Topic 14 Namespace

Namespace

- A namespace is a mechanism for expressing logical grouping.
  - Namespace is similar to package in Java; however, package involves physical containmnet
- A namespace is a collection of name definitions

```
namespace namespace-name {
        class-definitions ;
        variable-declarations ;
} ;
```

Namespace: An example

```
# include <iostream>
# include <cmath>
using namespace std ;

namespace MyNamespace {
   class Complex {
      float r, i ;
     public:
      Complex(float _r = 0.F, float _i = 0.F) { r  = _r ; i = _i ; }
     bool operator < (const Complex& c) const { return size() < c.size() ; }
      float size() const { return sqrt(r*r + i*i) ; }
      friend ostream& operator << (ostream& os, const Complex& c) ;
   } ;
   ostream& operator << (ostream& os, const Complex& c) {
      os << '(' << c.r << ',' << c.i << ')' ;
      return os ;
   }
} ;
```

Namespace: An example

- To access a member in a namespace, qualifier is needed !

```
int main() {
   MyNamespace::Complex c1(1, 1) ;
   MyNamespace::Complex c2(2, 2) ;

   if ( c1 < c2 )
      cout << c1 << endl ;
   else
      cout << c2 << endl ;
}
```

Typical Usage of namespace

- Namespace is usually used to avoid name conflict!

```
namespace NS1 {
   void sort(int v[], int size) {} // bubblesort

} ;
namespace NS2 {
   void sort(int v[], int size) {} // quicksort
} ;

int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;

   sort(iArray, 5) ; // error: NS1::sort or NS2::sort ?
   NS1::sort(iArray, 5) ;
   NS2::sort(iArray, 5) ;
}
```

using directive

- Make all names in a namespace available
- using directive allows us to avoid tedius qualification

```
using namespace MyNamespace ;

int main() {
   Complex c1(1, 1) ;     // MyNamespace:: is not necessary !
   Complex c2(2, 2) ;

   if ( c1 < c2 )
      cout << c1 << endl ;
   else
      cout << c2 << endl ;
}
```

using directive

- Without using namespace std

```
# include <iostream>
# include <cmath>
//using namespace std ;

namespace MyNamespace {
   class Complex {
      float r, i ;
     public:
      Complex(float _r = 0.F, float _i = 0.F) { r  = _r ; i = _i ; }
      bool operator < (const Complex& c) const { return size() < c.size() ; }
      float size() const { return sqrt(r*r + i*i) ; }
      friend std::ostream& operator << (std::ostream& os, const Complex& c) ;
   } ;
   std::ostream& operator << (std::ostream& os, const Complex& c) {
      os << '(' << c.r << ',' << c.i << ')' ;
      return os ;
   }
} ;
```

using directive

```
int main() {
   MyNamespace::Complex c1(1, 1) ;
   MyNamespace::Complex c2(2, 2) ;

   if ( c1 < c2 )
      std::cout << c1 << std::endl ;
   else
      std::cout << c2 << std::endl ;
}
```

using declaration

- Make a particular name available

```
# include <iostream>
# include <cmath>
using std::ostream ;

namespace MyNamespace {
   class Complex {
      float r, i ;
     public:
      Complex(float _r = 0.F, float _i = 0.F) { r  = _r ; i = _i ; }
      bool operator < (const Complex& c) const { return size() < c.size() ; }
      float size() const { return sqrt(r*r + i*i) ; }
      friend ostream& operator << (ostream& os, const Complex& c) ;
   } ;
   ostream& operator << (ostream& os, const Complex& c) {
      os << '(' << c.r << ',' << c.i << ')' ;
      return os ;
   }
} ;
```

using declaration

```
using MyNamespace::Complex ;

const Complex& min(const Complex &c1, const Complex &c2)
{
   return ( c1 < c2 ) ? c1 : c2 ;
}

int main() {
   using std::cout ;
   using std::endl ;

   Complex c1(1, 1) ;
   Complex c2(2, 2) ;

   Complex c3 = min(c1, c2) ;
   cout << c3 << endl ;
}
```

global namespace

- Without namespace, it is within the global namespace

```
// no namespace declaration ➔ global namespace
void sort(int v[], int size) ;

int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;
   sort(iArray, 5) ; // is equal to ::sort(iArray, 5) ;
}
```

global namespace

- Multiple definitions of the same variable/function in the global namespace are not allowed!

```
// globalsort1.cpp

void sort(int v[], int size) {} // bubble sort
int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;
   sort(iArray, 5) ; // multiple definions of sort
}
```

```
// globalsort2.cpp

void sort(int v[], int size) {} // selection sort
```

