Técnico superior en desarrollo de videojuegos

Programación 1

Unidad 1

Herencia

Constructores y Destructores en Herencia

Accesibilidad en Herencia

Herencia en C++

- Definir clases a partir de otras más generales y solo agregar las propiedades y métodos especializados.
- Implica reutilización de código a través de librerías de clases.
- Concepto de clase Base y clase Derivada (o superclase y subclase).

Herencia

- El concepto de herencia está presente en nuestras vidas diarias donde las clases se dividen en subclases, Así por ejemplo la clase vehículos se divide en automóviles, colectivos, camiones y motos.
- El principio de este tipo de división es que cada subclase comparte características comunes con la clase de la que se deriva. Los automóviles, camiones, colectivos y motos que pertenecen a la clase vehículo tienen ruedas y un motor; son estas las características de vehículos.
- Además de las características compartidas con otros miembros de la clase, cada subclase tiene sus propias características particulares: colectivos, por ejemplo, tienen un gran número de asientos, una televisión para los viajeros, mientras que las motos tienen dos ruedas, un manillar y un asiento doble.

Herencia

- En C++ la clase original se denomina ***clase base;*** las clases que se definen a partir de la clase base, compartiendo sus características y añadiendo otras nuevas, se denominan ***clases derivadas.***
- Las clases derivadas pueden heredar código y datos de su clase base añadiendo su propio código especial y datos a la misma.
- La herencia permite definir nuevas clases a partir de clases ya existentes.
- Por ejemplo, si se define una clase denominada figura geométrica se pueden definir las clases derivadas cuadrado, triángulo y círculo.

Herencia

Responde siempre a la sentencia “ES UN”

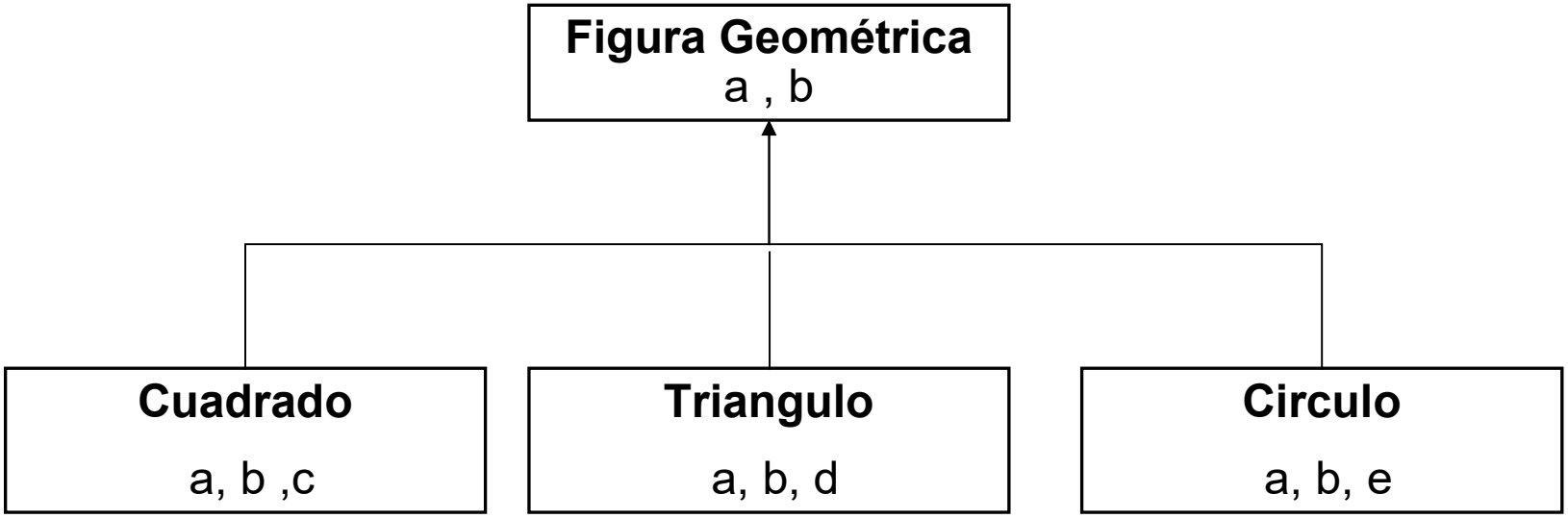

Figura Geométrica posee características a y b

Cuadrado posee características a, b por ser una figura geométrica y c propia de su clase

