Facultad: Ingeniería
Escuela: Computación
Asignatura: Programación Orientada a Objetos

Tema: Herencia en C#. Parte I.

Materiales y Equipo

- Computadora con Visual Studio /Visual C#.
- Guía Número 8.
- Dispositivo de almacenamiento o almacenamiento en la nube.

Introducción Teórica

La Herencia.

Es la capacidad de compartir atributos y métodos entre clases.
Es la propiedad que permite definir nuevas clases usando como base clases ya existentes.
La nueva clase (clase derivada) hereda los atributos y comportamiento que son específicos de la clase existente.
La herencia es una herramienta poderosa que proporciona un marco adecuado para producir software fiable, comprensible, de bajo costo, adaptable y reutilizable.

La ***herencia o relación es-un*** es la relación que existe entre dos clases, en la que una clase denominada ***derivada o subclase*** se crea a partir de otra ya existente, denominada ***clase base o superclase***.

Evidentemente, la clase base y la clase derivada tienen código y datos comunes, de modo que si se crea la clase derivada de modo independiente, se duplicaría mucho de lo que ya se ha escrito para la clase base.

C# soporta el mecanismo de derivación que permite crear clases derivadas, de modo que la nueva clase hereda todos los datos miembro que pertenecen a la clase ya existente.

La declaración de derivación de clases debe incluir el nombre de la clase base de la que se deriva y el especificador de acceso que indica el tipo de herencia (pública, privada ó protegida). La primera línea de cada declaración debe incluir la sintaxis siguiente:

```
class <nombre de clase derivada> : <nombre de clase base>
```

Tipos de Herencia.

Dependiendo del número de clases y de cómo se relacionen, la herencia puede ser:

a) Simple.
b) Múltiple.
c) De niveles múltiples.

Con independencia del tipo de herencia, una clase derivada no puede acceder a variables y funciones privadas de su clase base. Para ocultar los detalles de la clase base y de clases y funciones externas a la jerarquía de clases, una clase base utiliza normalmente elementos protegidos en lugar de elementos privados.

Herencia Simple.

Cuando sólo se tiene una clase base de la cual hereda la clase derivada, se dice que hay herencia simple (ver Figura 1.a).

Sin embargo, la herencia simple no excluye la posibilidad de que de una misma clase base se pueda derivar más de una subclase o clase derivada (Figura 1.b).

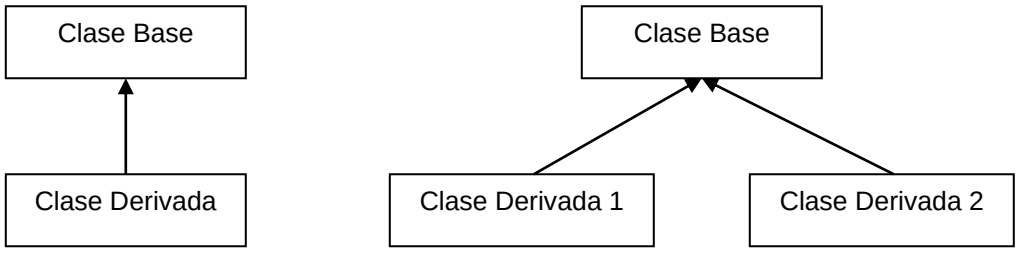


Figura 1. Herencia Simple.

Cuando se necesita representar un concepto general y a partir de éste, conceptos más específicos, resulta conveniente organizar la información usando herencia.

Esto permite compartir atributos y métodos ya definidos, evita la duplicidad y, por otra parte, proporciona mayor claridad en la representación que se haga de la información. Es decir, se logra un mejor diseño de la solución del problema.

Existen numerosos casos en los cuales se da este tipo de relación.

En la Figura 2 se presentan algunos ejemplos de herencia simple.

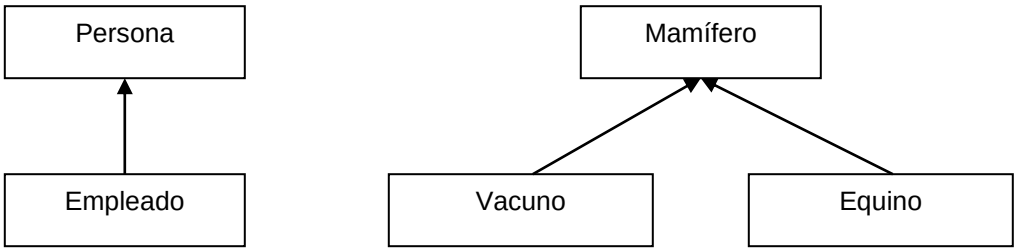


Figura 2. Ejemplos de Herencia Simple.