Unnamed namespace

- A namespace without name is an unnamed namespace

```
// namespace with no name ➔ unnamed namespace
namespace {
   void sort(int v[], int size) ;
}

int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;
   sort(iArray, 5) ; // is equal to ::sort(iArray, 5) ;
}
```

Unnamed namespace

- Every compilation unit(implementation file) has an unnamed namespace.

```
// unnamedsort1.cpp
namespace {
   void sort(int v[], int size) {} // bubble sort
}
int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;
   sort(iArray, 5) ; // invoke the sort in the same file
}
```

```
// unnamedsort2.cpp
namespace {
   void sort(int v[], int size) {} // selection sort
}
void f(int v[], int size) {
   sort(v, size) ;
}
```

Unnamed namespace

- Unnamed namespace is preferable to static declaration

```
// staticsort1.cpp
static void sort(int v[], int size) {} // bubble sort

int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;
   sort(iArray, 5) ; // invoke the sort in the same file
}
```

```
// staticsort2.cpp
static void sort(int v[], int size) {} // selection sort
void f(int v[], int size) {
   sort(v, size) ;
}
```

Namespaces are open

- More names can be added by several namespace declarations

```
// MyException.h
#ifndef __MyException_H
#define __MyException_H
namespace MyExceptionNS {
   class MyException ;
}
#endif
```

```
// RangeException.h
#ifndef __RangeException_H
#define __RangeException_H
# include "MyException.h"
namespace MyExceptionNS {
   class RangeException :
      pubilc MyException ;
}
#endif
```

```
// main.cpp

#include "MyException.h"
#include "RangeException.h"

using namespace MyExceptionNS ;

int main() {
   try {
      ...
   } catch ( const RangeException& e) {
   } catch ( const MyException& e) {
   }
}
```

namespace and overloading

- Overloading works across namespaces

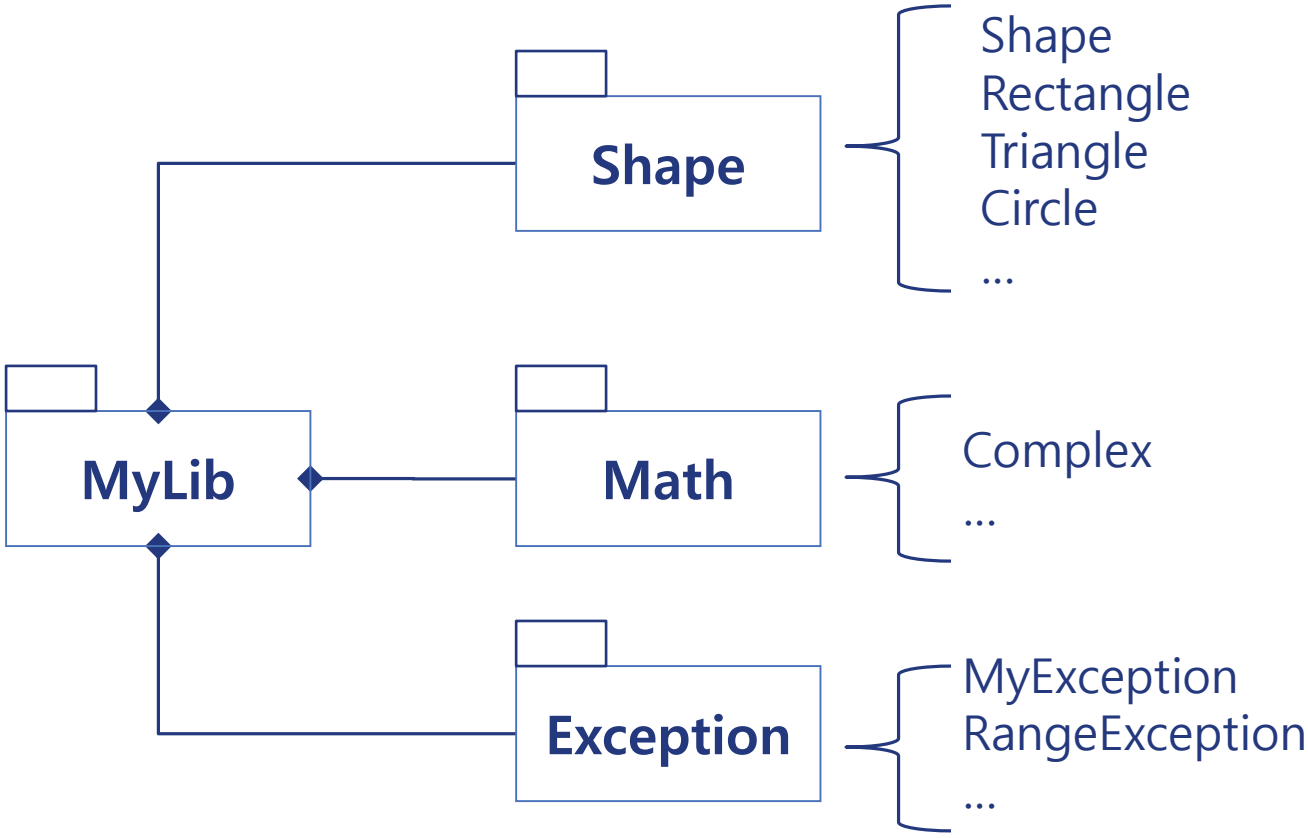
```
// NS1.h
#ifndef __NS1_H
#define __NS1_H
namespace NS1 {
   void print(int) ;
}
#endif
```

```
// NS2.h
#ifndef __NS2_H
#define __NS2_H
namespace NS2 {
   void print(char) ;
}
#endif
```

```
// main.cpp

#include "NS1.h"
#include "NS2.h"

using namespace NS1 ;
using namespace NS2 ;

int main() {
   print(100) ;       // invoke NS1::print(int)
   print('A') ;        // invoke NS2::print(char)
}
```