Cuadrado “es un”a figura geométrica

Triangulo “es un”a figura geométrica

Círculo “es un”a figura geométrica

Herencia

- La herencia impone una relación jerárquica entre clases en la cual una clase *derivada* hereda de su clase *base.* Si una clase sólo puede recibir características de otra clase base, la herencia se denomina *herencia simple.* Si una clase recibe propiedades de más de una clase base, la herencia se denomina *herencia múltiple.* Muchos lenguajes orientados a objetos no soportan herencia múltiple, sin embargo C++ sí incorpora esta propiedad.

Herencia

- La *herencia simple* es aquella en la que cada clase derivada hereda de una única clase, tiene un solo ascendiente..Cada clase puede tener, sin embargo, muchos descendientes.
- La *herencia múltiple* es aquella en la cual una clase derivada tiene más de una clase base.

Creación de una clase derivada

- Cada clase derivada se debe referir a una clase base declarada anteriormente.
  La declaración de una clase derivada es:

```
class Derivada: <especificadores de acceso> Base {
                        …
};
```

los especificadores de acceso pueden ser:

public, *protected* o *private*.

Clase DispElectronico


```
class DispElect{
    bool _encendido;  // indica si el disp esta encendido
public:
     DispElect( ){_encendido = false;}   // constructor por defecto
     void encender( ){ //si esta prendido lo apago y sino lo prendo
           if (!_encendido) _encendido=true;
           else                  _encendido=false;
          }

     bool getEncendido( ) { return _encendido;}
};
```

Clase Televisor


```
class Televisor : public DispElect {
        int _canal;
        int _brillo;
public:
        Televisor( ) { _canal = 2; _brillo = 15;}
        void setCanal ( int canal) { _canal = canal; }
        int getCanal( ) { return _canal;}
        void setBrillo( int brillo){ _brillo = brillo; }
        int getBrillo( ) { return _brillo;}
};
void main( ){
Televisor* sony=new Televisor();
     sony->encender( );
     sony->setcanal(12);
     delete sony;
}
```

Indica que hereda de la clase DispElect

Acá se observa la herencia ya que encender es de clase Base

Constructores y Destructores en Herencia

- Una clase derivada puede tener tanto constructores como destructores.
- Un constructor o destructor definido en la clase base debe estar coordinado con los encontrados en una clase derivada. Igualmente importante es el movimiento de valores de los atributos de la clase derivada a los atributos que se encuentran en la base**. En particular, se debe considerar cómo el constructor de la clase base recibe valores de la clase derivada para crear el objeto completo.**
- **Si un constructor se define tanto para la clase base como para la clase derivada, C++ llama primero al constructor base. Después que el constructor de la base termina sus tareas, C++ ejecuta el constructor derivado.**
- **Al contrario que los constructores, una función destructor de una clase derivada se ejecuta antes que el destructor de la clase base.**
- Por definición, un destructor de clase no toma parámetros.

Constructores y Destructores en Herencia

- Cuando una clase base define un constructor, éste se debe llamar durante la creación de cada instancia de una clase derivada, para garantizar la buena inicialización de los atributos que la clase derivada hereda de la clase base. En este caso la clase derivada debe definir a su vez un constructor que llama al constructor de la clase base proporcionándole los argumentos requeridos.
Un constructor de una clase derivada debe utilizar un mecanismo de pasar aquellos argumentos requeridos por el correspondiente constructor de la clase base.

Derivada::Derivada(tipo1 x, tipo2 y): Base (x, y) {...}

nota : x e y en Base pasan como argumentos no se coloca el tipo

- Otro aspecto importante es el orden en el que el compilador C++ inicializa las clases base de una clase derivada. Cuando el compilador C++ inicializa una instancia de una clase derivada, tiene que inicializar todas las clases base primero.

