**Evaluation Only. Created with Aspose.Words. Copyright 2003-2021 Aspose Pty Ltd.**

Conceptos de Orientación de Objetos – Curso Completo de Desarrollo C Sharp – Ángel Arias

**Conceptos de Orientación de Objetos**

**Clase**

Es una colección de objetos. **Objeto**

Es una entidad en tiempo real.

Un objeto puede considerarse una "cosa" que puede realizar un conjunto de actividades relacionadas. El conjunto de actividades que realiza el objeto define el comportamiento del objeto. Por ejemplo, una persona (objeto) puede correr o saltar. En términos POO puros, un objeto es una instancia de una clase.

El siguiente ejemplo describe la clase Persona.

**Persona**

* nombre: string;
* edad: int;
* Persona (string n, int e);
* Correr();
* Saltar ();

La clase se compone de tres cosas: nombre, atributos y métodos

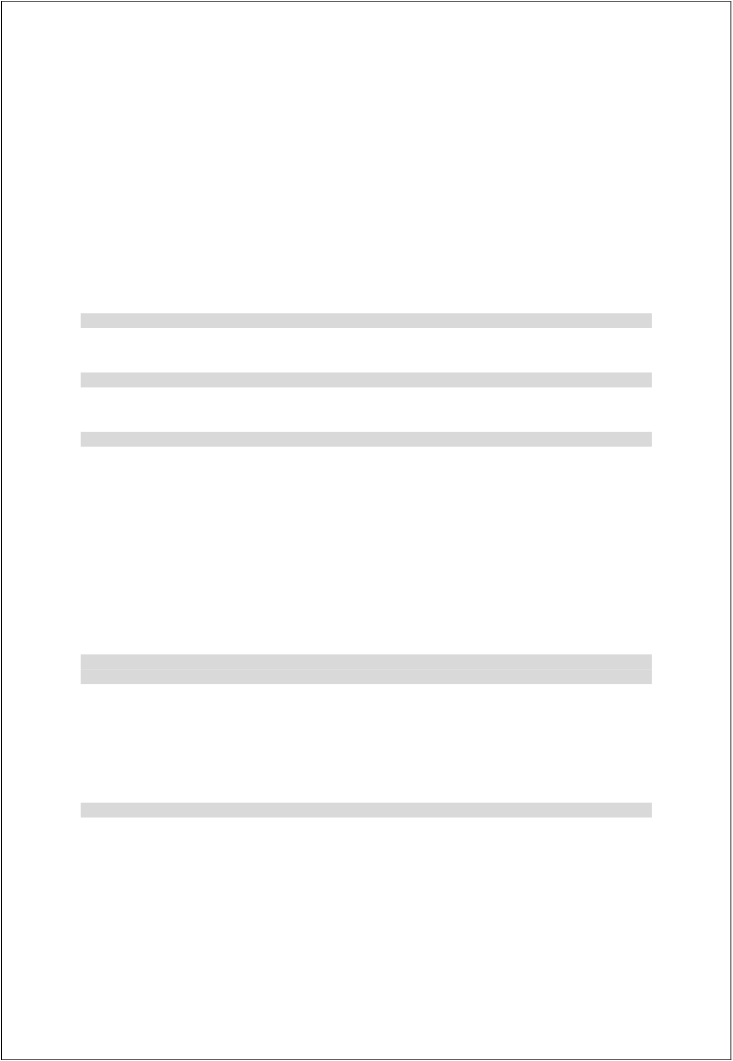
public class persona {}

persona objetoPersona = new persona();

En el ejemplo anterior podemos decir que el objeto **Persona**, llamado **objetoPersona**, se ha creado fuera de la clase de personas.

En el mundo real, a menudo encontrarás muchos objetos individuales del mismo tipo. Por ejemplo, puede haber miles de motos en existencia, todas de la misma marca y modelo. Cada moto se ha construido a partir del mismo plano de diseño. En términos orientados a objetos, decimos que la  **moto** es una instancia de la clase de objetos conocidos como **motos**.

Conceptos de Orientación de Objetos – Curso Completo de Desarrollo C Sharp – Ángel Arias



**La encapsulación**

La encapsulación es un proceso de enlace de los miembros de datos y las funciones de los miembros en una sola unidad.

Un ejemplo de encapsulación es la clase. Una clase puede contener estructuras de datos y métodos.

