Unidad VI

HERENCIA Y POLIMORFISMO

Herencia

- Se expresa como una relación de descendencia (es-un)
- Consiste en definir una nueva clase a partir de una existente
- La nueva clase se denomina subclase o clase derivada
- La clase existente se denomina superclase o clase base
- Es la propiedad que permite a los ejemplares de una subclase acceder a los miembros de la superclase

Herencia

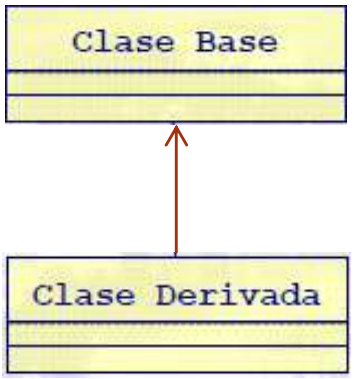
- Las subclases heredan tanto los atributos como los métodos de la superclase
- La herencia es transitiva
- Una subclase tiene todas las propiedades de la superclase y otras más (extensión)
- Una subclase constituye una especialización de la superclase (reducción)
- Un método de superclase es anulado por un método con el mismo nombre definido en la subclase

Herencia

- Un constructor de subclase siempre invoca primero al constructor de su superclase
- Un destructor de subclase se ejecuta antes que el destructor de su superclase
- No se necesita reescribir el código del comportamiento heredado
- La reutilización de software sólo exige conocer la funcionalidad del componente y su interfaz
- Los métodos heredados se ejecutan más lentamente que el código especializado

Concepto de la herencia

- La clase que hereda se denomina subclase o clase derivada.
- La clase de la cual se hereda se denomina superclase o clase base.
- Todo objeto de una subclase es un objeto de la superclase de la cual deriva.
- Las subclases pueden redefinir los métodos y atributos de la clase padre y añadir otros nuevos.



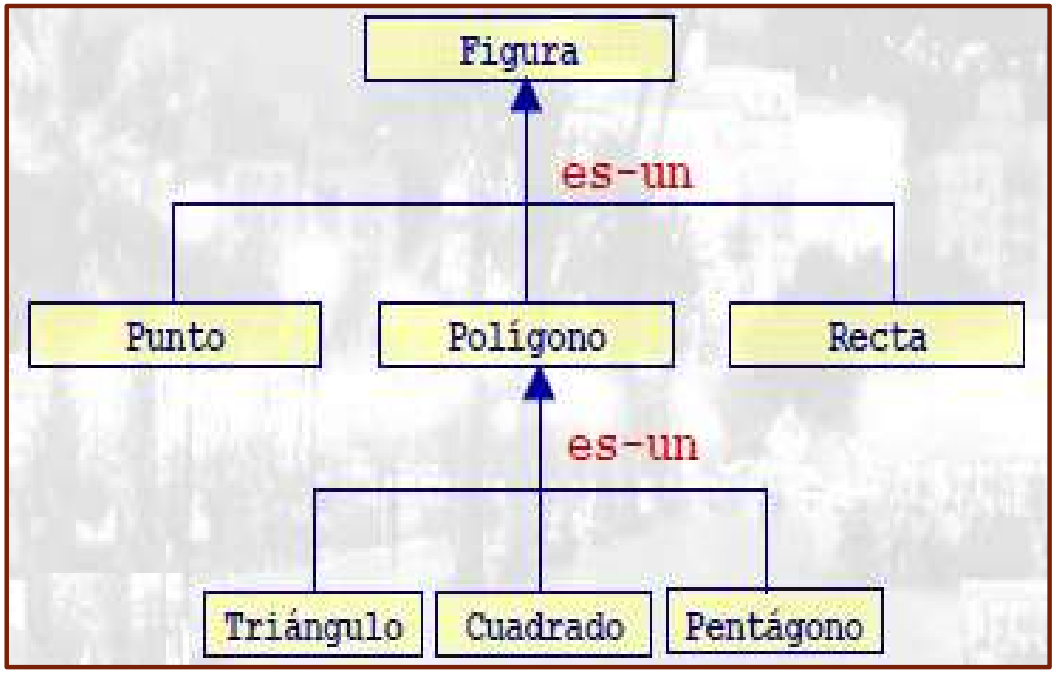
Jerarquía y herencia

- Herencia: (por ejemplo, la clase D recibe herencia de la clase C) Es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D.

