UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño y programación de videojuegos

UNLVIRTUAL

Introducción a la programación

Unidad 8
Herencia

INTRODUCCIÓN

Es normal en los juegos ver distintos objetos que tienen el mismo comportamiento, más bien, la misma estructura. Y aunque estos no sean del mismo tipo, se les puede distinguir una clara similitud. Por ejemplo, los Koopa, Goomba, Hammer y el resto de los personajes del Mario Bros[1]. Si lo pensamos, todos saltan, todos matan, todos van hacia Mario, luego cada personaje tiene su particularidad, las Tortugas se reducen la caparazón, los hongos mueren de un salto, los Hammer disparan, etc. Esta misma situación la podemos encontrar en prácticamente todos los juegos. Ahora bien, conociendo lo que conocemos de objetos, si quisiéramos diagramar algunas de estas clases nos encontraríamos con todos esos comportamientos repetidos en casa clase.

Koopa
saltar moverse esconderse

Goomba
saltar moverse

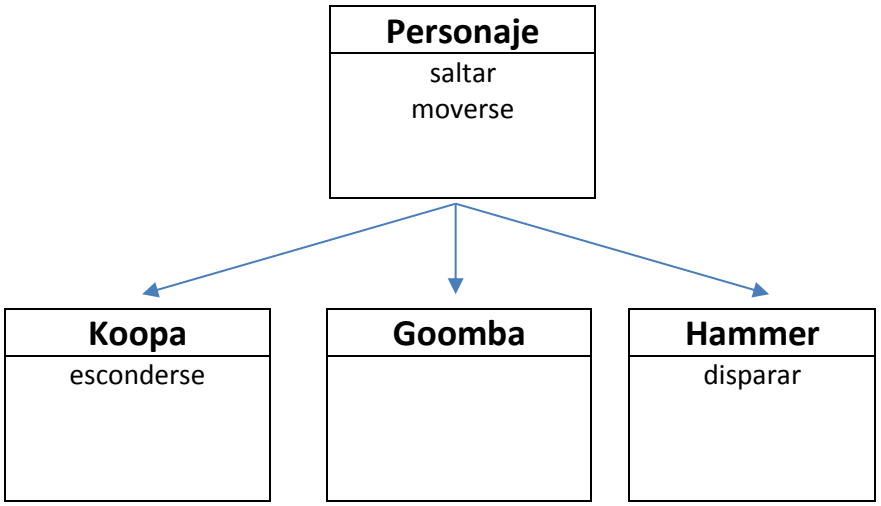
Hammer
saltar moverse disparar

Aquí los métodos que se repiten en cada clase son de idéntica implementación, es decir, si pudiéramos ver cómo están hechos estos métodos en todas las clases, el código sería exactamente el mismo. Es fácil imaginar que manejando objetos, es posible evitar duplicar código y aprovechar lo que ya está hecho. Bien, la Herencia nos permite hacer exactamente eso y es una característica propia de la Programación Orientada a Objetos.

Podríamos tener una clase que sea general a todas, le podríamos poner Personaje. Esta clase es como cualquier otra clase, con las mismas características. Vamos a darle a esta clase los métodos **saltar** y **moverse**. Así:

Personaje
saltar moverse

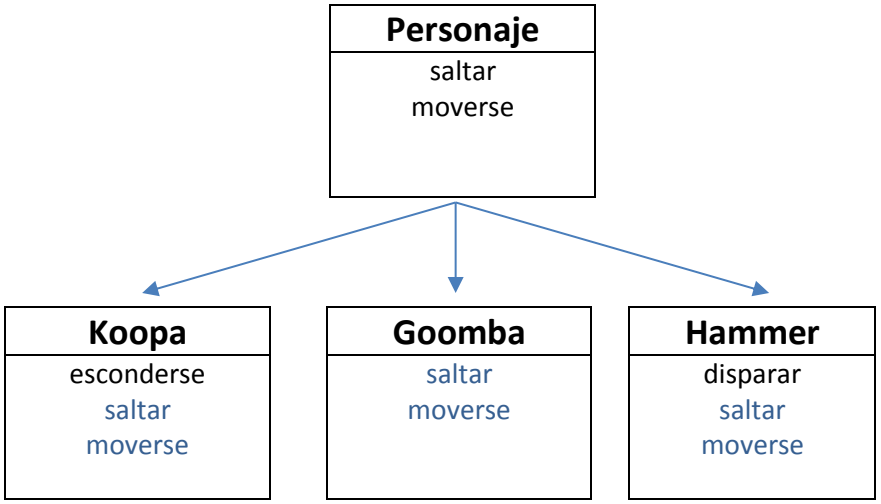
Y ahora vamos a decir que las clases Koopa, Goomba y Hammer se implementen con lo que hay en la clase Personaje:



