Programación Orientada a Objetos

U3: Herencia

LI. Jaime Jesús Delgado Meraz
j2deme+poo@gmail.com

Instituto Tecnológico de Ciudad Valles

Enero – Junio 2013

Conceptos Básicos

- Una clase representa un conjunto de objetos que comparten la misma estructura y comportamientos.
- Esta clase define la estructura de dichos objetos mediante la declaración de variables, contenidas en cada instancia de la misma, así como determina el comportamiento, mediante la definición de métodos.
- La idea central de la programación orientada a objetos, es permitir a las clases expresar similitudes entre objetos que comparten algo, pero no todo, de su estructura y comportamiento.
- Tales similitudes pueden expresarse usando **herencia** y **polimorfismo**.

- La herencia es usada principalmente cuando existe una clase que puede adaptarse para resolver el problema dado, con solo hacer pequeños cambios o adiciones, en contraposición a diseñar grupos de clases y subclases desde cero.
- Una relación de herencia entre clases puede darse de 2 formas:

  1. Una subclase puede añadir estructura y comportamiento a lo que hereda.
  2. Una subclase puede reemplazar o modificar el comportamiento heredado.

- Sin embargo una subclase no puede modificar la estructura heredada.

Herencia

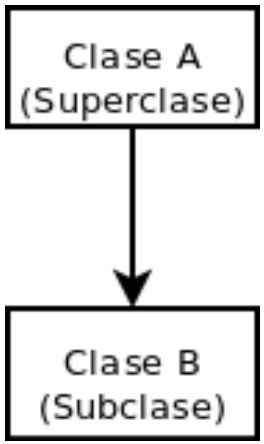
Definición

Herencia se refiere al hecho de que una clase puede heredar toda o parte de la estructura (atributos) y comportamiento (métodos) de otra clase.

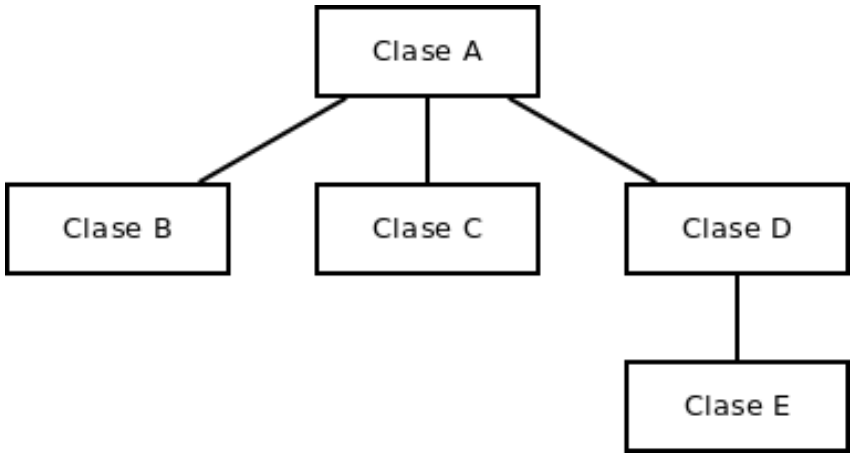
- A la clase que "da" la herencia, se le denomina **superclase** (base).
- En el sentido inverso, la clase que "recibe" la herencia, se denomina **subclase** (derivada) de la clase de la cual hereda.

- Múltiples clases pueden ser declaradas como subclases de la misma superclase.
- Dichas subclases, que pueden ser referidas como “clases hermanas”, comparten algunas estructuras y comportamientos, los cuales estan presentes en la superclase.
- La herencia puede extenderse por varias “generaciones” de clases.
- Finalmente, es conjunto de clases forma una **jerarquía** de clases.

Tipos de Herencia



(a) Simple



(b) Múltiple

Sintaxis para definir herencia

```
1    class Subclase : public ClaseExistente {
2        //Estructura y comportamiento
3    };
```

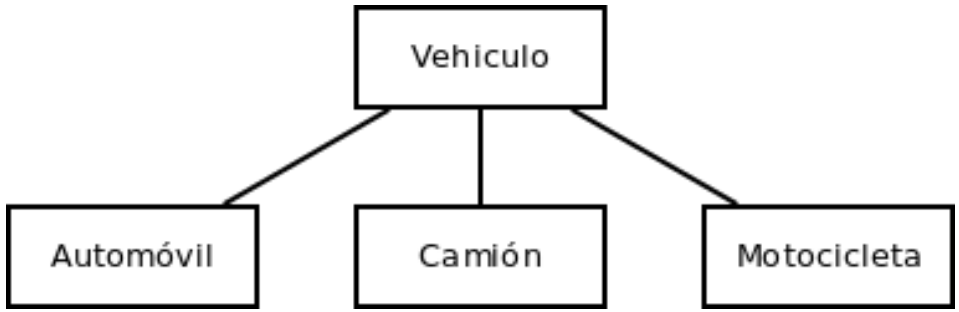
- Extender de clases existentes es una manera fácil de construir sobre trabajo previo.
- En la práctica, existe un gran número de clases estándar que han sido escritas específicamente para ser usadas como base para hacer subclases.

Modificadores de acceso

- Al trabajar con herencia, los modificadores de acceso como `public` y `private` se utilizan para control el acceso a los miembros de una clase.
- Además de los modificadores mencionados existe también el modificador `protected`.
- Cuando `protected` es aplicado como modificador de acceso a un método o variable de una clase, ese miembro puede ser usado en subclases (directas o indirectas) de dicha clase, pero no así con clases que no lo extiendan.
- Sin embargo existe una excepción, un miembro con el modificador `protected`, también puede ser accesado por cualquier clase en el mismo paquete de la clase que lo contiene.