- La Jerarquía es una propiedad que permite la ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son:
  - Jerarquía “es-un”: generalización/especialización
  - jerarquía “parte-de”: agregación

- Las jerarquías de generalización/especialización se conocen como herencia. Básicamente, la herencia define una relación entre clases, en donde una clase comparte la estructura o comportamiento definido en una o más clases (herencia simple y herencia múltiple, respectivamente).

MODELO DE GENERALIZACION/ESPECIALIZACION



GENERALIZACION

ESPECIALIZACION

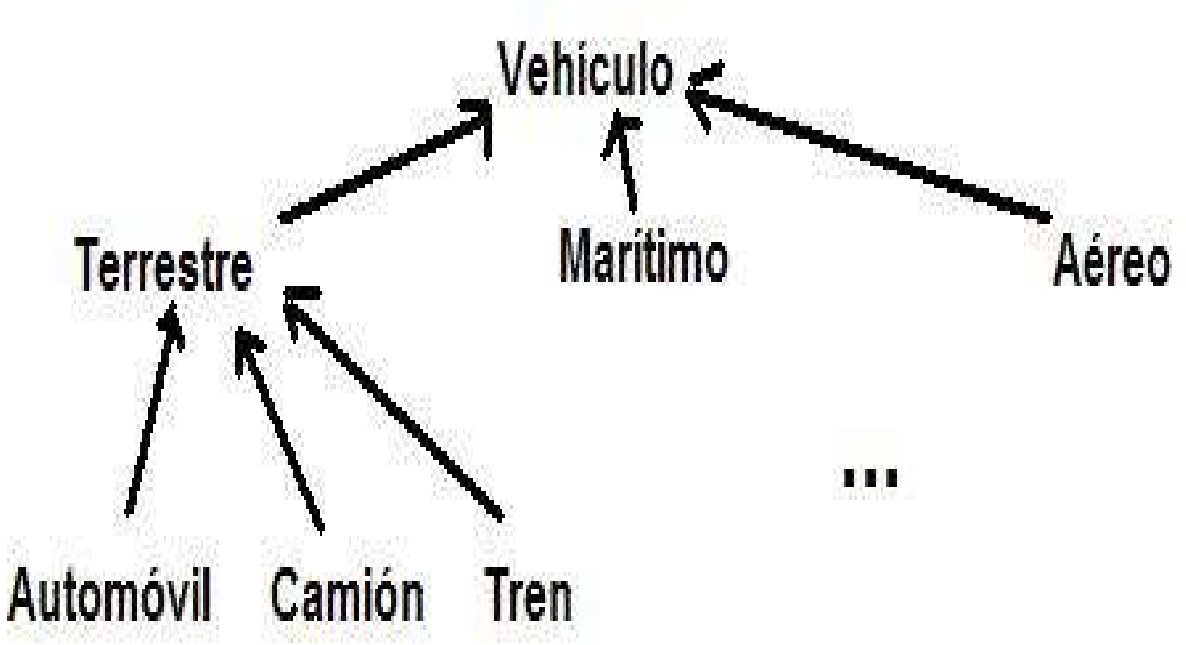
Se detectan clases con un comportamiento común.

Ejemplo: Triangulo y polígono son figuras.

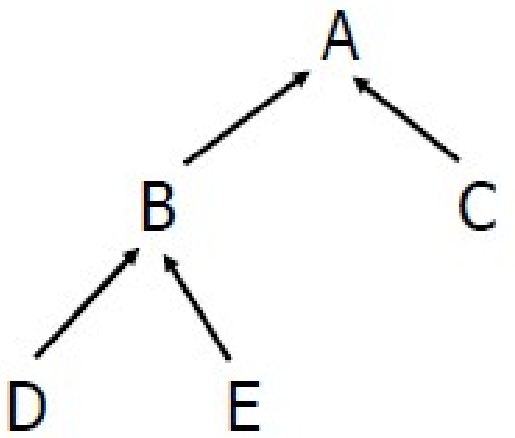
Se detecta que una clase es un caso especial de otra

Ejemplo: Triangulo es un tipo de Polígono.

