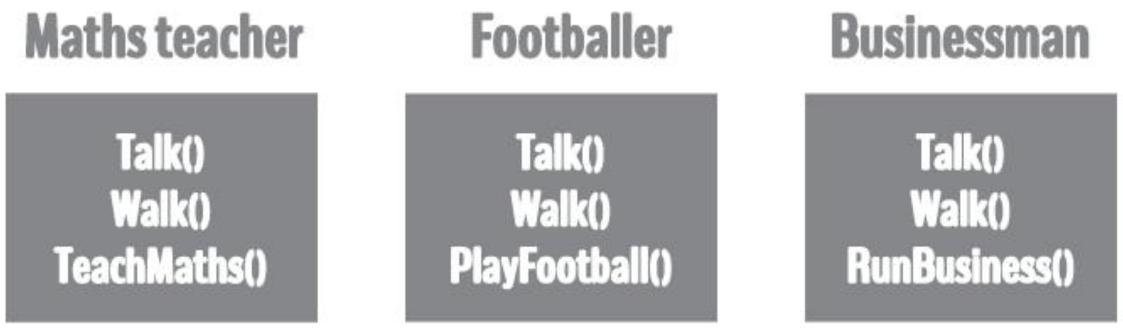
Inheritance is one of the key features of Object-oriented programming in C++. It allows user to create a new **class** (derived class) from an existing class (base class).
The derived class inherits all the features from the base class and can have additional features of its own.

Why inheritance should be used?

Suppose, in your game, you want three characters - a **math's teacher**, a **footballer** and a **businessman**.

Since, all of the characters are persons, they can walk and talk. However, they also have some special skills. A math's teacher can **teach maths**, a footballer can **play football** and a businessman can **run a business**.

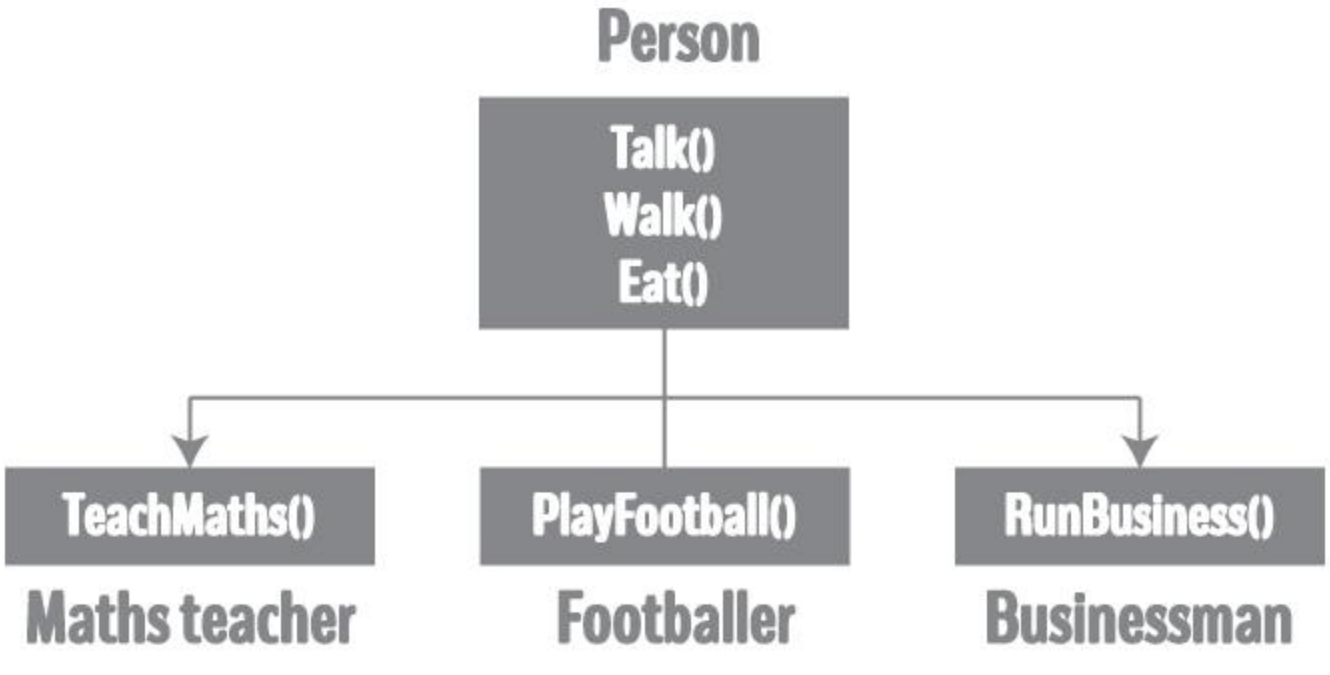
You can individually create three classes who can walk, talk and perform their special skill as shown in the figure below.