Veamos la siguiente clase:

public class EjemploEncapsulacion {

public class Apertura

{

public Apertura() {

}

double altura; double ancho; double grosor;

public double obtener\_volumen()

{

Console.WriteLine("Introduce la Altura");

altura = Convert.ToDouble(Console.ReadLine()); Console.WriteLine("Introduce la Ancho");

ancho = Convert.ToDouble(Console.ReadLine()); Console.WriteLine("Introduce la Grosor");

grosor = Convert.ToDouble(Console.ReadLine()); double volumen = altura \* ancho \* grosor;

if (volumen < 0)

return 0;

return volumen;

}

}

public static void Main()

{

double volumen;

Apertura a = new Apertura(); volumen = a.obtener\_volumen(); Console.WriteLine(volumen);

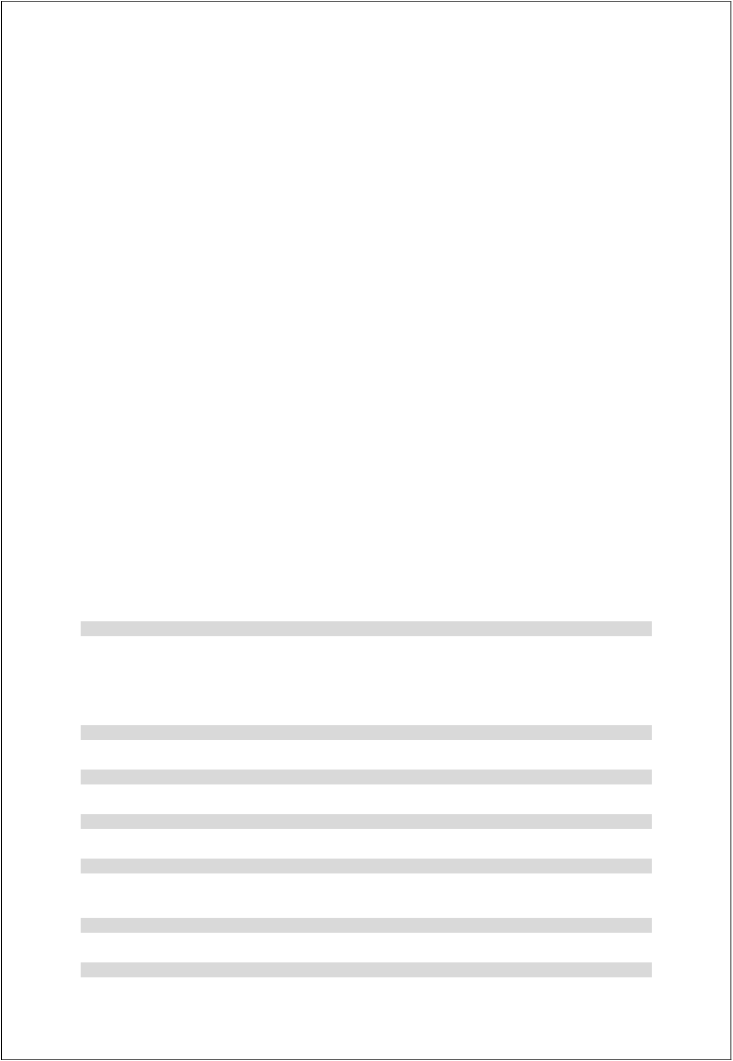
Console.ReadKey();

}

}

En este ejemplo, resumimos algunos datos como altura, ancho, grosor y el método **obtener\_volumen()**. Otros métodos u objetos podrán interactuar con este objeto a través de métodos que tengan un modificador de acceso público

Conceptos de Orientación de Objetos – Curso Completo de Desarrollo C Sharp – Ángel Arias



**Abstracción**

La abstracción es un proceso de ocultar los detalles de la implementación y mostrar las características esenciales.

Por Ejemplo: una computadora portátil consta de muchas cosas, como procesador, placa base, RAM, teclado, pantalla, antena inalámbrica, cámara web, puertos USB, batería, altavoces, etc. Para usarlo, no necesitas saber cómo funcionan las pantallas LCD internas, teclado, cámara web, batería, antena inalámbrica, trabajos de altavoz. Solo necesita saber cómo operar la computadora portátil. Imagínate si los usuarios tuviesen que conocer todos los detalles internos de la computadora portátil antes de poder trabajar con ella. Esto tendría un costo muy elevado y no sería fácil que los usuarios pudiesen usarlas.