Es decir, las clases Koopa, Goomba y Hammer ahora **heredan** de la clase Personaje. Al heredar de la clase Personaje, herendan todos los atributos y métodos que ésta le permita ver. En esta caso Personaje permite heredar a sus clases hijas (Koopa, Goomba

[1] Gameplay Mario Bros: https://www.youtube.com/watch?v=zD2CLUFGPqI

y Hammer). Los métodos **saltar** y **moverse**. Quedando el siguiente esquema:



Los métodos saltar y moverse se pueden usar y ver desde cualquiera de las clases hijas, como si hubiesen sido declarados en la misma clase.

HERENCIA EN C++

El concepto que introdujimos antes se denomina herencia, en C++ una herencia de un objeto a otro se realiza de la siguiente manera. Suponiendo la clase Vehiculo y la clase Auto:

```
class Vehiculo {

    public:
    int ruedas;
    bool acelerar(){};
    Vehiculo(){
        ruedas = 4;
    };
};
```

```
class Auto {

public:
    int puertas;
    bool retroceder(){};
    Auto(){ puertas = 3;};
};
```

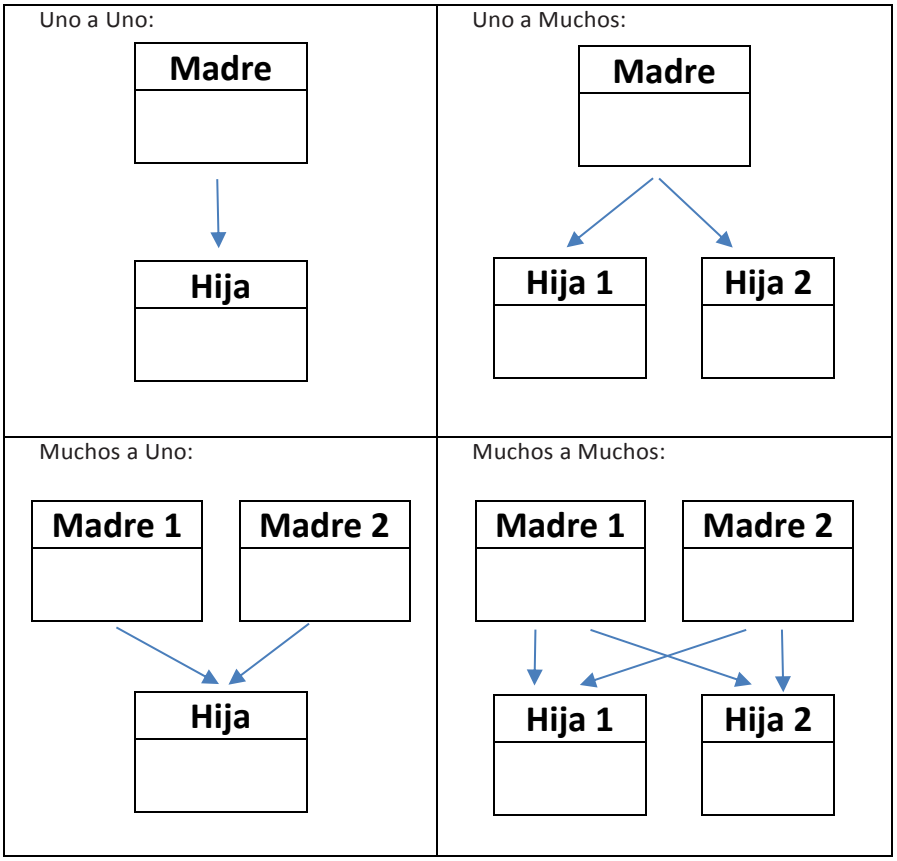
Para que Auto herede de Vehiculo, luego del nombre de la clase hija (Auto) se coloca el operador dos puntos (:) un modificador de acceso y el nombre de la clase madre.
Por ejemplo:

```
class Auto : public Vehiculo{

public:
    int puertas;
    bool retroceder(){};
    Auto(){ puertas = 3;};
};
```

En herencia la clase que hereda (Vehiculo) se denomina clase madre o padre y las clases heredadas (Auto) se denominan clases hijas.