Invocación de Constructores en herencia 1º ejemplo

Si la clase Televisor hereda de DispElect, ¿en qué orden se invocarán los constructores?

Cuando se instancia un objeto de clase derivada antes de ejecutar el código de su constructor se invoca la constructor de la clase base. Esto se realiza en forma automática.

```
…
DispElect( ){cout<<“se va a encender el dispositivo”<<endl;}
…
Televisor( ){cout<<“televisor”;}
…
void main( ){
    Televisor* t=new Televisor();
…
```

Se verá en pantalla

se va a encender el dispositivo
televisor

Invocación de Constructores en herencia 2 ejemplo (con parámetros)

Si la clase Televisor hereda de DispElect, ¿cómo se invocarán los constructores si uno de ellos tiene parámetros?

Cuando se instancia un objeto de clase derivada con un constructor parametrizado de la clase base hay que pasarle este parámetro al código de su constructor.

```
DispElect( bool encendido ) { _encendido = encendido; }
    …
Televisor( ) { cout<<“televisor”; }
    …
void main( ){
    Televisor* t=new Televisor();
    …
```

Este código causará un error de compilación pues el parámetro no le llega al constructor de la clase base.

Invocación de Constructores en herencia 3 ejemplo (con parámetros)

Si la clase Televisor hereda de DispElect, ¿cómo se invocarán los constructores si tienen parámetros?

Cuando se instancia un objeto de clase derivada con un constructor parametrizado hay que pasarle este parámetro al código de su constructor.

```
DispElect( ) { _encendido = false; }
    …
Televisor(int canal ) { _canal= canal; }
    …
void main( ){
    Televisor* t=new Televisor(11);
    …
```

Este código se compila correctamente pues el parámetro le llega al constructor de la clase derivada al instanciar la clase. Se realiza en forma automática la llamada al constructor de base.

Invocación de Constructores en herencia 4º ejemplo(con parámetros)

Cuando se instancia un objeto de clase derivada con un constructor con parámetro hay que pasarle el parámetro al código de su constructor y si la base también tiene en su constructor parámetros hay que pasarlo a través del constructor de la clase derivada. Se realiza en forma manual.

```
DispElect( bool encendido){_encendido = encendido ;}


Televisor(bool encendido, int canal ) : DispElect(encendido){
    _canal=canal;
}
void main( ){
Televisor* t = new Televisor(true,12);
…
```

Este valor le llega al constructor de base

Se coloca un : y luego la llamada al constructor de la clase base con el parámetro que le corresponde

El valor true va al constructor de base y el 12 al de derivada

de esta manera será necesario mandar dos parámetros en la instancia del objeto ya que un parámetro le llega al constructor de la clase base y el otro al constructor propio.

Invocación de Destructores en herencia

El modo de invocación de los destructores es inverso al de constructores. Se realiza en forma automática.

```
…
~DispElect( ){cout<<“se va a apagar el dispositivo”;}
…
~Televisor( ){cout<<“¡gracias por verme!”<<endl; }
…
void main( ){
Televisor* t=new Televisor();
…
delete t;
```

Se verá por pantalla:

¡gracias por verme!

se va a apagar el dispositivo

Código completo

```
class DispElect{
        bool _encendido; // indica si el dispositivo esta encendido o no
public:
        DispElect( ){_encendido=false;}   // constructor por defecto
        DispElect(bool encendido){__encendido=encendido;}
                                          // constructor con parámetro
        ~DispElect( ){cout<<"se va a apagar el dispositivo"; }    // destructor
        void encender( ){ if (!__encendido)__encendido=true; else
             _encendido=false;}
        bool getEncendido( )  { return _encendido; }
};
class Televisor : public DispElect{
        int _canal;
        int _brillo;
public:
        Televisor( ){_canal=2; _brillo=15;}     // constructor por defecto
        Televisor(int encendido, int canal): DispElect( encendido ){
             _canal= canal;
             _brillo=15;}
        ~Televisor( ){ cout<<"¡gracias por verme!"; }//  destructor
        void fijarcanal(int canal){_canal=canal;}
        void leercanal( ){ return _canal;}
        void fijarbrillo(int brillo){ _brillo=brillo;}
        Int leerbrillo( ){return _brillo;}
};
```