Así que aquí el portátil es un objeto que está diseñado para ocultar su complejidad. Para poder abstraer usamos los especificadores de acceso

* **Public**: accesible fuera de la clase a través de referencia de objeto.
* **Private**: accesible dentro de la clase solo a través de funciones miembro.
* **Protected**: Al igual que privado pero accesible en clases derivadas también a través de funciones miembro.
* **Internal**: Visible dentro del conjunto. Accesible a través de los objetos.
* **Protected internal**: visible dentro del ensamblaje a través de objetos y en clases derivadas fuera del ensamblaje a través de funciones miembro.

Veamos un ejemplo práctico:

using System;

using System.Runtime.InteropServices;

public class EjemploAbstraccion

{

public class Clase1

{

//Si no especifica Acceso, es privado int i;

// Publico public int j;

//Protected protected int k;

// Internal quiere decir que es visible dentro del ensamblado internal int m;

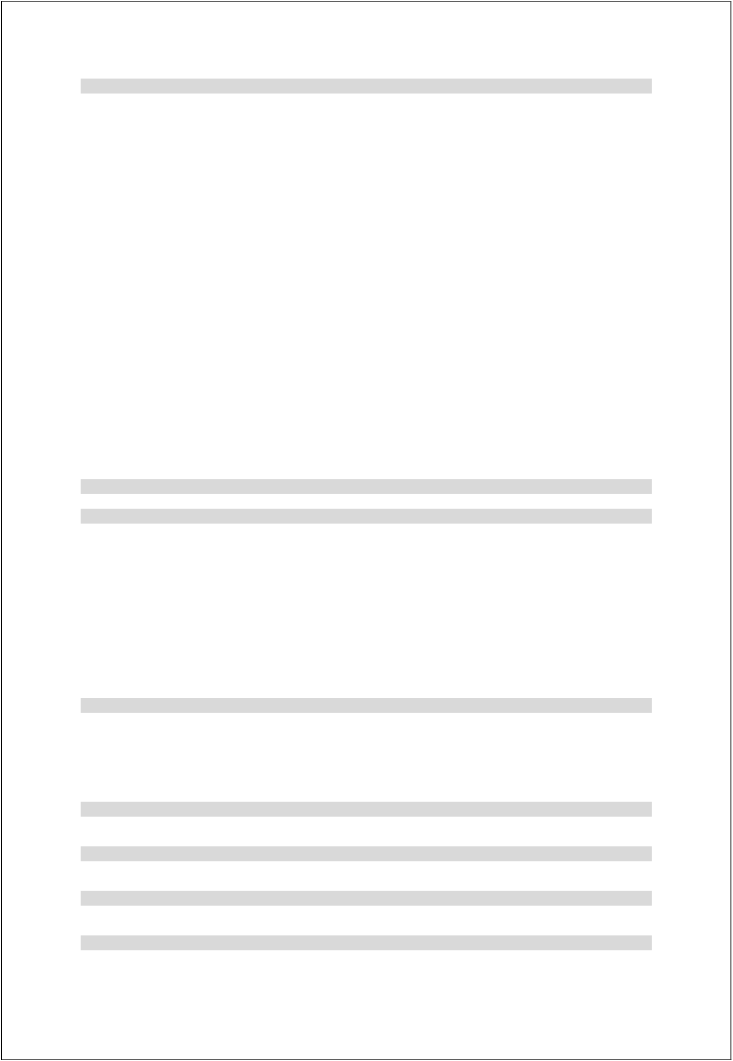
// Acceso desde ensamblaje interno así como desde clases derivadas fuera de ensamblaje

protected internal int n;

// También private static int x;

//Static significa compartido entre objetos

Conceptos de Orientación de Objetos – Curso Completo de Desarrollo C Sharp – Ángel Arias



public static int y;

//externo quiere decir que ha sido declarado en este ensamblado definido en algún otro ensamblado

[DllImport("MiDll.dll")]

public static extern int MyFoo();

public void metodo()

{

//Dentro de una clase si has creado un objeto de la misma clase entonces puedes acceder a todos los miembros a través de la referencia al objeto incluso si los datos son private

Clase1 objeto = new Clase1();

//Aquí los datos son accesibles

objeto.i = 10;

objeto.j = 10;

objeto.k = 10;

objeto.m = 10;

objeto.n = 10;

