سلسلة تعلم البرمجة بلغة C++ الحديثة

Learn Modern C++ Programming Course

إعداد المهندس أحمد الديب

#28: Namespaces p1

Composition Problems

```
// graphics library
class Shape { /* ... */ };
class Line : public Shape { /* ... */ };
class Text : public Shape { /* ... */ };


// text manipulation library
class Word { /* ... */ };
class Line { /* ... */ };
class Text { /* ... */ };
```

```
error : name clashes
workarounds: use different names, make the names
longer, add prefix, etc.
```

```
#include "Graph_lib.h"
#include "Text_lib.h"
// ...
```

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Namespaces

```
// graphics library
namespace Graph_lib {
  class Shape { /* ... */ };
  class Line : public Shape { /* ... */ };
  class Text : public Shape { /* ... */ }; // text label
}


// text manipulation library
namespace Text_lib {
  class Word { /* ... */ };
  class Line { /* ... */ };
  class Text { /* ... */ };
}
```

A namespace is a (named) scope. You can access members defined earlier in a namespace from later declarations, but you cannot (without special effort) refer to members from outside the namespace.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Explicit Qualification

```
namespace Calculator {
double add(double, double);
double sub(double, double);
double multiply(double, double);
}  // namespace Calculator


double Calculator::add(double a, double b)  // definition
{
  return a + b;
}
double val = Calculator::add(1, 2);  // use


double Calculator::divide(double, double);   // error : no divide()
double Calculator::multply(double, double);  // error : (misspelling)
double Calculator::add(int, int);            // error : (wrong type)
```

Explicit Qualification

- A namespace is a scope. The usual scope rules hold for namespaces.
- The global scope is a namespace and can be explicitly referred to using ::
- Classes are namespaces.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

using-Declarations

```
namespace Calculator {
double add(double, double);
double sub(double, double);
double multiply(double a, double b) { return a * b; }
}  // namespace Calculator
```

```
using Calculator::multiply;
double val2 = multiply(1, 2);
```

When a name is frequently used outside its namespace, it can be a bother to repeatedly qualify it with its namespace name.

When used for an overloaded name, a using-declaration applies to all the overloaded versions.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

using-Directives

```
// graphics library
namespace Graph_lib {
class Line {};
} // namespace Graph_lib

// text manipulation library
namespace Text_lib {
class Line {};
} // namespace Text_lib

using namespace std; // make every name from std accessible
using namespace Graph_lib;
using namespace Text_lib;

vector<Line> vec; // error: 'Line' is ambiguous
```

- we can use a using-directive to request that every name from a namespace be accessible in our scope without qualification.
- Overuse can lead to exactly the name clashes that namespaces were introduced to avoid.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Argument-Dependent Lookup ADL

- A function taking an argument of user-defined type X is more often than not defined in the same namespace as X. Consequently, if a function isn't found in the context of its use, we look in the namespaces of its arguments.

```
namespace shapes {
struct point {
  int x;
  int y;
};
void print(point&) {
  // ...
}
}  // namespace shapes


void fun(shapes::point& pt) { print(pt); }
```

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Argument-Dependent Lookup ADL

```
namespace N {
template <class T>
void f(T, int);  // N::f()
class X {};
}  // namespace N


namespace N2 {
N::X x;
void f(N::X, unsigned);
void g() {
  f(x, 1);  // calls N::f(X,int)
}
}  // namespace N2
```

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Thank you