C++ Class

A class is a blueprint for the object.

We can think of a 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.

Create a Class

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

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

```
class className {
  // some data
  // some functions
};
```

For example,

```
class Room {
   public:
      double length;
      double breadth;
      double height;

      double calculateArea(){
         return length * breadth;
      }

      double calculateVolume(){
         return length * breadth * height;
      }

};
```

Here, we defined a class named `Room`.

The variables `length`, `breadth`, and `height` declared inside the class are known as **data members**. And, the functions `calculateArea()` and `calculateVolume()` are known as **member functions** of a class.

C++ Objects

When a 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, we need to create objects.

Syntax to Define Object in C++

```
className objectVariableName;
```

We can create objects of `Room` class (defined in the above example) as follows:

```
// sample function
void sampleFunction() {
  // create objects
  Room room1, room2;
}

int main(){
  // create objects
  Room room3, room4;
}
```

Here, two objects `room1` and `room2` of the `Room` class are created in `sampleFunction()`. Similarly, the objects `room3` and `room4` are created in `main()`.

As we can see, we can create objects of a class in any function of the program. We can also create objects of a class within the class itself, or in other classes.

Also, we can create as many objects as we want from a single class.

C++ Constructor:

A constructor is a special type of member function that is called automatically when an object is created.

In C++, a constructor has the same name as that of the class and it does not have a return type. For example,

```
class Wall {
  public:
    // create a constructor
    Wall() {
      // code
    }
};
```

Here, the function `Wall()` is a constructor of the class `Wall`. Notice that the constructor

- has the same name as the class,
- does not have a return type, and
- is `public`

C++ Encapsulation

Encapsulation

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must declare class variables/attributes as private (cannot be accessed from outside the class). If you want others to read or modify the value of a private member, you can provide public **get** and **set** methods.

Access Private Members

To access a private attribute, use public "get" and "set" methods:

Example

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

class Employee {
  private:
    // Private attribute
    int salary;

  public:
    // Setter
    void setSalary(int s) {
      salary = s;
    }
    // Getter
    int getSalary() {
      return salary;
    }
};

int main() {
  Employee myObj;
  myObj.setSalary(50000);
  cout << myObj.getSalary();
  return 0;
}
```

Example explained

The salary attribute is private, which have restricted access.

The public setSalary() method takes a parameter (s) and assigns it to the salary attribute (salary = s).

The public getSalary() method returns the value of the private salary attribute.

Inside main(), we create an object of the Employee class. Now we can use the setSalary() method to set the value of the private attribute to 50000. Then we call the getSalary() method on the object to return the value.

Why Encapsulation?

- It is considered good practice to declare your class attributes as private (as often as you can). Encapsulation ensures better control of your data, because you (or others) can change one part of the code without affecting other parts
- Increased security of data

C++ Polymorphism

Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; **Inheritance** lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a base class called Animal that has a method called animalSound(). Derived classes of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

Example

```
// Base class
class Animal {
  public:
    void animalSound() {
    cout << "The animal makes a sound \n" ;
  }
};

// Derived class
class Pig : public Animal {
  public:
    void animalSound() {
    cout << "The pig says: wee wee \n" ;
  }
};

// Derived class
class Dog : public Animal {
  public:
    void animalSound() {
    cout << "The dog says: bow wow \n" ;
  }
};
```

C++ Inheritance

In this tutorial, we will learn about inheritance in C++ with the help of examples.

Inheritance is one of the key features of Object-oriented programming in C++. It allows us to create a new class (derived class) from an existing class (base class).

The derived class inherits the features from the base class and can have additional features of its own. For example,

```
class Animal {
  // eat() function
  // sleep() function
};

class Dog : public Animal {
  // bark() function
};
```

Here, the `Dog` class is derived from the `Animal` class. Since `Dog` is derived from `Animal`, members of `Animal` are accessible to `Dog`.

public, protected and private inheritance in C++

public, **protected,** and **private** inheritance have the following features:

- **public inheritance** makes `public` members of the base class `public` in the derived class, and the `protected` members of the base class remain `protected` in the derived class.
- **protected inheritance** makes the `public` and `protected` members of the base class `protected` in the derived class.
- **private inheritance** makes the `public` and `protected` members of the base class `private` in the derived class.

C++ Multilevel Inheritance

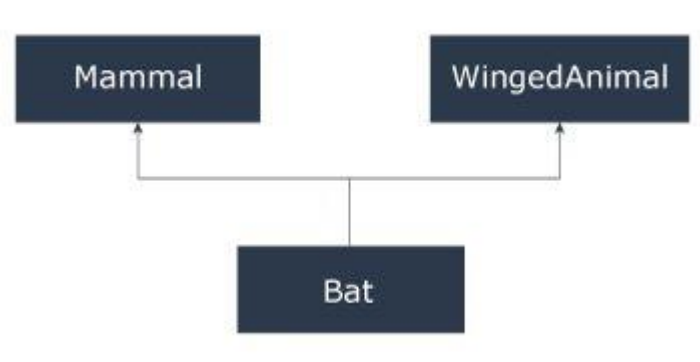
In C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance.

```
class A {

... .. ...
```

```
};

class B: public A {

... .. ...

};

class C: public B {

... ... ...

};
```

C++ Multiple Inheritance

In C++ programming, a class can be derived from more than one parent. 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.



C++ Hierarchical Inheritance

If more than one class is inherited from the base class, it's known as hierarchical inheritance. In hierarchical inheritance, all features that are common in child classes are included in the base class.

For example, Physics, Chemistry, Biology are derived from Science class. Similarly, Dog, Cat, Horse are derived from Animal class.

```
class base_class {

    ... .. ...

}
```

```
class first_derived_class: public base_class {

    ... .. ...

}

class second_derived_class: public base_class {

    ... .. ...

}

class third_derived_class: public base_class {

    ... .. ...}
```

C++ Exceptions

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an **exception** (throw an error).

C++ try and catch

Exception handling in C++ consist of three keywords: try, throw and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

Example

```
try {
  // Block of code to try
  throw exception; // Throw an exception when a problem arise
}
catch () {
  // Block of code to handle errors
}
```

Handle Any Type of Exceptions (...)

If you do not know the throw **type** used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception:

Example

```
try {
  int age = 15;
  if (age >= 18) {
    cout << "Access granted - you are old enough.";
  } else {
    throw 505;
  }
}
catch (...) {
  cout << "Access denied - You must be at least 18 years old.\n";
}
```