// objeto.x =10; //Error los datos Static pueden ser accedidos solo por el nombre de la clase

Clase1.x = 10;

// objeto.y = 10; // Error los datos Static pueden ser accedidos solo por el nombre de la clase

Clase1.y = 10;

}

}

public static void Main()

{

} }

Ahora si accedemos a los mismos miembros desde dentro del  **Main** e intentamos compilar:

using System;

using System.Runtime.InteropServices;

public class EjemploAbstraccion

{

public class Clase1

{

//Si no especifica Acceso, es privado int i;

// Publico public int j;

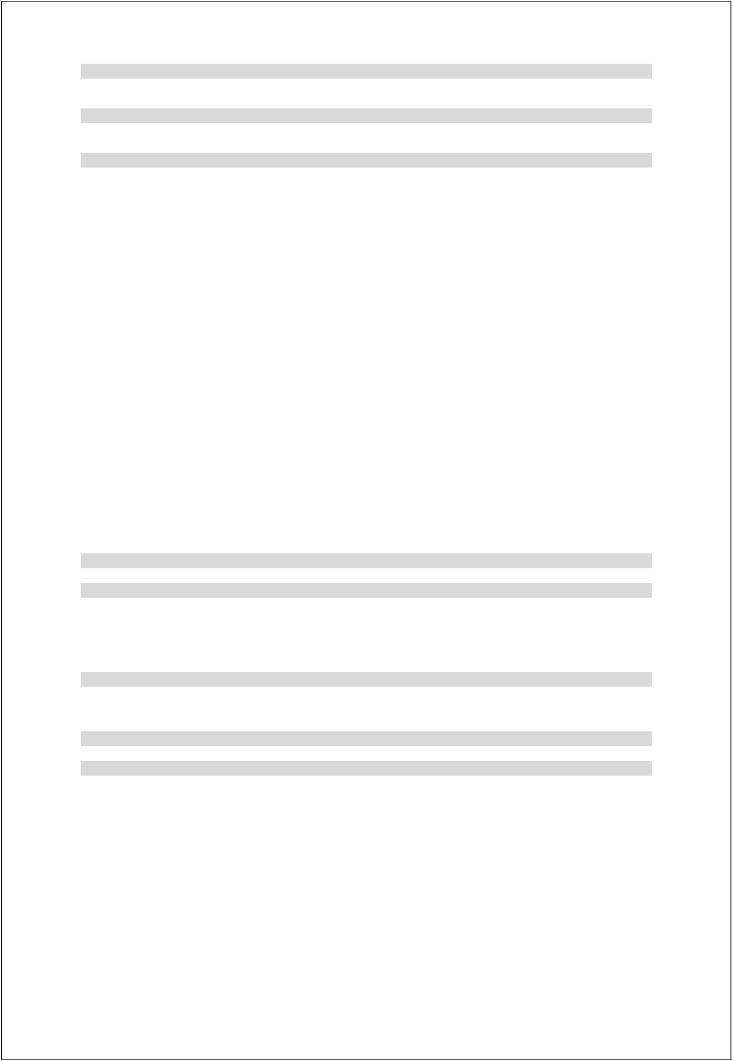
//Protected protected int k;

// Internal quiere decir que es visible dentro del ensamblado internal int m;

// Acceso desde ensamblaje interno así como desde clases derivadas fuera de ensamblaje

protected internal int n;

Conceptos de Orientación de Objetos – Curso Completo de Desarrollo C Sharp – Ángel Arias



// También private static int x;

//Static significa compartido entre objetos public static int y;

//externo quiere decir que ha sido declarado en este ensamblado definido en algún otro ensamblado

[DllImport("MiDll.dll")]

public static extern int MyFoo();

public void metodo()

{

//Dentro de una clase si has creado un objeto de la misma clase entonces puedes acceder a todos los miembros a través de la referencia al objeto incluso si los datos son private

Clase1 objeto = new Clase1();

//Aquí los datos son accesibles

objeto.i = 10;

objeto.j = 10;

objeto.k = 10;

objeto.m = 10;

objeto.n = 10;

// objeto.x =10; //Error los datos Static pueden ser accedidos solo por el nombre de la clase

Clase1.x = 10;

// objeto.y = 10; // Error los datos Static pueden ser accedidos solo por el nombre de la clase

Clase1.y = 10;

}

}

public static void Main()