In each of the classes, you would be copying the same code for walk and talk for each character.

If you want to add a new feature - eat, you need to implement the same code for each character. This can easily become error prone (when copying) and duplicate codes.

It'd be a lot easier if we had a **Person** class with basic features like talk, walk, eat, sleep, and add special skills to those features as per our characters. This is done using inheritance.



Using inheritance, now you don't implement the same code for walk and talk for each class. You just need to **inherit** them.

So, for Maths teacher (derived class), you inherit all features of a Person (base class) and add a new feature **TeachMaths**. Likewise, for a footballer, you inherit all the features of a Person and add a new feature **PlayFootball** and so on.

This makes your code cleaner, understandable and extendable.

It is important to remember: When working with inheritance, each derived class should satisfy the condition whether it **"is a"** base class or not. In the example above, Maths teacher **is a** Person, Footballer **is a** Person. You cannot have: Businessman **is a** Business.

Implementation of Inheritance in C++ Programming

```
class Person
{
 ... .. ...
};

class MathsTeacher : public Person
{
 ... .. ...
};

class Footballer : public Person
{
 .... .. ...
};
```

In the above example, class `Person` is a base class and classes `MathsTeacher` and `Footballer` are the derived from *Person*.

The derived class appears with the declaration of a class followed by a colon, the keyword `public` and the name of base class from which it is derived.

Since, `MathsTeacher` and `Footballer` are derived from `Person`, all data member and member function of `Person` can be accessible from them.

Multilevel Inheritance

In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance.

```
class A
{
... .. ...
};
class B: public A
{
... .. ...
};
class C: public B
```

{
… … …
};
Here, class B is derived from the base class A and the class C is derived from the derived class B.

C++ Multiple Inheritance

In C++ programming, a class can be derived from more than one parents. For example: A class Bat is derived from base classes Mammal and WingedAnimal. It makes sense because bat is a mammal as well as a winged animal.

Types of Inheritance:

1. Single Inheritance
2. Multi-level Inheritance
3. Multiple Inheritance
4. Hierarchal Inheritance
5. Hybrid Inheritance

POLYMORPHISM:

The word **polymorphism** means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Virtual Function:

A **virtual** function is a function in a base class that is declared using the keyword **virtual**. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function.

What we do want is the selection of the function to be called at any given point in the program to be based on the kind of object for which it is called. This sort of operation is referred to as **dynamic linkage**, or **late binding**.

Pure Virtual Functions:

It's possible that you'd want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class.

We can change the virtual function area() in the base class to the following:

```
class Shape {
  protected:
    int width, height;
  public:
    Shape( int a = 0, int b = 0) {
      width = a;
      height = b;
    }

    // pure virtual function
    virtual int area() = 0;
};
```

The = 0 tells the compiler that the function has no body and above virtual function will be called **pure virtual function**.

Data abstraction:

Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.
Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.

Let's take one real life example of a TV, which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players, BUT you do not know its internal details, that is, you do not know how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.
Thus, we can say a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having zero knowledge of its internals.

Now, if we talk in terms of C++ Programming, C++ classes provides great level of **data abstraction**. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally.

For example, your program can make a call to the **sort()** function without knowing what algorithm the function actually uses to sort the given values. In fact, the underlying implementation of the sorting functionality could change between releases of the library, and as long as the interface stays the same, your function call will still work.

In C++, we use **classes** to define our own abstract data types (ADT). You can use the **cout** object of class **ostream** to stream data to standard output like this:

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


int main( ) {
  cout << "Hello C++" <<endl;
  return 0;
}
```

Here, you don't need to understand how **cout** displays the text on the user's screen. You need to only know the public interface and the underlying implementation of cout is free to change.

Data encapsulation:

All C++ programs are composed of the following two fundamental elements:
Program statements (code): This is the part of a program that performs actions and they are called functions.
Program data: The data is the information of the program which affected by the program functions.
Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of **data hiding**.
Data encapsulation is a mechanism of bundling the data, and the functions that use them and **data abstraction** is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called **classes**. We already have studied that a class can contain **private, protected** and **public** members. By default, all items defined in a class are private.

For example:

```
class Box {
   public:
      double getVolume(void) {
         return length * breadth * height;
      }

   private:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box
};
```

The variables length, breadth, and height are **private**. This means that they can be accessed only by other members of the Box class, and not by any other part of your program. This is one way encapsulation is achieved.

To make parts of a class **public** (i.e., accessible to other parts of your program), you must declare them after the **public** keyword. All variables or functions defined after the public specifier are accessible by all other functions in your program.

Making one class a friend of another exposes the implementation details and reduces encapsulation. The ideal is to keep as many of the details of each class hidden from all other classes as possible.