Object Oriented Programming in C++ 1

Seonghyun Park

dd November, 2018

Outline

1 Class
- Syntax
- Methods
- Constructors
- Destructors
- Access specifiers
- Inheritance

Today's Goal

- Learn how to declare methods and fields associated with an object or a class.
- Learn how initialize an object with constructor and how to clean up an object with destructor.
- Learn abount access specifiers (e.g. `public`, `private`, `protected`)
- Learn how inheritance and virtual functions works.

Object-Oritned Programming (OOP)

Object-oriented programming (OOP) is a programming paradigm based on the concept of **"objects"***, which may contain data, in the form of* **fields***, often known as attributes; and code, in the form of* **procedures***, often known as methods.*
- From Wikipedia

Class Syntax

```
// shape.h
class Circle {
private: // access specifier
  // member variables
  double c;
  Point p;
public: // access specifier
  // constructors
  Circle() = delete;
  Circle(double r) : c(Point{0, 0}), r(r) {}
  double area(); // method prototype
  double getRadius(); // getter
  void setRadius(); // setter
}; // Don't forget to put semicolon
```

Class Syntax

```
// shape.cpp
#include <cmath>

#include "shape.h"

double Circle::area()
{
  return std::pow(r, 2.0);
}

double Circle::getRadius() { return r; }
void Circle::setRadius(double r) { this->r = r; }
```

Methods

- Method is assoiated with either an object or a class i.e. *non-static* or *static*.
- All non-static method has access to member variables (both its member public and private member variables)
- All non-static method has access to special `this` pointer that points to the object itself.
- Note that you can declare `const` methods and those methods can be called with const references or const pointers.

```
class Circle {
  ...
  double area() const;
};
```

Constructors

- Constructors are used to define how to initialize an object.
- There are special kind of constructors copy constructor and move constructor.
- Copy constructor takes `const T&` and invoked with assignment. e.g. `C c2 = c1; C c3{c1};`

Constructors (Cont'd)

```
class Buffer {
private:
  int *buf;
  int cap;

public:
  Buffer() : buf(new int[1024]), cap(1024) {}
  Buffer(int N) : buf(new int[N]), cap(N) {}
  Buffer(const Buffer& b)
    : buf(new int[b.cap]), cap(b.cap)
  {
    for (int i = 0; i < N; i++)
      buf[i] = b.buf[i];
  }
};
```

Destructors

- Destructors are used to define how to clean up memory allocated for an object.

```
class Buffer {
private:
  int *buf;
  int cap;

public:
  ~Buffer() { delete[] buf; }
};
```

- For example, a class that represents a linked list should clean up the memory of all nodes.

Access specifiers

- If you are a library writer, you may want to hide or protect your implementation details from library users for many reasons (For better security, clean interfaces, modularity, and so on)
  1. `public`: A public member of a class is accessible everywhere.
  2. `protected`: A protected member of a class Base can only be accessed (1) by the members and friends of Base or (2) by the members and friends (until C++17) of any class derived from Base, but only when operating on an object of a type that is derived from Base (including this)
  3. `private`: A private member of a class can only be accessed by the members and friends of that class, regardless of whether the members are on the same or different instances

Inheritance

- In object-oriented programming, inheritance establishes an **is-a** relationship between a parent and a child.
- See the links below for further information.
  - `https://en.cppreference.com/book/intro/inheritance`
  - `https://en.cppreference.com/book/intro/derived_class`
  - `https://en.cppreference.com/book/intro/virtual`