Namespaces, friends & smart pointers

Taller de Programación I - FIUBA

Namespaces

Taller de Programación I - FIUBA

Namespaces: concepto

- **namespace**
  - Los **namespaces** (espacios de nombres) proporcionan un método para evitar conflictos de nombres en proyectos grandes.
  - Los símbolos declarados dentro de un bloque **namespace** se colocan en un ámbito con nombre que evita que se confundan con símbolos con nombres idénticos en otros ámbitos.
- **using**
  - Introduce un nombre que se definió en otro lugar, en la región declarativa donde aparece

namespace: Ejemplo

```
namespace Q
{
   namespace V              // V es un miembro de Q. Está completamente definido dentro de Q
   {
      class C{ void m(); }; // C es miembro de V (y completamente definido dentro de V)
                            // C::m solamente se encuentra declarada
      void f();             // f es una función miembro de V. Solamente está declarada aquí
   }

   void V::f()              // Definición de f (miembro de V) realizada fuera de V
   {
   }

   void V::C::m()           // Definición de V::C::m fuera del namespace y de la clase
   {}
}
```

using: Ejemplo

```
namespace Q
{
   void A() {}
   void B() {}
}

Q::A();                    // Se necesita usar Q::A()

using namespace Q;         // Se Importa Q
A();                       // OK. Equivale a Q::A();
B();                       // OK. Equivale a Q::B();
```

Namespaces: Extensión

```
namespace Graphics {
  namespace Effects {
     class Animation { ... };
  }
}

namespace Graphics {
  class Vertex { ... };
  class Line { ... };
}
```

Permite agrupar funciones, clases, tipos y otros elementos bajo un cierto nombre.

Es una agrupación lógica, independiente de la ubicación física de los elementos al momento de compilación.

A diferencia de una clase, un **namespace** puede ser extendido.

Using: Importación parcial o total

```
Graphics::Vertex v = Graphics::Vertex();
Graphics::Line l = Graphics::Line();

using Graphics::Vertex;
Vertex v = Vertex();
Graphics::Line l = Graphics::Line();

using namespace Graphics:
Vertex v = Vertex();
Line l = Line();
```

Para acceder a los elementos dentro de un **namespace** hay que nombrarlos incluyendo el nombre del **namespace**.

Otra forma es importando algún elemento del **namespace** con la directiva **using**.

O bien importar todos los elementos del **namespace** con **using namespace**.

Objetos Friends

Taller de Programación I - FIUBA

Friend

```
class List {
    friend class ListIterator;
    void* head;  // private
};

class ListIterator {
     void* get_first_elem(List *l) {
          return l->head;
     }
};
```

Una clase **friend** puede acceder a los atributos y métodos privados.

Es posible hacer **friend** no solo a una clase sino a un solo método.

friend no es transitivo: el amigo de mi amigo no es necesariamente mi amigo.

Caso de Estudio - Clase Complejo

```
1   class Complex {
2     float re;
3     float im;
4   public:
5     Complex(float re, float im) : re(re), im(im) {
6     }
7     Complex(const Complex& other) : re(other.re), im(other.im) {
8     }
9     ... //operators overloading
10    float getRe() const { return re; }
11    float getIm() const { return im; }
12  };
```

Ejemplo - Operator+ (modo binario)

```
1   class Complex {
2     ...
3   };
4   Complex operator+(const Complex& a, const Complex& b) {
5     Complex result(a.getRe() + b.getRe(),
6                    a.getIm() + b.getIm());
7     return result;
8   }
9   ...
10  Complex var1(1,2);
11  Complex var2(1,3);
12  Complex var3 = var1 + var2;
```

Ejemplo - Operator+ (modo binario - friend)

```
1   class Complex {
2     ...
3     friend Complex operator+(const Complex& a, const Complex& b);
4   };
5
6
7   Complex operator+(const Complex& a, const Complex& b) {
8     Complex result(a.Re + b.Re, a.Im + b.Im);
9     return result;
10  }
11
12
```

Smart pointers

Taller de Programación I - FIUBA

Smart pointers: Concepto

- std::**unique_ptr**
  - es un puntero inteligente que posee y administra otro objeto a través de un puntero. Solo un objeto **unique_ptr** puede poseer el objeto y elimina ese objeto cuando el **unique_ptr** queda fuera de alcance (RAII)
- std::**shared_ptr**
  - es un puntero inteligente que posee y administra otro objeto a través de un puntero. Varios objetos **shared_ptr** pueden poseer el mismo objeto. El objeto es destruido cuando el último objeto pierde la referencia (RAII)
- std::**weak_ptr**
  - es un puntero inteligente que contiene una referencia no propietaria ("débil") a un objeto administrado por std::shared_ptr. Debe convertirse a std::shared_ptr para acceder al objeto al que se hace referencia.

Smart pointers

```
std::unique_ptr p1 = new Object();
std::unique_ptr q1 = p1;                    // transfiere ownership

std::shared_ptr p2 = new Object();
std::shared_ptr q2 = p2;                    // comparte ownership

std::weak_ptr p3 = p2;                      // accede pero no comparte ownership
```

Asignación por Move

```
1   struct Vector
2   {
3           int *data;
4           int size;
5           Vector& operator=(Vector&& other)
6           {
7                   if (this == &other)
8                   {
9                           return *this;                   // other is myself!
10                  }
11          if (this->data)
12                  free(this->data);
13          this->data = other.data;
14          this->size = other.size;
15          other.data = nullptr;
16          other.size = 0;
17          return *this;
18          }
19  }
```