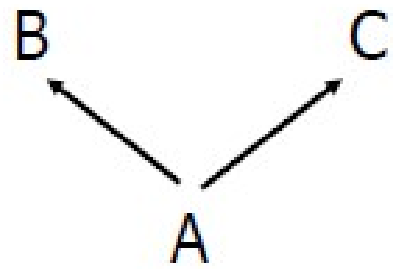
Las relaciones de herencia forman una estructura de árbol (jerarquía)



Tipos de herencia



- Herencia simple Una clase puede heredar de una única clase.



- Herencia múltiple Una clase puede heredar de varias clases.

Herencia Simple
class ClaseDerivada: public o private ClaseBase

Herencia múltiple
class CuentaEmpresarial: public Cuenta, public Empresa

- Un nombre redefinido oculta el nombre heredado.
- Hay algunos elementos de la clase base que ***no pueden ser heredados***:
  - Constructores
  - Destructores
  - Funciones y datos estáticos de la clase
  - Operador de asignación (=) sobrecargado

Constructores de clases derivadas

- Debe llamar al constructor de la clase base.
- Se debe especificar un ***inicializador base***.
- Se puede omitir si la clase base cuenta con un constructor por defecto.

```
C_CuentaJoven(const char *unNombre, int
  laEdad, double unSaldo=0.0, double
  unInteres=0.0): C_Cuenta(unNombre,
  unSaldo, unInteres)
```

Polimorfismo
Funciones Virtuales

- Son funciones distintas con el mismo nombre, declaradas `virtual` en la clase base (ligadura dinámica).
- Funciones convencionales se invocan de acuerdo al tipo del objeto (en tiempo de compilación).
- Con funciones virtuales se resuelve en tiempo de ejecución el problema de la asignación.

Funciones virtuales

```
class A {
   public:
        virtual void mostrar();
}

class B: public A {
   public:
        void mostrar();
}
```

```
A objA;
B objB;

A* ptrA1;
A* ptrA2;

ptrA1 = &objA;
ptrA2 = &objB;

ptrA2->mostrar();
```

Funciones virtuales puras

- La función virtual de la clase base debe declararse a pesar de no ser utilizada.
- En este caso no es necesario definirla.
- Se declara como función virtual pura:

```
virtual funcion1() const = 0;
```

- No se pueden definir objetos de esa clase.
- Se pueden definir punteros a esa clase.

Clases abstractas

- Contienen una o más funciones virtuales puras.
- Si una clase derivada no define una función virtual pura, la hereda como pura y por lo tanto también es abstracta.
- Una clase que define todas las funciones virtuales es una clase concreta.

COMPOSICIÓN DE CLASES

Composición de clases

- Se manifiesta como una relación de pertenencia (tiene-un)
- Consiste en declarar objetos de una clase A como atributos de otra clase B
- El constructor de la clase que contiene objetos de otras clases llamará a los correspondientes constructores
- Un constructor por defecto de la clase llamará implícitamente a los constructores por defecto de los objetos declarados como atributos

Composición de clases

```
class Curso {
    public:
        Curso(int t=30);
        void Inscribir(Alumno&);
        void Listar();
        double Promedio();
        int Aprobados();
        int Reprobados();
    private:
        int N;
        char nomCur[25];
        char codCur[7];
        Alumno v[50];
};
```

Composición de clases

```
Alumno::Alumno() {
    k = 0;
    t = 0;
}
Alumno::Alumno(char *n, char *r, int m, int c) {
    nom = new char[strlen(n)+1];
    rut = new char[strlen(r)+1];
    strcpy(nom, n);
    strcpy(rut, r);
    mat = m;
    carrera = c;
    k = 0;
    t = 0;
}
```

Composición de clases

```
Curso::Curso(int t) {
    N=t;
    cin.getline(nomCur, 25);
    cin.getline(codCur, 7);
    char *x, *y;
    int z, c;
    for (int i=0; i<N; i++) {
        cout << "NOMBRE: ";
        cin.getline(x, 20);
        cout << "RUT   : "; cin >> y;
        cout << "MAT   : "; cin >> z;
        cout << "Carr  : "; cin >> c;
        Alumno a(x, y, z, c);
        v[i] = a;
    }
}
```

Composición de clases

```
void Curso::Listar() {
    for (int i=0; i<N; i++)
        v[i].Listar();
}
```

```
int main() {
    Curso C;
    C.Listar();
    return 0;
}
```