Presentación

Asignatura
Programación I

Tema
Herencia

Docente
Joerlyn Mariano Morfe Ureña

Sustentado por:
Cris Leidy Rosario De los santos
24-1046

Fecha
03/03/2025

Actividad:

- **Realice 5 ejemplos de código de herencia, debe tomar diferentes casos de usos.**
- **Investigar sobre herencias, realizar un trabajo de investigación sobre las formas de herencias que existen.**

La Programación Orientada a Objetos (POO) es un paradigma de desarrollo de software que permite modelar entidades del mundo real a través de clases y objetos. Entre sus principales características se encuentran la encapsulación, la abstracción, el polimorfismo y la herencia.

La herencia es un proceso esencial en la Programación Ordenada que posibilita que una clase (conocida como clase derivada o subclase) adopte atributos y procedimientos de otra clase (conocida como clase base o superclase). Esto permite reutilizar el código, minimizar las redundancias y estructurar los programas de forma más eficaz y modular. La herencia permite establecer vínculos jerárquicos entre las clases, facilitando la modelación de estructuras organizativas, taxonomías y otros sistemas donde se presentan niveles de especialización.

La correcta utilización de la herencia no solo mejora el desarrollo del software, sino que también incrementa la capacidad de mantenimiento y expansión de los programas. No obstante, un diseño incorrecto puede generar problemas como la rigidez en la arquitectura del sistema o la complejidad para efectuar modificaciones sin impactar en diversas secciones del código. Por esta razón, resulta crucial entender las distintas modalidades de herencia y sus usos apropiados para cada situación.

Formas de herencia

Herencia Simple: Es el tipo más básico de herencia, donde una clase hija hereda directamente de una única clase base. Permite reutilizar métodos y atributos sin necesidad de redefinirlos.

Ejemplo:

```
class Vehiculo {
    public string Marca { get; set; }
    public void MostrarInfo() {
        Console.WriteLine($"Marca: {Marca}");
    }
}

class Coche : Vehiculo {
    public int Puertas { get; set; }
}
```

Aplicación:

- Modelado de entidades que comparten características comunes.
- Reducción de código redundante.

Herencia Multinivel: Ocurre cuando una clase hereda de otra que a su vez ha heredado de otra clase. Se forma una cadena de herencia en varios niveles.

Ejemplo:

```
class Persona {
    public string Nombre { get; set; }
}

class Empleado : Persona {
    public double Salario { get; set; }
}

class Gerente : Empleado {
    public string Departamento { get; set; }
}
```

Aplicación:

- Modelos jerárquicos como organigramas empresariales.
- Sistemas donde las entidades tienen niveles de especialización progresiva.

Herencia Jerárquica: Se da cuando múltiples clases heredan de una misma clase base.

Ejemplo:

```
class Animal {
    public void HacerSonido() {
        Console.WriteLine("El animal hace un sonido");
    }
}

class Perro : Animal {
    public void Ladrar() {
        Console.WriteLine("Guau guau");
    }
}

class Gato : Animal {
    public void Maullar() {
        Console.WriteLine("Miau");
    }
}
```

Aplicación:

- Modelado de taxonomía biológica.
- Sistemas en los que una entidad base tiene múltiples especializaciones.

Herencia Múltiple (A través de Interfaces en C#): C# no permite la herencia múltiple directa, pero se puede simular mediante la implementación de múltiples interfaces.

Ejemplo:

```
interface IEncendible {
    void Encender();
}

interface IApagable {
    void Apagar();
}

class Dispositivo : IEncendible, IApagable {
    public void Encender() {
        Console.WriteLine("El dispositivo está encendido");
    }
    public void Apagar() {
        Console.WriteLine("El dispositivo está apagado");
    }
}
```

Aplicación:

- Sistemas que requieren múltiples comportamientos sin una relación jerárquica estricta.
- Implementación de contratos en desarrollo de software.

Herencia Híbrida: Es una combinación de varios tipos de herencia, aunque en C# se limita debido a la falta de soporte para herencia múltiple directa.

Ejemplo: Se podría combinar la herencia jerárquica con interfaces para lograr una estructura híbrida.

```
class Dispositivo {
    public string Nombre { get; set; }
}

interface IEncendible {
    void Encender();
}

class SmartPhone : Dispositivo, IEncendible {
    public void Encender() {
        Console.WriteLine("El smartphone está encendido");
    }
}
```

Aplicación:

- Sistemas de software que combinan diferentes modelos de herencia para optimizar la reutilización del código.

Conclusión

La herencia es un instrumento crucial en la Programación Orientada a Objetos que facilita la reutilización del código y la estructuración jerárquica de las clases. Según la complejidad del sistema y las necesidades del proyecto, pueden utilizarse diversas modalidades de herencia.

En C#, pese a que no se soporta directamente la herencia múltiple, las interfaces facilitan alcanzar un comportamiento parecido sin comprometer la integridad del diseño. Es crucial seleccionar el tipo de herencia correcto para asegurar un código más puro, flexible y fácil de mantener.