**C++ Class Methods**

## **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;  
}

### **Inside Example**

#include <iostream>

using namespace std;

class Car {

public:

string model;

void details(){

cout<<"The car model is:"<<model<<endl;

}

};

int main() {

Car myObj; // Create an object of Car

myObj.model="Toyotal Car";

myObj.details();

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;  
}

### **Outside Example**

#include <iostream>

using namespace std;

class Car {

public:

string model;

void details();

};

void Car::details(){

cout<<"The car model is:"<<model<<endl;

}

int main() {

Car myObj; // Create an object of Car

myObj.model="Toyotal Car";

myObj.details();

return 0;

}

### **Example**

#include <iostream>

using namespace std;

class Student

{

public:

string name;

int id;

void myName(); //Define function outside the class definition

void myNumber(){

cout <<"The number is " <<id<<endl;

}

};

void Student:: myName() // Definition of myName using ::

{

cout <<"My name is " <<name<<endl;

}

int main(){

Student nam1;

nam1.name="Nick";

nam1.id=20180022;

nam1.myName();

nam1.myNumber();

return 0;

}

**Output**

**Graphical user interface, text

Description automatically generated**

## **Parameters**

You can also add parameters:

### **Example**

#include <iostream>

using namespace std;

class Car {

public:

void model(string carmodel);

};

void Car::model(string carmodel) {

cout<<"The model of the car is "<< carmodel<<endl;

}

int main() {

Car myObj; // Create an object of Car

myObj.model("Toyota"); // Call the method with an argument

return 0;

}

### **Example**

#include <iostream>  
using namespace std;  
  
class Car {  
  public:  
    int speed(int maxSpeed);  
};  
  
int Car::speed(int maxSpeed) {  
  return maxSpeed;  
}  
  
int main() {  
  Car myObj; // Create an object of Car  
  cout << myObj.speed(200); // Call the method with an argument  
  return 0;  
}

// C++ program to demonstrate function

// declaration outside class

### **Example**

#include <iostream>

using namespace std;

class Book {

public:

void details(string title, int index);

};

void Book::details(string title, int index) {

cout<<"The title of the book is"<< title << " and ID is "<<index <<endl;

}

int main() {

Book mybook; // Create an object of Car

mybook.details("Programming Principles" , 123546); // Call the method with an argument

return 0;

}

// C++ program to demonstrate function

// declaration outside class

### **Example**

#include <iostream>

using namespace std;

class Name{

public: //Access specifier

string myname; //Data Member

int id1; //Data Member

int id2; //Data Member

// printname is not defined inside class definition

void printname(); // Member function

void printid1() // Member function

{

cout<<"My id1 is:" << id1 <<endl;

}

void printid2() // Member function

{

cout<<"My id2 is:" << id2 <<endl;

}

};

// Definition of printname using scope resolution operator ::

void Name :: printname()

{

cout<<"My name is:" << myname <<endl;

}

int main()

{

Name nam1; //Declaring an object

nam1.myname = "Emmanuel"; //Accessing data member

nam1.id1 = 200; //Accessing data member

nam1.id2 = 250; //Accessing data member

nam1.printname(); //Accessing member function

nam1.printid1(); //Accessing member function

nam1.printid2(); //Accessing member function

return 0;

}

**Output**

Graphical user interface, text

Description automatically generated

## **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**

#include <iostream>

using namespace std;

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.

[**Default constructors**](http://quiz.geeksforgeeks.org/constructors-c/)

**1. Default Constructor**

A default constructor, also called no-argument constructor, can initialize data members to constant values.

Syntax:

class\_name()

{

//body

}

### **Example**

#include <iostream>

using namespace std;

class BookId

{

public:

int id;

BookId() //Default Constructor

{

id=200;

}

};

int main(){

BookId dee; // dee will call Default Constructor

cout <<"The id number is "<<dee.id;

return 0;

}

**Output**



### **Example**

#include <iostream>

using namespace std;

class BookId

{

public:

int id1;

int id2;

BookId()//Default Constructor

{

id1=200;

id2=300;

}

};

int main(){

BookId dee; // dee will call Default Constructor

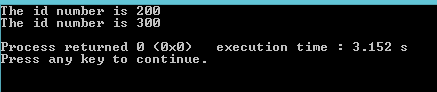
cout <<"The id number is "<<dee.id1<<endl;

cout <<"The id number is "<<dee.id2<<endl;

return 0;

}

**Output**



### **Example**

#include <iostream>

using namespace std;

class BookId

{

public:

int id;

string name;

BookId() //Default Constructor

{

id=200;

name="Programming Principles";

}

};

int main(){

BookId dee; // dee will call Default Constructor

cout <<"The id number is " << dee.id << "and the name is" << dee.name;

return 0;

}

**Output**

The id number is 200 and the name is Programming Principles

--------------------------------

Process exited after 0.1923 seconds with return value 0

Press any key to continue . . .

## **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;  
}

### **Example**

#include <iostream>

using namespace std;

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;

cout<<brand<<" "<<model<<" "<<year<<"\n";

}

};

int main() {

// Create Car objects and call the constructor with different values

Car carObj1("BMW", "X5", 1999);

Car carObj2("Ford", "Mustang", 1969);

return 0;

}