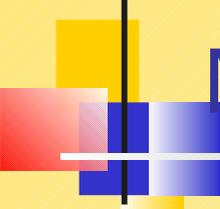




UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

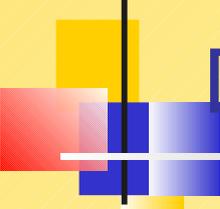
Miembros Estáticos (Static) de Clases y Funciones Amigas (Friend)

Agustín J. González
ELO329



Miembros Estáticos de una Clase

- Estas variables tiene existencia desde que el programa se inicia hasta que termina.
- Datos Miembro estáticos
 - Una instancia de la variable es compartida por todas las instancias de la clase. Todos los objetos de esa clase comparten el mismo valor del miembro estático
- Funciones miembro Estáticas
 - Estas funciones pueden ser invocadas sobre la clase, no solo sobre una instancia en particular.
 - La función sólo puede acceder miembros estáticos de la clase
- Es posible pensar en miembros estáticos como atributos de la clase y no de objetos.
- **No hay gran diferencia con Java**



Declaración de Datos Estáticos

- La palabra clave static debe ser usada.

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

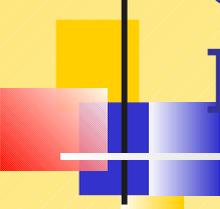
Creación de un contador de instancias

- La inicialización del dato estático no se efectúa en el constructor pues tiene su existencia previo a la creación de cualquier objeto.
- Un miembro estático puede contar el número de instancias de una clase.

```
// student.cpp
```

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

Asigna memoria e inicia el valor de partida



Creación de un Contador de Instancias

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

```
Student::Student ()
```

```
{
```

```
    m_snCount++;
```

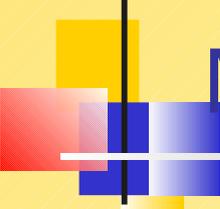
```
}
```

```
Student::~~Student ()
```

```
{
```

```
    m_snCount--;
```

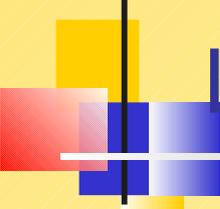
```
}
```



Miembros de Función Estáticos

- Usamos miembros de función estáticos para permitir el acceso público a miembros de datos estáticos.

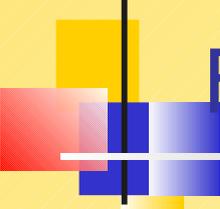
```
class Student {  
public:  
    static int get_InstanceCount();  
  
private:  
    static int m_snCount; // instance count  
};
```



Llamando a Funciones Estáticas

- Usamos ya sea el nombre de la clase o una instancia de la clase como calificador:

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

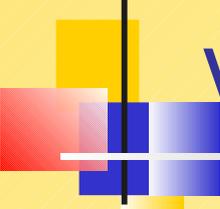


Funciones Friend

- Una función Friend es una función que no es miembro de una clase pero tiene acceso a los miembros privados y protegidos de la clase.
- Es una función global con acceso como si fuera de la clase.

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

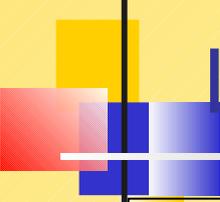
Función global No exclusiva de la clase!! Solo prototipo, su definición no pertenece a la clase



ValidateCourseData()

- El calificador friend no aparece en la implementación de la función
- Notar el acceso a miembros privados de la clase

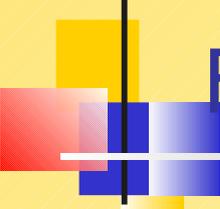
```
bool ValidateCourseData(const Course & C)
{
    if( C.m_nCredits < 1 || C.m_nCredits > 5 )
        return false;
    }
    return true;
}
```



Llamado a ValidateCourseData()

```
void Transcript::ReadFromFile(ifstream &infile)
{
    Course aCourse;
    int count;
    infile >> count;
    for(int i = 0; i < count; i++)
    {
        aCourse.ReadFromFile( infile );

        if( ValidateCourseData( aCourse ) )
            m_vCourses.push_back( aCourse );
        else
        {
            cout << "Invalid course data encountered: ";
            aCourse.Display();
        }
    }
}
```



Funciones Friend, otro ejemplo

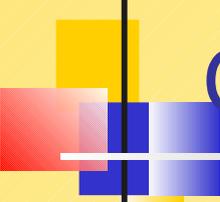
```
class Complex {  
public:  
    Complex( float re, float im );  
    friend Complex operator+( Complex first, Complex second );  
private:  
    float real, imag;  
};
```

Sólo prototipo



```
Complex operator+( Complex first, Complex second ) // implementación  
{  
    return Complex( first.real + second.real, first.imag + second.imag );  
}
```

En este Ejemplo, la función operator+ tiene acceso a los miembros privados de Complex



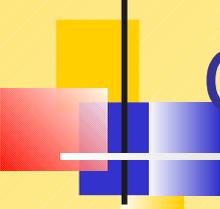
Classes Friend (1)

```
// Example of the 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 (2)

- La "Amistad" no es mutua a menos que explícitamente sea especificada. En el ejemplo previo, los miembros función 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.
- Es importante en sobre carga del operador << para salida de datos en clases estándares. Ver ejemplo item.h e Item.cpp