OOP in C++

C++ Access Specifiers

In C++, there are three access specifiers:

- `public` - members are accessible from outside the class
- `private` - members cannot be accessed (or viewed) from outside the class
- `protected` - members cannot be accessed from outside the class, however, they can be accessed in inherited classes. You will learn more about Inheritance later.

In the following example, we demonstrate the differences between `public` and `private` members:

Example

```
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
```

```
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

```
error: y is private
```

Note: By default, all members of a class are `private` if you don't specify an access specifier:

Example

```
class MyClass {
  int x;   // Private attribute
  int y;   // Private attribute
};
```

C++ Constructors

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;
}
```

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:

OOP in C++

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;
}
```

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

Static and non-static function (assignment)