En la figura 2a, la clase **Persona** es la clase base y **Empleado** es la clase derivada. Un objeto de esta clase también es un objeto de la clase "Persona", por lo tanto, tendrá los atributos y métodos de ambas clases.

En la figura 2b, la clase **Mamífero** es la clase base y **Vacuno** y **Equino** son las clases derivadas. En este caso, se dice que todo "Vacuno" y todo "Equino" también son objetos de la clase "Mamífero" y en consecuencia tendrán todos los atributos y métodos que heredan de la clase base.

Los miembros protegidos de la clase base podrán ser usados por los métodos de las clases derivadas, pero no por sus clientes.

Los miembros públicos estarán disponibles para los métodos de las clases derivadas y para todos sus clientes.

```
class Base
{     private:
     // Miembros declarados en la sección privada: accesibles sólo para miembros de esta clase
      protected:
     /* Miembros declarados en la sección protegida: accesibles sólo para miembros de esta
          clase y de sus derivadas */
      public:
     // Miembros declarados en la sección pública: accesibles para todos
};
```

Para declarar una clase derivada de una clase previamente definida se utiliza la siguiente sintaxis:

```
class Base
{       // Declaración de atributos y métodos de la clase Base
};
        // Relación de herencia pública entre las clases Base y Derivada
class Derivada :Base
{       // Declaración de atributos y métodos de la clase Derivada
};
```

Si se utilizara un constructor con parámetros se debe declarar en la clase "base" y en la "derivada" sus respectivos constructores. En el constructor de la clase derivada, se especifican qué parámetros recibe de la base. Esto es gracias a la palabra reservada **Base.**
Para ello se utiliza la siguiente sintaxis:

```
public Derivada (parámetros) : base (parámetros propios de la clase Base)
{       // Cuerpo del constructor de la clase Derivada
}
```

Cuando se declara un objeto del tipo de la clase derivada se invoca al constructor de ésta. De este constructor lo primero que se ejecuta es la llamada al constructor de la clase base, y posteriormente se ejecutan sus propias instrucciones.

En cuanto a los parámetros, al invocar al constructor de la clase base se le deben proporcionar los parámetros que necesita para asignar valores a los atributos propios de la clase base y que la clase derivada hereda.

En el cuerpo de la clase derivada se harán las asignaciones correspondientes a los atributos propios de esta clase.

Herencia Múltiple (No aceptada en C#)

La herencia múltiple no es admitida en C#, sin embargo algunos lenguajes de POO sí la permiten, en el tipo de herencia múltiple se usan dos o más clases base para derivar una

clase. Es decir, la clase derivada comparte los atributos y los métodos de más de una clase (ver Figura 3).

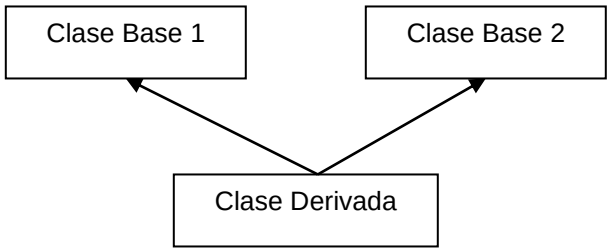


Figura 3. Herencia Múltiple.

Herencia de Niveles Múltiples.

Se presenta cuando una clase derivada se usa como base para definir otra clase derivada. Es decir, existen diferentes niveles de herencia: en el primero, la clase derivada hereda los miembros de una clase base, mientras que en el segundo, la clase derivada funciona a su vez como una clase base y de esta forma comparte con una tercera clase sus propios miembros y los que heredó (ver Figura 4). Esta relación puede extenderse a tantos niveles como lo requiera el problema que se esté resolviendo.

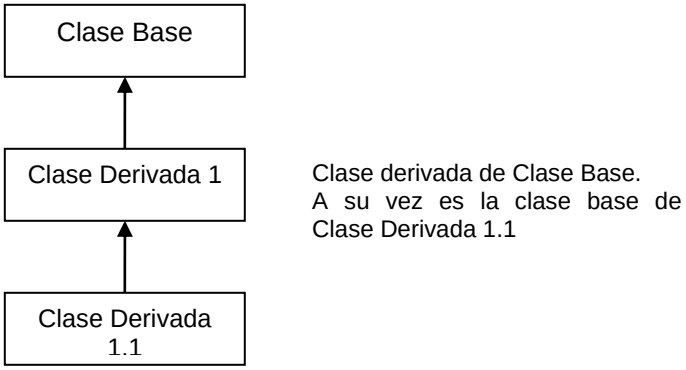
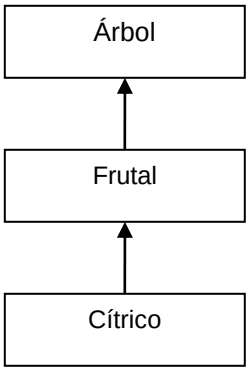


Figura 4. Herencia de niveles múltiples.

