

Herencia

Conceptos

- El principal objetivo de la herencia es la **simplificación**, logrando que el código sea más extensible/mantenible/reutilizable
- Utilizando herencia desarrollamos aplicaciones complejas más rápidamente, evitando “*copy-paste*” de código y modificar código existente:
 - reutilizamos aquellas partes que necesitamos
 - heredamos lo que queremos y aumentamos sus funcionalidades

Conceptos

- La herencia solamente funciona con clases, no funciona con funciones no-miembro o variables
- Las clases que heredan de otra clase se llaman clases **derivadas** o clases **hijas**
- La clase de la que heredan las otras se llama clase **base** o clase **padre**
- La herencia puede tener varios niveles (una clase hereda de otra, que a su vez hereda de otra...)

Conceptos

- Aunque C++ soporta herencia **múltiple** (una clase hija tiene más de una clase padre y hereda de ambas), raramente se usa
- Prácticamente cualquier problema de diseño de clases puede resolverse con herencia **simple** (una clase hija tiene únicamente una clase padre)
- Se suele usar la analogía del árbol de directorios para explicar la herencia simple y la del árbol genealógico para la herencia múltiple

¿Qué se hereda?

- Por defecto, la clase hija hereda *toda* la parte **pública** de la clase padre (métodos y atributos públicos)
- Por tanto en la clase hija no necesitamos volver a declararlos ni implementarlos → *reutilización*
- Normalmente los atributos no son públicos: podemos usar el especificador **protected**, para que sean accesibles a la clase hija pero no al resto del programa
- Los constructores **NO** se heredan nunca


Ejemplo

```
class PersonAddr
{ // base class
public:
    PersonAddr();
    PersonAddr read_info();
    void print();
protected:
    char name[30];
    char addr[30];
    char city[20];
};
```

```
class Patient : PersonAddr
{ // derived class
public:
    Patient();
    void send_bill();
private:
    char phone[10];
    char birthdate[8];
    double balance;
};
```

Ejemplo



```
class PersonAddr
{ // base class
public:
    PersonAddr();
    PersonAddr read_info();
    void print();
protected: 
    char name[30];
    char addr[30];
    char city[20];
};
```

```
class Patient : PersonAddr
{ // derived class
public:
    Patient();
    void send_bill();
private:
    char phone[10];
    char birthdate[8];
    double balance;
};
```

¿Cómo se hereda?

- La forma de acceso a la clase base también puede ser **public/protected/private**

```
class Patient : public PersonAddr  
class Patient : protected PersonAddr  
class Patient : private PersonAddr
```
- **private** (por defecto): solo se hereda lo public/protected, que será private en la clase hija
- **public** (lo más habitual): todo lo heredado mantiene su condición (public/protected) en la clase hija
- **protected**: lo que fuera protected en la clase padre mantiene su condición, mientras que lo que fuera public pasa a ser protected en la clase hija

Ejemplo

```
class Base { // base class
    int i;
public:
    void set_i(int value) { i = value; }
    void print() { cout << i << endl; }
};

class Derived1 : public Base { // first hierarchy level
    int j;
};

class Derived2 : public Derived1 { // second hierarchy level
    int k;
};
```

Ejemplo

```
int main() {  
    Base father;  
    Derived1 son;  
    Derived2 grandson;  
  
    father.set_i(123);  
    son.set_i(456);  
    grandson.set_i(789);  
    father.print();  
    son.print();  
    grandson.print();  
  
    return 0;  
};
```

Ejemplo

```
class Derived1 : public Base { // first hierarchy level
    int j;
public:
    void set_j(int value) { j = value; }
    void print() { cout << j << " " << i << endl; }
};
```

```
class Derived2 : public Derived1 { // second hierarchy level
    int k;
public:
    void set_k(int value) { k = value; }
    void print() { cout << k << " " << j << " " << i << endl; }
};
```

Ejemplo

```
class Derived1 : public Base { // first hierarchy level
    int j;
public:
    void set_j(int value) { j = value; }
    void print() { cout << j << endl; }
};
```

```
class Derived2 : public Derived1 { // second hierarchy level
    int k;
public:
    void set_k(int value) { k = value; }
    void print() { cout << k << endl; }
};
```

Ejemplo

```
int main() {  
    Base father;  
    Derived1 son;  
    Derived2 grandson;  
  
    father.set_i(123);  
    son.set_j(456);  
    grandson.set_k(789);  
    father.print();  
    son.print();  
    grandson.print();  
  
    return 0;  
};
```