



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Funciones y Clases Amigas (Friend) Miembros Estáticos (Static)

Agustín J. González
ELO329


Funciones y Clases Friend

- El calificador **friend** se aplica a **funciones globales** y **clases** para otorgar acceso a miembros privados y protegidos de una clase.
- La función función global “*friend*” tendrá acceso a los miembros como si fuera un método de la clase.
- Una clase *friend* B es aquella cuyos métodos tiene acceso a los miembros privados y protegidos de otra clase A que la ha declarado friend.

Ejemplo de función Friend:

```
class Course {  
public:  
    friend bool ValidateCourseData(const Course &C);  
private:  
    int nCredits;  
    //...  
};
```

Función global No exclusiva de la clase!! Solo prototipo, su implementación no pertenece a la clase



Implementación de ValidateCourseData()

- El calificador friend no aparece en la implementación de la función global.
- Notar el acceso a miembros privados (`nCredits`) de la clase. Es posible por ser función amiga.

```
bool ValidateCourseData(const Course & C) {  
    if( C.nCredits < 1 || C.nCredits > 5 )  
        return false;  
}  
return true;  
}
```

Funciones Friend, otro ejemplo

```
class Complex {  
public:  
    Complex( double re, double im );  
    friend Complex operator*( double factor, Complex z );  
private:  
    double real, imag;  
};
```

Solo prototipo



```
Complex operator*(double factor, Complex z ) // implementación  
{  
    return Complex( factor*z.real, factor*z.imag );  
}
```

En este Ejemplo, la función `operator*` tiene acceso a los miembros privados de `Complex`. Notar la sobrecarga de operaciones del tipo: $z2 = 3 * z1$; cosa que no podemos hacer como método de la clase.

Ahora Clases Friend: clases amigas

```
// Example of a friend class
class YourClass {
    // .....
    friend class YourOtherClass; // Declare a friend class
private:
    int topSecret;
};

class YourOtherClass{
public:
    void change(YourClass & yc);
};

void YourOtherClass::change(YourClass & yc) {
    yc.topSecret++; // Puede acceder datos privados
}
```

Una clase amiga (friend) es una clase cuyas funciones miembros son como funciones miembros de la clase que la hace amiga. Sus funciones miembros tienen acceso a los miembros privados y protegidos de la otra clase.

Clases Friend (cont.)

- La “Amistad” no es mutua a menos que explícitamente sea especificada. En el ejemplo previo, los métodos de YourClass no pueden acceder a miembros privados de YourOtherClass.
- La “Amistad” no se hereda; esto es, clases derivadas de YourOtherClass no pueden acceder a miembros privados de YourClass. Tampoco es transitiva; esto es clases que son “friends” de YourOtherClass no pueden acceder a miembros privados de YourClass.
- La “amistad” es importante en sobrecarga de operador <<, para escritura a pantalla pues en este caso no podemos agregar sobrecargas a clases estándares. Ver ejemplo CVectorFriend

Static: Miembros Estáticos

- **No hay gran diferencia con Java**
- Estas variables tienen existencia desde que el programa se inicia hasta que termina.
- Atributos estáticos
 - El atributo es compartido por todas las instancias de la clase. Todas las instancias de la clase comparten el mismo valor del atributo estático. Igual que en Java.
- Métodos Estáticos
 - Estos métodos pueden ser invocados sobre la clase, no solo sobre una instancia en particular. Igual que en Java.
 - El método sólo puede acceder miembros estáticos de la clase
- Es posible pensar en miembros estáticos como atributos de la clase y no de objetos. Hasta aquí igual a Java.

Declaración de Datos Estáticos

- La palabra clave **static** debe ser usada.

```
class Student {  
    //...  
    private:  
        static int m_snCount; //instance m_snCount  
};
```



Creación de un contador de instancias

- La inicialización del dato estático no se efectúa en el constructor pues existe previo a la creación de cualquier objeto.
- En Java lo hacíamos en bloque de iniciación static
static { }
- La iniciación es una diferencia entre C++ y Java.

```
// student.cpp
```

```
int Student::m_snCount = 0;
```

Asigna memoria e inicia el valor de partida. Se ejecuta antes de ingresar al main.



Creación de un Contador de Instancias

- Usamos el constructor y destructor para incrementar y decrementar el contador:

```
Student::Student() {  
    m_snCount++;  
}  
  
Student::~~Student() {  
    m_snCount--;  
}
```

Métodos Estáticos

- Usamos métodos estáticos para permitir el acceso público a miembros de datos estáticos sin necesidad de instanciar la clase.

```
class Student {  
public:  
    static int get_InstanceCount();  
  
private:  
    static int m_snCount; // instance count  
};
```

Llamando a Métodos Estáticos

- Usamos ya sea el nombre de la clase o una instancia de la clase para acceder al método:

```
cout << Student::get_InstanceCount(); // 0
Student S1;
Student S2;
cout << Student::get_InstanceCount(); // 2
cout << S1.get_InstanceCount(); // 2
```