{

Clase1 objeto = new Clase1();

// objeto.i = 10; // Error debido al nivel de protección - private objeto.j = 20;

// objeto.k = 30; // Error debido al nivel de protección - static objeto.m = 40;

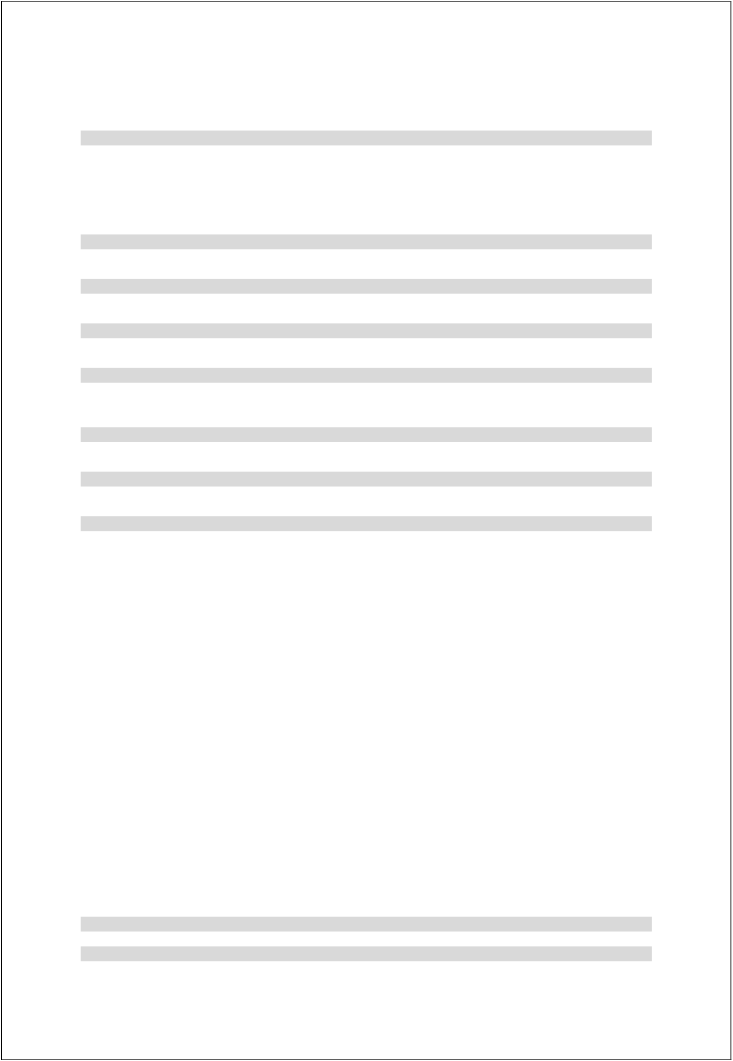
objeto.n = 50;

Console.WriteLine(objeto.j); Console.WriteLine(objeto.m); Console.WriteLine(objeto.n);

Console.ReadKey(); }

}

Conceptos de Orientación de Objetos – Curso Completo de Desarrollo C Sharp – Ángel Arias



¿Qué pasa si **Main** está dentro de otro ensamblaje?

using System;

using System.Runtime.InteropServices;

public class EjemploAbstraccion

{

public class Clase1

{

//Si no especifica Acceso, es privado int i;

// Publico public int j;

//Protected protected int k;

// Internal quiere decir que es visible dentro del ensamblado internal int m;

// Acceso desde ensamblaje interno así como desde clases derivadas fuera de ensamblaje

protected internal int n;

// También private static int x;

//Static significa compartido entre objetos public static int y;

//externo quiere decir que ha sido declarado en este ensamblado definido en algún otro ensamblado

[DllImport("MiDll.dll")]

public static extern int MyFoo();

public void metodo()

{

//Dentro de una clase si has creado un objeto de la misma clase entonces puedes acceder a todos los miembros a través de la referencia al objeto incluso si los datos son private

Clase1 objeto = new Clase1();

//Aquí los datos son accesibles

objeto.i = 10;

objeto.j = 10;

objeto.k = 10;

objeto.m = 10;

objeto.n = 10;

// objeto.s =10; //Error los datos Static pueden ser accedidos solo por el nombre de la clase

Clase1.x = 10;

// objeto.y = 10; // Error los datos Static pueden ser accedidos solo por el nombre de la clase