Este tipo de herencia es muy útil cuando es necesario representar, a partir de conceptos generales, conceptos más específicos. Cuantos más niveles se deriven, mas especificidad se definirá.

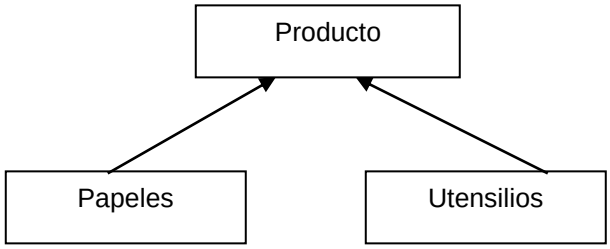
La figura 5 presenta un ejemplo de herencia de múltiples niveles. El nivel superior representa la clase más general, la clase “Árbol”. Luego, la clase “Frutal” es una clase derivada de la primera, lo cual indica que los frutales son una clase más específica de árboles. Por último, se

define la clase Cítrico como una subclase de la clase “Frutal”. Esta relación también indica que los cítricos son una variante, una clase más específica, de los árboles frutales.



Ejemplo 3:

Se muestra un ejemplo de Herencia simple en entorno consola. Se realiza la implementación de la jerarquía de clases mostrada en la siguiente figura:



La cual corresponde a una Papelería que maneja esos dos tipos de Productos, papeles y utensilios, se reutilizan elementos comunes de la clase principal y se combinan con los propios de cada clase derivada.

CLASE PRODUCTO

```
class Producto
{
    private float precio;

    protected float Precio...
    private long codigo;

    protected long Codigo...
    private string marca;

    protected string Marca...
    private string nombre;

    protected string Nombre...

    protected void almacenar()
    {
        Console.WriteLine("Ingrese el producto que almacenaremos: ");
        Console.WriteLine("\nNombre: ");
        nombre = Console.ReadLine();
        Console.WriteLine("\nMarca: ");
        marca = Console.ReadLine();
        Console.WriteLine("\nNúmero de código de barras: ");
        codigo = long.Parse(Console.ReadLine());
        Console.WriteLine("\nPrecio: ");
        precio = float.Parse(Console.ReadLine());
    }
    protected void mostrar()
    {
        Console.WriteLine("\n Hemos almacenado esto en bodega: ");
        Console.WriteLine(Nombre + " " + " de la marca " + Marca);
        Console.WriteLine(" con el código " + Codigo + " y con el precio de " + Precio);
    }


}
```

CLASE PAPELES

```
class Papeles: Producto
{
    string tipo;
    public string Tipo...
    string color;
    public string Color...
    string tamanio;
    public string Tamanio...

    public void selecpapel()
    {
        almacenar();
        Console.WriteLine("\nElijamos el papel que prefiere");
        Console.WriteLine("\nTipo: ");
        tipo = Console.ReadLine();
        Console.WriteLine("\nColor: ");
        color = Console.ReadLine();
        Console.WriteLine("\nTamaño: ");
        tamanio = Console.ReadLine();
    }

    public void mostrarPapel()
    {
        mostrar();
        Console.WriteLine("esto es un papel de tipo " + tipo +"\n con el color "+color + " de tamaño:" +tamanio);

    }
}
```

CLASE UTENSILIOS

```
class Utensilios:Producto
{
    string categoria;
    public string Categoria...
    double peso;
    public double Peso...
    string material;
    public string Material...

    public void selecutensi()
    {
        almacenar();
        Console.WriteLine("\n Qué tipo de utensilio prefiere");
        Console.WriteLine("\nCategoria: ");
        categoria = Console.ReadLine();
        Console.WriteLine("\nPeso: ");
        peso = double.Parse(Console.ReadLine());
        Console.WriteLine("\nMaterial: ");
        material = Console.ReadLine();
    }

    public void mostrarUti()
    {   mostrar();
        Console.WriteLine("esto es de la categoria " + categoria + " de " + material + "\ncon un peso " + peso);

    }
}
```

