INSTITUTO KHIPU

Semestre III

Sesión 18

CARRERA PROFESIONAL

DESARROLLO DE SISTEMAS DE INFORMACIÓN

HERRAMIENTAS DE PROGRAMACION C#

Tema

POLIMORFISMO

POLIMORFISMO

El polimorfismo es uno de los cuatro pilares de la Programación Orientada a Objetos (POO), junto con la encapsulación, la abstracción y la herencia. En términos simples, el polimorfismo se refiere a la capacidad de un objeto de tomar muchas formas. En el contexto de C#, el polimorfismo permite que una clase base se comporte de manera diferente dependiendo de qué clase derivada la esté utilizando.

Este concepto es esencial para escribir código flexible y reutilizable, ya que permite que diferentes clases implementen los mismos métodos de formas distintas.

Tipos de Polimorfismo

En C#, el polimorfismo se puede clasificar en dos tipos principales:

1. **Polimorfismo en tiempo de compilación (polimorfismo estático)**.
2. **Polimorfismo en tiempo de ejecución (polimorfismo dinámico)**.

1. Polimorfismo en Tiempo de Compilación (Sobrecarga)

El **polimorfismo en tiempo de compilación** ocurre cuando un método o un operador tiene múltiples definiciones, permitiendo que se comporten de manera diferente en función de la lista de parámetros que reciban. Esto se conoce como **sobrecarga de métodos** y **sobrecarga de operadores**.

Sobrecarga de Métodos

La **sobrecarga de métodos** consiste en definir múltiples versiones de un mismo método dentro de una clase, pero con diferentes firmas (parámetros). Esto permite

que un método pueda ser llamado de diferentes formas según el número o tipo de argumentos pasados.

Ejemplo de Sobrecarga de Métodos

```
class Calculadora
{
    // Sobrecarga de métodos
    public int Sumar(int a, int b)
    {
        return a + b;
    }

    public int Sumar(int a, int b, int c)
    {
        return a + b + c;
    }

    public double Sumar(double a, double b)
    {
        return a + b;
    }
}
```

```
class Programa
{
    static void Main()
    {
        Calculadora calc = new Calculadora();

        Console.WriteLine(calc.Sumar(3, 4)); // 7
        Console.WriteLine(calc.Sumar(3, 4, 5)); // 12
        Console.WriteLine(calc.Sumar(2.5, 3.5)); // 6.0
    }
}
```

En este ejemplo, el método Sumar se ha sobrecargado tres veces con diferentes parámetros: dos enteros, tres enteros y dos dobles. C# selecciona automáticamente cuál de los métodos sobrecargados debe ejecutarse en función de los tipos de parámetros pasados al llamar al método.

Sobrecarga de Operadores

La sobrecarga de operadores permite que los operadores estándar (como +, -, *, ==, etc.) se comporten de manera especial cuando se aplican a instancias de clases personalizadas.

Ejemplo de Sobrecarga de Operadores

```
using System;

9 referencias
class Complejo
{
    6 referencias
    public int Real { get; set; }
    6 referencias
    public int Imaginario { get; set; }

    // Sobrecarga del operador +
    1 referencia
    public static Complejo operator +(Complejo c1, Complejo c2)
    {
        return new Complejo { Real = c1.Real + c2.Real, Imaginario = c1.Imaginario + c2.Imaginario };
    }
}
```

```
0 referencias
class Programa
{
    0 referencias
    static void Main()
    {
        Complejo c1 = new Complejo { Real = 1, Imaginario = 2 };
        Complejo c2 = new Complejo { Real = 3, Imaginario = 4 };
        Complejo resultado = c1 + c2;

        Console.WriteLine($"Resultado: {resultado.Real} + {resultado.Imaginario}i"); // 4 + 6i
    }
}
```

En este ejemplo, sobrecargamos el operador + para que funcione con objetos de la clase Complejo, sumando sus partes reales e imaginarias.