Ejemplo: Vehículos

- Suponga un programa que debe manejar vehículos, incluyendo automóviles, camiones y motocicletas.
- El programa podría utilizar una clase `Vehiculo` para representar todos los tipos de vehículos.
- Puesto que automóviles, camiones y motocicletas, son tipos de vehículos, estos podrían representarse como una subclase de `Vehiculo`.



- La clase `Vehiculo` podría incluir variables como `numeroRegistro` y `propietario`, y métodos tales como `cambiarPropietario()`, puesto que estas son variables y métodos comunes a todos los vehículos.
- Las 3 subclases de `Vehiculo`: `Automovil`, `Camion` y `Motocicleta`, podrían usarse para contener métodos y variables específicas a cada tipo de vehículo.

Automovil	Variable de tipo entero `numeroPuertas`.
Camion	Variable de tipo entero `numeroEjes`.
Motocicleta	Variable de tipo booleana `tieneSidecar`.

```
1  class Vehiculo {
2  public:
3      int numeroRegistro;
4      Persona propietario; //Asumiendo que Persona ha sido definida
5      void cambiarPropietario(Persona nuevoPropietario){. . .}
6  };
7  class Automovil : public Vehiculo {
8  public:
9      int numeroPuertas;
10 };
11 class Camion : public Vehiculo {
12 public:
13     int numeroEjes;
14 };
15 class Motocicleta : public Vehiculo {
16 public:
17     bool tieneSidecar;
18 }
```

- Supongamos que `miAuto` es una variable de tipo `Automovil` que ha sido inicializada usando:

```
1    Automovil miAuto;
```

- Dada esta declaración, un programa podría referir:

```
1    miAuto.numeroPuertas;
```

  Puesto que `numeroPuertas` es una variable instanciada de la clase `Automovil`.

- Ademas, puesto que `Automovil` extiende `Vehiculo`, un automovil tiene toda la estructura y comportamiento de un vehículo. Esto significa que las variables y método:

```
1    miAuto.numeroRegistro;
2    miAuto.propietario;
3    miAuto.cambiarPropietario();
```

  También existen para `miAuto` por ser heredadas de la clase `Vehiculo`.

- En el mundo real automóviles, camiones y motocicletas son vehículos. Lo mismo ocurre con un programa, es decir, un objeto de tipo `Automovil`, `Camion` o `Motocicleta` es también, automáticamente, un objeto de tipo `Vehiculo`.
- Esto nos demuestra un hecho importante:

Una variable que puede contener una referencia a un objeto de la clase `A`, puede contener una referencia a un objeto perteneciente a cualquier subclase de `A`.

- El efecto práctico del enunciado anterior, es que un objeto de tipo `Automovil`, puede ser asignado a una variable de tipo `Vehiculo`.

```
1        Vehiculo miVehiculo = miAuto;
2        Vehiculo miOtroVehiculo = new Automovil();
```

Otro ejemplo

- La subsunción[1] (*casting* en inglés) es una propiedad que todos los objetos que pertenecen a una jerarquía de clases deben satisfacer. Un objeto de la clase base puede sustituirse por cualquier objeto que deriva de ella (directa o indirectamente).
- Por ejemplo: Todos los mamíferos son animales (derivan de ellos), y todos los gatos son mamíferos. Por lo tanto, por la propiedad de subsunción podemos "tratar" cualquier mamífero como un animal, y cualquier gato como mamífero.
- Esto implica el uso de abstracción, puesto que cuando estamos "tratando" un mamífero como un animal, la única información que necesitamos saber es que vive, crece, etcetera, pero nada relacionado con mamíferos.

[1] Inclusión de un objeto o de un concepto en la extensión de otro.

- Esta propiedad se aplica en C++, cuando se utilizan punteros o referencias a objetos que pertenecen a una jerarquía de clases. En otras palabras, un puntero de clase `Animal`, puede apuntar a un objeto de clase `Animal`, `Mamifero` o `Gato`.

```
1  class Animal {
2  public:
3     string tipo;
4     bool estaVivo;
5     int numeroCrias;
6  };
7  class Mamifero : public Animal{
8  public:
9      void amamantar();
10 };
11 class Gato : public Mamifero{
12 public:
13     bool comePescado;
14 };
15
```

```
16 int main() {
17   Animal* a1 = new Animal;
18   Animal* a2 = new Mamifero;
19   Animal* a3 = new Gato;
20   Mamifero* m  = new Gato;
21
22   a2->estaVivo = true;       // Correcto
23   a2->tipo = "Herbivoro";    // Correcto
24   m->amamantar();            // Correcto
25
26   a2->amamantar();           // Incorrecto
27   a3->comePescado() = true; // Incorrecto
28
29   Gato* g1 = (Gato*)a3;       // Downcast correcto pero no recomendado
30   Gato* g2 = dynamic_cast<Gato*>(a3); //Downcast recomendado
31   g1->comePescado = false;    // Correcto (aunque raro)
32
33   Gato* g3 = new Gato;
34   Motocicleta* moto = dynamic_cast<Motocicleta*>(g3);
35 }
```

Conclusión

- La herencia, juega un papel fundamental en la POO, ya que permite el paso entre clases de ciertas características, parametros y métodos.
- Permite la reutilización de código y la combinación de clases, eliminando la cantidad de código duplicado.
- Mejora la organización del código, resultando en unidades de compilación más pequeñas y simples de manejar.