Programming Fundamentals

Classes

Class

- 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" (plan) for creating objects
- A way to map real world objects into programming constructs
- C++ classes make code modular
- A C++ class is composed of methods and variables where
  - Attributes are mapped with variables
  - Behaviors are mapped with methods

Creating a class in C++

- To create a class, use the keyword class

Example:

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

Explanation

- 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.
- 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 1

```
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;
}
```

Example 2

```
// 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;
}
```

Classes and 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

Inside class 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;
}
```

Explanation

- In the following example, we define a function inside the class, and we name it myMethod
- Methods can be accessed by creating an object of the class and by using the dot syntax (.)

Outside class example

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

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

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

Explanation

- 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 specifying the name of the class, followed the scope resolution operator :: , followed by the name of the function.

Method with parameter

```
#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;
}
```

Constructor

- 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 ()
- The constructor has same name as of class
- The constructor has no return type
- It is always public

Example - Constructor

```
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;
}
```

Private access specifier

- Most restricted access specifier
- Members cannot be accessed (or viewed) from outside the class
- Accessible through public methods
- Default access specifier of class

Private access specifier - Example

```
class Student
{           private:
                        string name;
                        int   rollNo;
                        int   total;
                        float perc;
            public:
                        void getDetails();
                        void putDetails();
};
int main()
{           Student std;

            std.getDetails();
            std.putDetails();
            return 0;
}
```

```
//member function definition, outside of the class
void Student::getDetails(){
            cout << "Enter name: " ;
            cin >> name;
            cout << "Enter roll number: ";
            cin >> rollNo;
            cout << "Enter total marks outof 500: ";
            cin >> total;
            perc= (float) total/500*100;
}

//member function definition, outside of the class
void Student::putDetails(){
            cout << "Student details:\n";
            cout << "Name:"<< name << ",Roll Number:"
<< rollNo << ",Total:" << total << ",Percentage:" <<
perc;
}
```

Destructor

- Destructor is a special member function that is called when the lifetime of an object ends
- The purpose of the destructor is to free the resources that the object may have acquired during its lifetime
- It has same name as the class
- It is preceded by a tilde (~)
- Destructor does not take arguments, it can never be overloaded

Rules of destructor

- The name of the destructor must begin with the tilde character (~). You must not include any return type, not even void!
- A class can have no more than one destructor.
- The destructor cannot have any parameters.
- The class destructor is automatically invoked when the instance of that class goes out of scope.

References

- C++ How to Program
  By Deitel & Deitel

- The C++ Programming Language
  By Bjarne Stroustrup

- Object oriented programming using C++ by Tasleem Mustafa, Imran Saeed, Tariq Mehmood, Ahsan Raza

- https://www.tutorialspoint.com

- http://ecomputernotes.com

- http://www.cplusplus.com

- https://www.w3schools.com/cpp/