Constructor parametrizado
Notar que si no recibimos todos los parámetros necesarios en el constructor los demás deben ser inicializados tal como se hace en el constructor por defecto.

```
void main( ) {
Televisor* t=newTelevisor(true,12);
        t->encender( );
        t->fijarcanal(15);
        t->encender( );
        delete t;
}
```

Acá se observa la herencia ya que encender es de clase base

Accesibilidad en Herencia

La determina el modificador declarado en la Clase.

Modificador	En la Clase Base	En la Clase Derivada	En el Objeto
private	Visible	Oculto	Oculto
protected	Visible	Visible	Oculto
public	Visible	Visible	Visible

Protected es público para la clase que hereda y privado para el objeto.

Modificadores de acceso en la Herencia

Modificador en clase base	Modificador en herencia	Modificador resultante en derivada
private protected public	 public	Sin acceso protected public
private protected public	 protected	Sin acceso protected protected
private protected public	 private	Sin acceso private private

Modificador de herencia public


```
class A{
private:
   int a;
protected:
   int d;
public:
   void metodoa();
};
```

```
class B: public A{
private:
 int j;
public:
  void metodob( ){
   a=0; // error, no es visible por ser privado en base
   d=0; // correcto por ser protected en base privado
   J=0; // correcto por ser visible en mi clase}
};
```

Si se hereda public el resultante en derivada es que todos los métodos o atributos públicos de la base serán públicos en la derivada.

Todos los métodos o atributos protected de la base serán protected en la derivada.

Los privados no serán accesibles en las instancias de objetos de derivada lo que es public será público y lo que es protected o private será privado.

```
void main (){
B* obj=new B();
obj->a=0;  // error, no es visible por ser privado
obj->d=0;  // error, no es visible por ser privado   (protected es privado para todos menos el que  hereda)
obj->j=0;   // error, no es visible por ser privado
…
```

Modificador de herencia public

Deseamos heredar las propiedades y métodos públicos como public y protegidos como protected

```
class A{
private:
  int a;
protected:
  int d;
  void metodod( );
public:
  void metodoa( );
};
```

```
class B: public A{
private:
 int j;
public:
 void metodob( ){
   a=0; // error, no es visible por ser privado en base
   d=0; // correcto por ser protected en base privado
   J=0; // correcto por ser visible en mi clase
  }
};
```

```
void main (){
B* obj=new B();
obj->metodoa( );  //correcto es público
obj->metodob( );  //correcto es público
obj->metodod( );  // error, no es visible por ser protected en base
  ...                 // se convierte en private para el objeto
}
```

Modificador de herencia protected

Deseamos heredar las propiedades y métodos públicos y protegidos como protected

```
class A{
private:
  int a;
protected:
  int d;
  void metodod( );
public:
  void metodoa( );
};
```

```
class B: protected A{
private:
 int j;
public:
   void metodob( ){
     a=0; // error, no es visible por ser privado en base
     d=0; // correcto por ser protected en base privado
     J=0; // correcto por ser visible en mi clase
   }
};
```

```
void main (){
B* obj=new B();
obj->metodoa( );  // error, no es visible por ser protected
obj->metodob( );  // correcto es público
obj->metodod( );  // error, no es visible por ser protected en base;
    …             //se convierte en private para el objeto
}
```

Modificadores de acceso en objetos con Herencia

Modificador en clase base	Modificador en herencia	Modificador resultante en derivada	Acceso en objeto de derivada
private	public	Sin acceso	Sin acceso
protected		protected	Sin acceso
public		public	visible
private	protected	Sin acceso	Sin acceso
protected		protected	Sin acceso
public		protected	Sin acceso
private	private	Sin acceso	Sin acceso
protected		Sin acceso	Sin acceso
public		Sin acceso	Sin acceso

Fin