Object-Oriented Programming and Data Structures

COMP2012: Namespace

Brian Mak
Desmond Tsoi

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China

Motivation

Suppose that you want to use two libraries, each consisting of a bunch of useful classes and functions, but some of them have the same name.

```
/* File: apple-utils.h */
class Stack       { /* incomplete */ };
class Some_Class { /* incomplete */ };
void safari()     { cout << "Apple's browser" << endl; };
void app(int x)  { cout << "Apple's app: " << x << endl; };
```

```
/* File: ms-utils.h */
class Stack        { /* incomplete */ };
class Other_Class { /* incomplete */ };
void edge()        { cout << "Microsoft's browser" << endl; };
void app(int x)   { cout << "Microsoft's app: " << x << endl; };
```

Motivation ..

Even if you don't use Stack and app, you run into troubles:

- compiler complains about multiple definitions of Stack;
- compiler/linker complains about multiple definitions of app.

```
#include <iostream>      /* File: use-utils.cpp */
using namespace std;
#include "apple-utils.h"
#include "ms-utils.h"
enum class OS { MSWindows, MacOS } choice;

int main()
{
    Some_Class sc;
    Other_Class oc;

    if (choice == OS::MacOS)
        safari();
    else if (choice == OS::MSWindows)
        edge();
    return 0;
}
```

Solution: namespace

Solution: namespace ..

If the library writers would have used namespaces, multiple names wouldn't be a problem.

```
/* File: apple-utils-namespace.h */
namespace apple
{
    class Stack      { /* incomplete */ };
    class Some_Class { /* incomplete */ };
    void safari()    { cout << "Apple's browser" << endl; };
    void app(int x)  { cout << "Apple's app: " << x << endl; };
}
```

```
/* File: ms-utils-namespace.h */
namespace microsoft
{
    using namespace std;
    class Stack       { /* incomplete */ };
    class Other_Class { /* incomplete */ };
    void edge()       { cout << "Microsoft's browser" << endl; };
    void app(int x)   { cout << "Microsoft's app: " << x << endl; };
}
```

Namespace Alias & Scope Operator ::

Refer names in a namespace with the scope resolution operator.

```
#include <iostream>                          /* File: utils-namespace.cpp */
using namespace std;
#include "ms-utils-namespace.h"
#include "apple-utils-namespace.h"
namespace ms = microsoft;                    // Namespace alias
enum class OS { MSWindows, MacOS } choice;

int main()
{
    apple::Some_Class sc; apple::Stack apple_stack;
    ms::Other_Class oc; ms::Stack ms_stack;
    ms::app(42);

    cout << "Input your OS choice: ";
    int int_choice; cin >> int_choice; // Can't cin to choice. Why?
    switch (choice = static_cast<OS>(int_choice))
    {
        case OS::MSWindows: ms::edge(); break;
        case OS::MacOS: apple::safari(); break;
        default: cerr << "Unsupported OS" << endl;
    }
    return 0;
}
```

using Declaration

If you get tired of specifying the namespace every time you use a name, you can use a using declaration.

```
#include <iostream>        /* File: utils-using.cpp */
using namespace std;
#include "ms-utils-namespace.h"
#include "apple-utils-namespace.h"
namespace ms = microsoft; // Namespace alias
using apple::Some_Class;
using ms::Other_Class;
using apple::Stack;
using ms::app;

int main()
{
    Some_Class sc;          // Refer to apple::Some_Class
    Other_Class oc;         // Refer to ms::Other_Class
    Stack apple_stack;      // Refer to apple::Stack
    ms::Stack ms_stack;
    app(2); return 0;       // Refer to ms::app
}
```

Ambiguity With using Declarations

You can also bring all the names of a namespace into your program at once, but make sure it won't cause any ambiguities.

```
#include <iostream>                /* File: utils-using-err.cpp */
using namespace std;
#include "ms-utils-namespace.h"
#include "apple-utils-namespace.h"

namespace ms = microsoft;          // Namespace alias
using namespace apple;
using namespace ms;

int main()
{
    Some_Class sc;                 // Refer to apple::Some_Class
    Other_Class oc;                // Refer to ms::Other_Class
    Stack S;                       // Error: ambiguous;
    ms::Stack ms_stack;            // OK
    apple::Stack apple_stack;      // OK
    return 0;
}
```

Namespace std

```cpp
#include <iostream>      /* File: std-using.cpp */
using namespace std;

int main()
{
    string s;
    cin >> s;
    cout << s << endl;

    s += " is good!";
    cout << s << endl;

    return 0;
}
```

How Should We Declare Namespaces?

- Functions and classes of the standard library (`string`, `cout`, `isalpha()`, . . .) and the STL (`vector`, `list`, `foreach`, `swap`, . . .) are all defined in namespace std.
- Here, we bring all the names that are declared in the three header files into the global namespace.
- Although the previous program works, it is considered bad practice to declare the namespace std globally.
- It is better to introduce only the names you really need, or to qualify the names whenever you use them.
- Although this takes more typing effort, it is also immediately clear which functions and classes are from the standard (template) library, and which are your own.
- A combination of using declarations and explicit scope resolution is also possible; this is mostly a matter of taste.

Explicit Use of using Declaration Per Object/Function

```cpp
#include <iostream>      /* File: std-per-obj-using.cpp */
using std::string;
using std::cin;
using std::cout;
using std::endl;


int main()
{
    string s;
    cin >> s;
    cout << s << endl;

    s += " is good!";
    cout << s << endl;

    return 0;
}
```

Explicit Use of namespace Per Instance of Object/Function

```cpp
#include <iostream>     /* File: std-per-instance-using.cpp */
using namespace std;

int main()
{
    std::string s;
    std::cin >> s;
    std::cout << s << std::endl;

    s += " is good!";
    std::cout << s << std::endl;

    return 0;
}
```

Namespace Is Expansible

- Namespaces can be defined in steps and nested.

```cpp
#include <iostream>      /* File: misc-namespace.cpp */

namespace hkust
{
    namespace cse { int rank() { return 1; } } // Nested namespace
    void good() { std::cout << "Good!" << std::endl; }
}

namespace hkust   // Extend the namespace
{
    void school() { std::cout << "School!" << std::endl; }
}

int main()
{
    std::cout << "CSE's rank: " << hkust::cse::rank() << std::endl;
    hkust::good();
    hkust::school(); return 0;
}
```