Las herencias pueden ser: Uno a Uno, Uno a muchos, Muchos a Uno o Muchos a Muchos. Veamos cómo serían los casos.



Cuando una clase recibe herencia de varias, todas las herencias van separadas por coma (,), ejemplo:

```
class Hija : public Madre1, public Madre2, … {}
```

MODIFICADORES DE ACCESO

Para realizar la herencia utilizamos nuevamente modificadores de acceso, recordamos que los modificadores de acceso en C++ son Private, Public y Protected. Este último lo dejamos sin mayor explicación hasta ahora.

Aquellos atributos y/o métodos bajo el modificador protected serán visibles para el interior de la clase y sus hijas, pero invisible para el exterior de la clase. Es decir, es pública para adentro y sus hijas y privada para afuera.

Los decoradores de herencia también pueden ser Private, Public y Protected. La combinación entre los decoradores de la clase y los de la herencia generan el siguiente cuadro resumido:

Modificador en clase base (sobre los métodos y atributos)	**Modificador utilizado en la herencia**	**Modificador resultante en la clase derivada**
public	public	public
protected		protected
private		*no se hereda*
public	protected	protected
protected		protected
private		*no se hereda*
public	private	private
protected		private
Private		*no se hereda*

La primer columna del cuadro muestra los modificadores de las clases (métodos y

atributos), los de la columna del medio los modificadores de la herencia y la tercer columna, como quedan los métodos y atributos en las clases hijas.

En resumen se puede decir. Si la herencia es Public, los modificadores de la clase madre a las hijas se heredan sin mayores modificaciones. Si la herencia es Protected, todos los atributos pasan a ser Protected de la clase madre a la hija. Si la herencia es Private, todos los atributos pasan a ser Private de la clase madre a la hija. Aquellos atributos y/o métodos private en una clase madre, directamente no se heredan.

Veamos nuevamente con el ejemplo, vamos a declarar métodos y atributos con cada modificador.

```
class Vehiculo {

private:
    int velocidad;

protected:
    int aumentarVelocidad(){
        if (velocidad<100){velocidad++;}}

public:

    bool verVelocidad(){cout<<"velocidad:"<<velocidad<<endl;};
    Vehiculo(){velocidad = 0;};
};
```

Aquí el atributo *velocidad* es privado, el método *aumentarVelocidad* es protegido y modifica *velocidad*. Finalmente *verVelocidad* y el Construtor son públicos.

En la clase hija Auto heredamos Vehiculo como público.

```
class Auto : public Vehiculo{

public:
    void avanzar(){
        aumentarVelocidad();
    };
    Auto(){};
};
```

En Auto tenemos el método *avanzar* como público y también el constructor. Este método llama al método *aumentarVelocidad*. Al ser la herencia pública todo se hereda tal cual está en la clase madre. Entonces tenemos que, el atributo velocidad no se hereda, *aumentarVelocidad* se puede acceder desde el interior de la clase pero no desde afuera (cómo si fuera privado). *verVelocidad* y el constructor son públicos.
Luego, desde el main podemos tener algo así:

```
int main (int argc, char *argv[]) {

    Auto A1;
    A1.avanzar();
    A1.verVelocidad();

    return 0;
}
```

Desde el main no podremos acceder a *aumentarVelocidad*, pero sí a verVelocidad.

CONSTRUCTORES HEREDADOS

Cuando las clases madres tienen constructores con parámetros, debemos asignarle esos valores al momento de declaración de los constructores hijos, similar a cómo hacemos con las clases incluidas.

Por ejemplo, de la siguiente clase madre:

```
class Vehiculo {

public:
    int ruedas;
    Vehiculo(int r){
        ruedas = r;
    };
};
```

La clase Vehiculo tiene un constructor que recibe un entero como parámetro. Cuando lo heredamos desde la clase hija le pasamos el valor desde el constructor de la clase hija de la siguiente manera:

```
class Auto : public Vehiculo{

public:
    Auto(int rd) : Vehiculo(rd){

    };
};
```

Luego en el main podemos instanciar la clase:

```
int main (int argc, char *argv[]) {

    Auto A1(10);
    return 0;
}
```