Clase1.y = 10;

}

}

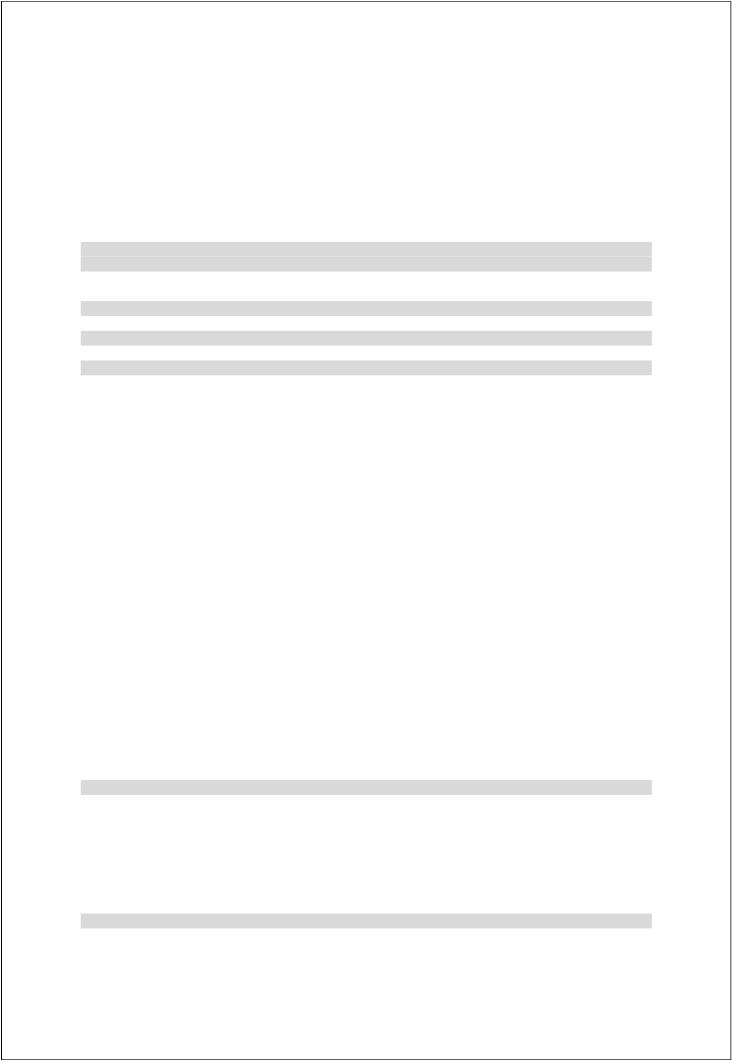
public static void Main()

{

Clase1 objeto = new Clase1();

// objeto.i = 10; // Error debido al nivel de protección objeto.j = 20;

Conceptos de Orientación de Objetos – Curso Completo de Desarrollo C Sharp – Ángel Arias



// objeto.k = 30; // Error debido al nivel de protección

// objeto.m = 40; // Error debido a que no puede acceder a los datos internos fuera del ensamblaje

// objeto.n = 50; // Error debido a que no puede acceder a los datos internos fuera del ensamblaje

// objeto.s =10; // Error debido a que solo se puede acceder a los datos Static por nombres de clase

// Clase1.x = 10; // Error debido a que no se puede acceder a datos privados fuera de clase

// objeto.y = 10; // Error debido a que solo se puede acceder a los datos Static por nombres de clase

Clase1.y = 10;

Console.WriteLine(objeto.j); Console.WriteLine(Clase1.y);

Console.ReadKey(); }

}

En la programación orientada a objetos, la complejidad se gestiona utilizando la abstracción. La abstracción es un proceso que implica identificar el comportamiento crítico de un objeto y eliminar detalles irrelevantes y complejos.

**Herencia**

La herencia es el proceso de derivar la nueva clase de una clase ya existente.

C # es un lenguaje de programación orientado a objetos. La herencia es uno de los conceptos principales de la programación orientada a objetos. Te permite reutilizar el código existente. A través del uso efectivo de la herencia, puedes ahorrar mucho tiempo en la programación y también reducir los errores, lo que a su vez aumentará la calidad del trabajo y la productividad. Un ejemplo simple para entender la herencia en C #.

using System;

public class EjemploHerencia

{

**This document was truncated here because it was created in the Evaluation Mode.**