CLASE PROGRAM

```
class Program
{
    static void Main(string[] args)
    {
        int op;
        Console.WriteLine("Ingrese con qué producto desea trabajar\n");
        Console.WriteLine("1. Papel\n");
        Console.WriteLine("2. Utensilios\n");
        Console.WriteLine("**********************");

        op = int.Parse(Console.ReadLine());
        switch (op)
        {

        case 1:
        Papeles paper = new Papeles();
        paper.selecpapel();
        paper.mostrarPapel();
        break;

        case 2:
        Utensilios util = new Utensilios();
        util.selecutensi();
        util.mostrarUti();
        break;

       default:
       Console.WriteLine("No es una opción válida");
       break;
        }
        Console.ReadKey();
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Secretaria secre1 = new Secretaria(100,"Gerardo","Matilde",23,140.0,3,400.00,"Financiera");
        secre1.mostrar();
        Console.ReadKey();
    }
}
```

Desarrollo de habilidades

Ejercicio 1:

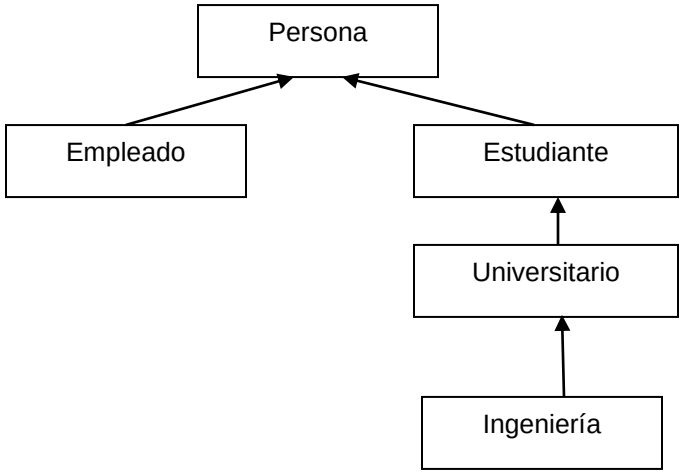
Basados en el ejercicio 3 en consola, se le pide que lo modifique de forma que funcione en entorno gráfico, respete los atributos asociados a cada clase y su estructura de herencia.

Ejercicio 2:

Programe la funcionalidad del resto de botones para el formulario de registro de estudiante y de docente y que puedan llenarse los datagridview que se encuentran en la parte inferior de los formularios.

Ejercicio 3:

Construya una solución para la jerarquía de clases mostrada en la siguiente figura:



EN ENTORNO GRÁFICO desarrolle:

Para la clase Estudiante, considere los atributos: número de carnet, nivel de estudios.

Para la clase Universitario considerar los atributos: nombre de la universidad, carrera, materias inscritas, notas, CUM.

Para la clase Ingeniería considerar los atributos: nombre del proyecto, total de horas (duración de la pasantía), número de horas completadas.

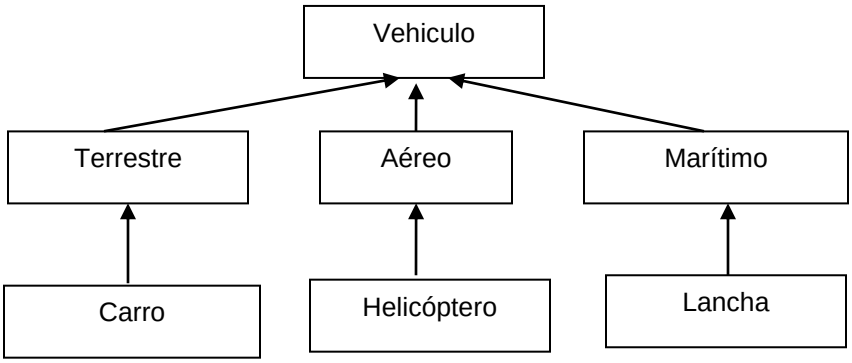
La solución se debe manejar a través de un menú (botones, opciones, combobox u otros) que permita realizar las siguientes acciones:

a) Crear los objetos de tipo Ingeniería, solicitando los datos al usuario.
b) Verificar que universidad proporciona la mayoría de estudiantes de ingeniería
c) Visualizar el promedio de notas de los estudiantes
d) Salir de la aplicación.

Investigación Complementaria

Ejercicio 1:

Considere la siguiente jerarquía de herencias del transporte disponible que tiene una Compañía turística:



Definir las clases. Decidir que atributos y métodos incluir en cada clase de tal manera que su programa pueda realizar como mínimo las siguientes acciones:

a) Crear objetos de cualquier tipo de las clases finales (carro, helicóptero, crucero), solicitando los datos al usuario.
b) Verificar qué opciones de transporte terrestre se tienen para la realización de un viaje de fin de semana.
c) Reservar un vehículo y si estuviera ocupado, enviarme un mensaje de que no está disponible.
d) Salir de la aplicación.