2. Polimorfismo en Tiempo de Ejecución (Sobreescritura)

El **polimorfismo en tiempo de ejecución** se basa en la **herencia** y en la capacidad de una clase derivada para **sobrescribir** un método de la clase base. Esto se logra utilizando las palabras clave **virtual**, **override** y **abstract**.

En el polimorfismo en tiempo de ejecución, cuando un método definido en la clase base se marca como virtual, una clase derivada puede sobrescribir ese método usando la palabra clave override. Esto permite que una misma llamada a un método se comporte de manera diferente según el tipo del objeto que la invoque.

Ejemplo de Sobreescritura de Métodos

```
class Animal
{
    // Método virtual que puede ser sobrescrito
    public virtual void HacerSonido()
    {
        Console.WriteLine("El animal hace un sonido.");
    }
}

class Perro : Animal
{
    // Sobrescribe el método virtual
    public override void HacerSonido()
    {
        Console.WriteLine("El perro ladra.");
    }
}
```

```
class Gato : Animal
{
    // Sobrescribe el método virtual
    public override void HacerSonido()
    {
        Console.WriteLine("El gato maúlla.");
    }
}

class Programa
{
    static void Main()
    {
        Animal miAnimal1 = new Perro();
        Animal miAnimal2 = new Gato();

        miAnimal1.HacerSonido(); // El perro ladra.
        miAnimal2.HacerSonido(); // El gato maúlla.
    }
}
```

En este ejemplo, aunque miAnimal1 y miAnimal2 son del tipo Animal, el método HacerSonido() se comporta de manera diferente dependiendo de si el objeto es un Perro o un Gato. Este es el **polimorfismo en tiempo de ejecución**.

Polimorfismo con Interfaces

El polimorfismo también se puede lograr a través de **interfaces**. Cuando varias clases implementan la misma interfaz, los objetos pueden tratarse como si fueran de la interfaz en lugar de la clase específica.

Ejemplo con Interfaces

```
interface IVehiculo
{
    void Arrancar();
}

class Coche : IVehiculo
{
    public void Arrancar()
    {
        Console.WriteLine("El coche ha arrancado.");
    }
}

class Motocicleta : IVehiculo
{
    public void Arrancar()
    {
        Console.WriteLine("La motocicleta ha arrancado.");
    }
}

class Programa
{
    static void Main()
    {
        IVehiculo vehiculo1 = new Coche();
        IVehiculo vehiculo2 = new Motocicleta();

        vehiculo1.Arrancar(); // El coche ha arrancado.
        vehiculo2.Arrancar(); // La motocicleta ha arrancado.
    }
}
```

En este ejemplo, Coche y Motocicleta implementan la interfaz IVehiculo. Esto permite que tanto el coche como la motocicleta se traten como vehículos (IVehiculo), logrando polimorfismo a través de interfaces.

Beneficios del Polimorfismo

1. **Reutilización del código**: Puedes reutilizar el mismo código en diferentes clases.
2. **Flexibilidad**: Permite que los programas sean más flexibles y extensibles, ya que las clases pueden sobrescribir comportamientos según lo necesiten.
3. **Mantenimiento**: Hace que el código sea más fácil de mantener, ya que el comportamiento compartido puede estar centralizado en la clase base y el comportamiento específico se puede manejar en las clases derivadas.
4. **Abstracción**: Permite a los desarrolladores escribir código que puede funcionar con objetos de diferentes tipos, siempre y cuando compartan el mismo contrato (ya sea a través de herencia o interfaces).

Conclusión

El polimorfismo es un concepto esencial en C# que permite a los desarrolladores escribir código más flexible, reutilizable y mantenible. Al utilizar sobrecarga de métodos, sobrecarga de operadores, y sobrescritura de métodos a través de herencia o interfaces, los programas pueden manejar diferentes tipos de objetos de manera uniforme, lo que facilita la extensión y modificación del comportamiento sin cambiar el código existente.

INSTITUTO

KHIPU