C++ Classes/Objects

C++ is an object-oriented programming language.

Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

Attributes and methods are basically **variables** and **functions** that belongs to the class. These are often referred to as "class members".

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the `class` keyword:

Example

Create a class called "`MyClass`":

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};
```

Example explained

- The `class` keyword is used to create a class called `MyClass`.
- The `public` keyword is an **access specifier**, which specifies that members (attributes and methods) of the class are accessible from outside the class. You will learn more about access specifiers later.
- Inside the class, there is an integer variable `myNum` and a string variable `myString`. When variables are declared within a class, they are called **attributes**.
- At last, end the class definition with a semicolon `;`.

Create an Object

In C++, an object is created from a class. We have already created the class named `MyClass`, so now we can use this to create objects.

To create an object of `MyClass`, specify the class name, followed by the object name.

To access the class attributes (`myNum` and `myString`), use the dot syntax (`.`) on the object:

Example

Create an object called "`myObj`" and access the attributes:

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Multiple Objects

You can create multiple objects of one class:

Example

```
// Create a Car class with some attributes
class Car {
  public:
    string brand;
    string model;
    int year;
};

int main() {
  // Create an object of Car
  Car carObj1;
  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;

  // Create another object of Car
  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;

  // Print attribute values
  cout << carObj1.brand << " " << carObj1.model << " " <<
carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " <<
carObj2.year << "\n";
  return 0;
}
```

Class Methods

Methods are **functions** that belongs to the class.

There are two ways to define functions that belongs to a class:

- Inside class definition
- Outside class definition

In the following example, we define a function inside the class, and we name it "`myMethod`".

Note: You access methods just like you access attributes; by creating an object of the class and using the dot syntax (`.`):

Inside Example

```
class MyClass {         // The class
  public:               // Access specifier
    void myMethod() {  // Method/function defined inside the class
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

To define a function outside the class definition, you have to declare it inside the class and then define it outside of the class. This is done by specifiying the name of the class, followed the scope resolution `::` operator, followed by the name of the function:

Outside Example

```
class MyClass {         // The class
  public:               // Access specifier
    void myMethod();   // Method/function declaration
};

// Method/function definition outside the class
void MyClass::myMethod() {
  cout << "Hello World!";
}

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

Constructors

A constructor in C++ is a **special method** that is automatically called when an object of a class is created.

To create a constructor, use the same name as the class, followed by parentheses `()`:

Example

```
class MyClass {     // The class
  public:           // Access specifier
    MyClass() {     // Constructor
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;    // Create an object of MyClass (this will call the
constructor)
  return 0;
}
```

Note: *The constructor has the same name as the class, it is always* `public`*, and it does not have any return value.*

Constructor Parameters

Constructors can also take parameters (just like regular functions), which can be useful for setting initial values for attributes.

The following class have `brand`, `model` and `year` attributes, and a constructor with different parameters. Inside the constructor we set the attributes equal to the constructor parameters (`brand=x`, etc). When we call the constructor (by creating an object of the class), we pass parameters to the constructor, which will set the value of the corresponding attributes to the same:

Example

```
class Car {         // The class
  public:           // Access specifier
    string brand;   // Attribute
    string model;   // Attribute
    int year;       // Attribute
    Car(string x, string y, int z) { // Constructor with parameters
      brand = x;
      model = y;
      year = z;
    }
};

int main() {
  // Create Car objects and call the constructor with different values
  Car carObj1("BMW", "X5", 1999);
  Car carObj2("Ford", "Mustang", 1969);

  // Print values
  cout << carObj1.brand << " " << carObj1.model << " " <<
carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " <<
carObj2.year << "\n";
  return 0;
}
```

Just like functions, constructors can also be defined outside the class. First, declare the constructor inside the class, and then define it outside of the class by specifying the name of the class, followed by the scope resolution `::` operator, followed by the name of the constructor (which is the same as the class):

Example

```
class Car {         // The class
  public:           // Access specifier
    string brand;   // Attribute
    string model;   // Attribute
    int year;       // Attribute
    Car(string x, string y, int z); // Constructor declaration
};

// Constructor definition outside the class
```

```
Car::Car(string x, string y, int z) {
  brand = x;
  model = y;
  year = z;
}

int main() {
  // Create Car objects and call the constructor with different values
  Car carObj1("BMW", "X5", 1999);
  Car carObj2("Ford", "Mustang", 1969);

  // Print values
  cout << carObj1.brand << " " << carObj1.model << " " <<
carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " <<
carObj2.year << "\n";
  return 0;
}
```

C++ Destructor

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

C++ Constructor and Destructor Example

Let's see an example of constructor and destructor in C++ which is called automatically.

```
#include <iostream>

using namespace std;

class Employee

 {

   public:

        Employee()
        {
            cout<<"Constructor Invoked"<<endl;
        }
        ~Employee()
        {
            cout<<"Destructor Invoked"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    return 0;
}
```

What is a Friend Function in C++?

A friend function in C++ is defined as a function that can access private, protected and public members of a class.

The friend function is declared using the friend keyword inside the body of the class.

Friend Function Syntax:

```
class className {
    ... .. ...
    friend returnType functionName(arguments);
    ... .. ...
}
```

By using the keyword, the **'friend'** compiler understands that the given function is a friend function.

We declare friend function inside the body of a class, whose private and protective data needs to be accessed, starting with the keyword friend to access the data. We use them when we need to operate between two different classes at the same time.

A simple example of a C++ friend function used to print the length of the box.

```
#include <iostream>
using namespace std;
class Box
{
   private:
        int length;
   public:
         Box (): length (0) {}
   friend int printLength (Box); //friend function
};
int printLength (Box b)
{
    b. length +=10;
    return b. length;
}
int main ()
{
   Box b;
   cout <<" Length of box:" <<printLength (b)<<endl;
    return 0;
}
```

Output:

```
Length of box:10
```