Namespace Hierarchy

- Namespaces can be nested; many classes can be hierarchically organized.



Namespace Hierarchy

```
namespace MyLib {
   namespace Shape {
      class Shape {} ;
      class Rectangle : public Shape {} ;
      class Triangle : public Shape {} ;
      class Circle : public Shape {} ;
   }
}

namespace MyLib {
   namespace Exception {
      class MyException {} ;
      class RangeException :
         public MyException {} ;
   }
}
```

```
using MyLib::Shape::Triangle ;
using namespace MyLib::Exception ;

int main() {
   try {
      Triangle tr ;
   } catch ( const MyException& e) { }
}
```

Namespace Hierarchy: Practically

```
// shape.h
namespace MyLib {
   namespace Shape {
      class Shape {} ;
   }
}
```

```
// triangle.h
# include "shape.h"
namespace MyLib {
   namespace Shape {
      class Triangle :
         public Shape {} ;
   }
}
```

```
// myexception.h
namespace MyLib {
   namespace Exception {
      class MyException {} ;
   }
}
```

```
# include "triangle.h"
# include "myexception.h"

using namespace MyLib::Shape ;
using namespace MyLib::Exception ;

int main() {
   try {
      Triangle tr ;
   } catch ( const MyException& e) { }
}
```

Compilation-once principle should be enforced

Namespace Aliases

- An alias can refer to a long name of a name space

```
namespace Pusan_National_University_Computer_Engineering_Lib {
   void sort(int v[], int size) {}
} ;

namespace PNU_CE = Pusan_National_University_Computer_Engineering_Lib ;
int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;
   PNU_CE::sort(iArray, 5) ;
}
```

Namespace and Evolution

- Program can be evolved by replacing one version with another

```
// Lib_v1.h

namespace PNU_Lib_v1{
   // bubble sort
   void sort(int v[], int size) {}
}
```

```
# include "Lib_v1.h"
namespace PNU_Lib = PNU_Lib_v1 ;

using namespace PNU_Lib ;
int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;
   sort(iArray, 5) ; // PNU_Lib_v1::sort() 호출
}
```

Namespace and Evolution

- The program can be evolved by using Lib v2

```
// Lib_v2.h

namespace PNU_Lib_v2{
   // quick sort
   void sort(int v[], int size) {}
} ;
```

```
# include "Lib_v2.h"
namespace PNU_Lib = PNU_Lib_v2 ;

using namespace PNU_Lib ;
int main() {
   int iArray[] = {0, 10, 5, 9, 3} ;
   sort(iArray, 5) ; // PNU_Lib_v2::sort() 호출
}
```

Practice #1

- determine namespaces from a collection of classes below

> Monitor, Mouse, Won, Dollar, MainMemory, Franc, HardDisk, Log, Exp, Min, Max, RangeError, AllocError, NoSocketError, List, Stack, Queue, HashTable

- Answer
  - Namespace1 – class1, class2, …
  - Namespace2 – class1, class2, …
  - …

Practice #2

- Implement Stack and StackException so that the main function works correctly.

```
using namespace MyLib::Collection ;
using namespace MyLib::Exception ;
int main() {
   Stack<int, 3> is ;
   while ( true ) {
      char cmd ; int value ;
      cin >> cmd ;
      try {
         switch ( cmd ) {
            case 'i' : { cin >> value ;
              is.Push(value) ; cout << value << " is pushed!" << endl ; break ;
                  }
            case 'x': {
              value = is.Pop() ; cout << value << " is popped!" << endl ; break ;
                  }
            default : { cout << "Invalid command!" << endl ; return 0 ; }
         }
      }
      catch ( const StackException& e ) { e.print() ; }
   }
}
```