UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Electrónica

Herencia en C++

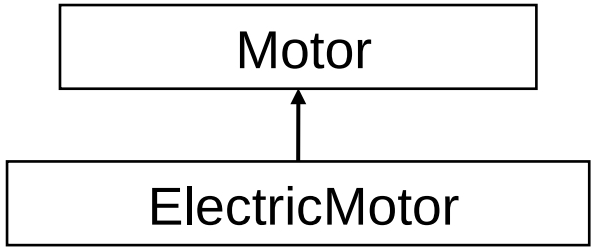
ELO329: Diseño y Programación Orientados a Objetos

Departamento de Electrónica

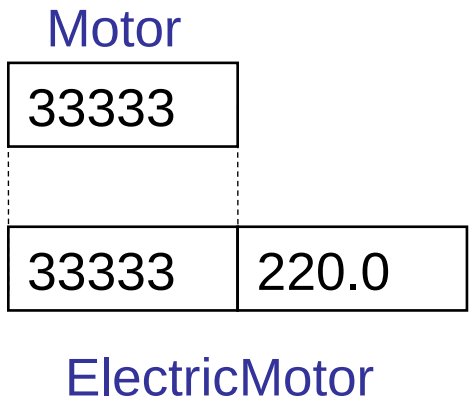
Universidad Técnica Federico Santa María

Motor y ElectricMotor

- Consideremos dos clases que tienen algo en común.



- Un objeto ElectricMotor contiene el mismo número de identificación (ID) como un Motor, más el voltaje.



Clase CMotor

- Definición de la clase CMotor (en CMotor.h):

```
class CMotor {
public:
    CMotor() { }
    CMotor( const string & id );
    string get_ID() const;
    void set_ID(const string & s);
    void Display() const;
    void Input();

private:
     string m_sID;
};                              // más...
```

En CMotor.cpp

```
CMotor::CMotor( const string & id ){
     set_ID(id);
}
string CMotor::get_ID() const {
   return m_sID;
}
void CMotor::set_ID(const string & s) {
    m_sID = s;
}
void CMotor::Display() const {
          cout << "[CMotor] ID=" << m_sID;
}
void Cmotor::Input() {
          string temp;
          cout << "Enter Motor ID: ";
          cin >> temp;
          set_ID(temp);
}
```

Creación de Clases Derivadas

```
class base {
...
};

class derivada : public base {
...
};
```

La clase base debe aparecer primero en las declaraciones. Típicamente vía un “base.h”

Nivel de acceso, puede ser:
public, protected, private u omitido
(omitido equivale a private).

Un clase puede derivar de más de una clase base (ojo, ésta es una diferencia con JAVA)

Clase CElectricMotor

Público equivale al uso de Java

```
class CElectricMotor : public CMotor  {
public:
  CElectricMotor();
  CElectricMotor(const string & id,  double volts);

  void Display() const;
        void Input();
        void set_Voltage(double volts);
        double get_Voltage() const;

private:
        double m_nVoltage;
};
```

Inicialización de Clase Base

- Para inicializar los atributos definidos en la clase base se llama al constructor de la clase base. En este ejemplo, el número ID del motor es pasado al constructor de CMotor.

```
CElectricMotor::CElectricMotor(const string & id, double volts) : CMotor(id) {
    m_nVoltage = volts;
}
```

- Estructura general:

```
derived_constructor (parameters) : base_constructor (parameters) {...}
```

Llamando a métodos de la clase base

```
void CElectricMotor::Input()
{
    CMotor::Input();      // llama método de la clase base
                          // En java lo hacíamos con super.Input()
    double volts;
    cout << "Voltage: ";
    cin >> volts;
    set_Voltage(volts);
}
```

La función Input existe en ambas clases CMotor y CElectricMotor. En lugar de duplicar el código ya escrito, se llama al método correspondiente en la clase base.

Obligatorio si los atributos involucrados son privados en clase base.

