COMP 322 Lecture 7 - Classes in C++ 2

Junji Duan

2024/2/16

Today's Outline

- Constructors
- Destructors

C++ classes: destructors

- When an object gets out of scope, a special method is implicitly called
  - This method is the destructor
- Every class has a destructor (and only one)
- If a destructor is not provided by the programmer, the compiler will provide a default implicit destructor (that usually calls the destructor for each data member but will not delete dynamically allocated memory for you)
- Destructor method:
  - is used to clean and liberate any resource that was being held by the object
  - must have the same name as the class preceded by the tilde ~ operator
  - must be declared public in general
    - There are some exceptions when implementing advanced design patterns
  - does not have a return type nor does it take parameters
    - Destructors don't return values

```
class Person
{
public:
    // constructors
    Person();
    Person(int age, char sex);
    Person(int age, char sex, char *name);
    // destructor
    ~Person();
    int  getAge();
    char* getName();
protected:
    int age;
    char sex;
    char *name;
};
```

```
Person::Person(int age, char sex, char *name)
{
    cout << "Constructor got called" << endl;
    this->age = age;
    this->sex = sex;
    this->name = new char[strlen(name)];
    strcpy(this->name, name);
}

Person::~Person()
{
    cout << "Destructor got called" << endl;
    delete [] this->name;
}

char* Person::getName()
{
    return this->name;
}
```

```
Person::Person(int age, char sex, char *name)
{
    cout << "Constructor got called" << endl;
    this->age = age;
    this->sex = sex;     allocate 7
    this->name = new char[strlen(name)];
    strcpy(this->name, name);
}

Person::~Person()
{
    cout << "Destructor got called" << endl;
    delete [] this->name;
}

char* Person::getName()
{
    return this->name;
}
```

```
// main function
int main()
{
    Person Mike(24, 'M', "Michael");

    cout << Mike.getName() << endl;
}
```

actual output

```
Constructor got called
Michael
Destructor got called
```

C++ classes: dynamic allocation

```
class GPS
{
public:
    GPS(double altitude, double longitude, double latitude):
        altitude(altitude),
        longitude(longitude),
        latitude(latitude)
    {
        cout << "GPS Constructor" << endl;
    }

    ~GPS()
    {
        cout << "GPS Destructor" << endl;
    }

private:
    double altitude;
    double longitude;
    double latitude;
};

int main()
{
    cout << "Program started ..." << endl;
    GPS* gps;
    cout << "A GPS pointer was being declared but not allocated yet" << endl;
    gps = new GPS(10, 25, 14);
    cout << "GPS Object was being allocated" << endl;
    delete gps;
    cout << "GPS Object was being deleted" << endl;
}
```

```
Program started ...
A GPS pointer was being declared but not allocated yet
GPS Constructor
GPS Object was being allocated
GPS Destructor
GPS Object was being deleted
```

Classes - design pattern example

```
class Singleton
{
public:
    static Singleton& getUniqueInstance()
    {
        static Singleton instance;
        return instance;
    }
    // Add the needed public methods
    void doSomething() {}

private:
    Singleton(){}
    ~Singleton(){}
    Singleton(Singleton &);
    Singleton operator= (Singleton &);
};

int main()
{
    Singleton& mySingleton = Singleton::getUniqueInstance();
    mySingleton.doSomething();
}
```

- The Singleton design pattern provides an example of a case where declaring a constructor private makes sense
- Singleton enforces that only one object of a class can be present during the lifetime of a program

In C++, constructor and destructor are two special member functions that are automatically called at different stages of an object's lifecycle, used for object initialization and cleanup.

Constructor

- Definition: Constructor is a special member function used for initializing objects when they are created. It has the same name as the class and no return type.

- Purpose: Used for resource allocation, initializing member variables, etc.
- Types: Can have default constructor (no parameters), parameterized constructor, copy constructor, etc.
- Characteristics: Can be overloaded but cannot be inherited.

```cpp
class Student {
public:
    // 默认构造函数
    Student() {
        name = "未知";
        age = 0;
    }

    // 参数化构造函数
    Student(string n, int a) {
        name = n;
        age = a;
    }

private:
    string name;
    int age;
};
```

In the example above, the Student class has two constructors: one is the default constructor, called when an object is created without providing any parameters; the other is the parameterized constructor, used to initialize the member variables name and age when creating an object.

Destructor

- Definition: Destructor is a special member function used for performing cleanup tasks before an object is destroyed. It is named by prefixing the class name with a tilde ( ), has no return type, and takes no parameters.
- Purpose: Used for releasing resources allocated during the object's lifetime, such as dynamically allocated memory, file handles, etc.
- Characteristics: A class can have only one destructor, and it cannot be overloaded or inherited.

```cpp
class Student {
public:
    // 构造函数
    Student() {
        name = new string;
        *name = "未知";
    }

    // 析构函数
    ~Student() {
        delete name; // 释放动态分配的内存
    }

private:
    string* name;
};
```

In this example, the constructor of the Student class dynamically allocates a string to store the student's name. Its destructor ensures that this memory is released when the object's lifetime ends, avoiding memory leaks.

In summary, constructor and destructor are essential concepts in C++, responsible for object initialization and cleanup tasks, ensuring effective resource management.