Llamando a métodos de la clase base

- La función miembro Display funciona de la misma manera. Ésta llama a CMotor::Display primero.

```
void CElectricMotor::Display() const {
        // call base class function first
        CMotor::Display();  // muestra atributos de CMotor.

        cout << "  [CElectricMotor]"
      << " Voltage=" << m_nVoltage << endl;
}
```

Probando Clases

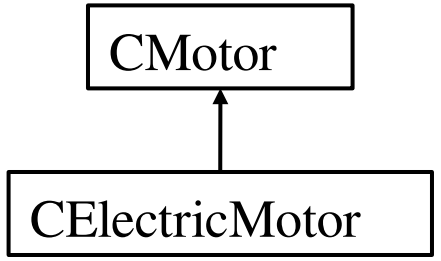
- Cuando el mismo nombre de método existe en ambas clases, C++ llama al método implementado para la clase según la declaración del objeto. Éste es el caso con los métodos Input y Display (es lo esperable):

```
CElectricMotor elec;       // CelectricMotor
elec.Input();          // CElectricMotor

elec.Display();        // CElectricMotor
```

Asignación de Objetos

- Podemos asignar objetos de clases derivadas a un objeto de la clase base.




```
CMotor mot;
CElectricMotor elec;

mot = elec;              // se copian sólo los atributos de Motor
elec.get_Voltage();      // ok
mot.get_Voltage();       // error, no es motor eléctrico
elec = mot;              // error, voltaje no está definido
```

Asignación errada de Objetos

- Pero no podemos asignar una instancia de una clase base a una instancia de una clase derivada (similar restricción en Java). Algunos campos de la clase derivada no existen en la clase base.

```
CMotor mot;
CElectricMotor elec;
elec = mot;                 // error, No se permite.
                            // pues el voltaje sólo existe en
                            // CElectricMotor.
```

Acceso a miembros Protected (Protegidos)

- Miembros de una clase designados como protected son visibles a ambas: la clase base y las clases derivadas (y a clases amigas -friend- pero a nadie más). Es análogo a Java.

```
class CMotor {
public:
  CMotor() { }
  CMotor( const string & id );
protected:
  string get_ID() const;
  void set_ID(const string & s);
//...
}
```

Herencia Protegida

- Supongamos por un momento que CMotor usa miembros públicos para todos sus métodos:

```
class CMotor {
public:
  CMotor() { }
  CMotor( const string & id );
  string get_ID() const;
  void set_ID(const string & s);

//...
}
```

Herencia Protegida

- Podemos usar el calificador protected cuando declaramos una clase derivada.
- **Con herencia protegida, todos los métodos públicos en la clase base pasan a ser protegidos en la clase derivada. Los restantes mantienen su nivel de acceso**.

```
class CElectricMotor : protected CMotor  {

  //...
};
```

Herencia Protegida

```
class CElectricMotor : protected CMotor
     {
  //...
};
```

- Por ejemplo, el programa principal no puede llamar set_ID y get_ID en un motor eléctrico porque ahora esos métodos son protegidos para instancias de motor eléctrico.

```
CElectricMotor EM;

EM.set_ID("12345");    // error, set_ID está en CMotor, es protegido aquí

EM.get_ID();           // error, get_ID está en CMotor, ídem.
```

Herencia Privada

- La herencia privada causa que todos los métodos declarados en la clase base tienen acceso privado en la clase derivada.
- Pareciera que no hay diferencia con herencia protegida: En ambos casos, métodos de CElectricMotor pueden acceder a métodos de CMotor…

```
class CElectricMotor : private CMotor  {
        //...
};
```

Herencia Privada

- Pero cuando derivamos una nueva clase (CPumpMotor) de CElectricMotor, la diferencia se hace notar: con herencia privada, métodos en CPumpMotor no pueden acceder a miembros públicos de CMotor.

```
class CPumpMotor : public CElectricMotor {
public:
  void Display() {
    CMotor::Display();              // not accessible!
    CElectricMotor::Display();      // this is OK
  }
};
```

Miembros que no son heredados

- El constructor y destructor de la clase base no son heredados, deben ser definidos en la clase heredada.
- El constructor por omisión y el destructor sí son llamados cuando se crea o destruye una instancia de la clase derivada.
- Si la clase base tiene sobrecargado el operador =, éste no se hereda en la clase derivada.
- La relación de “amistad” no se hereda. Las clases y funciones friend no son friend en la clase hija.

struct es una palabra reservada. Puede ser usada en lugar de la palabra reservada class. La principal diferencia es que al omitir el nivel de acceso, éste se entiende público y no privado como con class.