La implementación System.ValueType de Equals utiliza la reflexión porque debe ser capaz de determinar qué son los campos en cualquier struct. Al crear sus propios structs, invalide el método Equals para proporcionar un algoritmo de igualdad eficaz específico de su tipo.

- Para determinar si los valores de los campos de dos instancias de clase son iguales, puede usar el método Equals o el operador ==. Sin embargo, se deben utilizar solo si la clase los ha invalidado o sobrecargado para proporcionar una definición personalizada de lo que significa "igualdad" para los objetos de ese tipo. La clase también puede implementar las interfaces IEquatable(Of T) o IEqualityComparer(Of T). Ambas interfaces proporcionan métodos que se pueden utilizar para probar si sus valores son iguales. Al diseñar sus propias clases que invaliden **Equals**, asegúrese de seguir las directrices indicadas en Cómo: Definir la igualdad de valores para un tipo (Guía de programación de C#) y Object.Equals(Object).

//

Propiedades (Guía de programación de C#)

Visual Studio 2010
Otras versiones

Una propiedad es un miembro que ofrece un mecanismo flexible para leer, escribir o calcular el valor de un campo privado. Las propiedades pueden utilizarse como si fuesen miembros de datos públicos, aunque en realidad son métodos especiales denominados *descriptores de acceso*. De este modo, se puede obtener acceso a los datos con facilidad, a la vez que se promueve la seguridad y flexibilidad de los métodos.

En este ejemplo, la clase TimePeriod almacena un período de tiempo. Internamente, la clase almacena el tiempo en segundos, pero una propiedad denominada Hours permite a un cliente especificar el tiempo en horas. Los descriptores de acceso de la propiedad Hours realizan la conversión entre horas y segundos.

Ejemplo

C#

```
class TimePeriod
{
    private double seconds;

    public double Hours
    {
        get { return seconds / 3600; }
```

```
        set { seconds = value * 3600; }
    }
}


class Program
{
    static void Main()
    {
        TimePeriod t = new TimePeriod();

        // Assigning the Hours property causes the 'set' accessor to be called.
        t.Hours = 24;

        // Evaluating the Hours property causes the 'get' accessor to be called.
        System.Console.WriteLine("Time in hours: " + t.Hours);
    }
}
// Output: Time in hours: 24
```

Información general sobre propiedades

- Las propiedades permiten que una clase exponga una manera pública de obtener y establecer valores, ocultando el código de implementación o comprobación.
- El descriptor de acceso de una propiedad get se utiliza para devolver el valor de la propiedad y el descriptor de acceso set se utiliza para asignar un nuevo valor. Estos descriptores de acceso pueden tener niveles de acceso diferentes. Para obtener más información, vea Restringir la accesibilidad del descriptor de acceso (Guía de programación de C#).
- La palabra clave value se usa para definir el valor asignado por el descriptor de acceso set.
- Las propiedades que no implementan un descriptor de acceso set son de solo lectura.
- En el caso de las propiedades simples que no requieren ningún código de descriptor de acceso personalizado, considere la posibilidad de utilizar propiedades autoimplementadas. Para obtener más información, vea Propiedades autoimplementadas (Guía de programación de C#).

//

Interfaces (Guía de programación de C#)

Visual Studio 2010

Otras versiones

Las interfaces describen un grupo de funcionalidades relacionadas que pueden pertenecer a cualquier elemento class o struct. Para definir una interfaz, utilice la palabra claveinterface, como se muestra en el ejemplo siguiente.

C#

```
interface IEquatable<T>
{
   bool Equals(T obj);
}
```

Las interfaces están compuestas de métodos, propiedades, eventos, indizadores o cualquier combinación de estos cuatro tipos de miembros. Una interfaz no puede contener constantes, campos, operadores, constructores de instancias, destructores o tipos. No puede contener miembros estáticos. Los miembros de interfaz son automáticamente públicos y no pueden incluir ningún modificador de acceso.

Cuando una clase o struct implementa una interfaz, dicha clase o struct proporciona una implementación para todos los miembros que define la interfaz. La propia interfaz no proporciona ninguna funcionalidad que una clase o struct pueda heredar de la manera en que se puede heredar la funcionalidad de una clase base. Sin embargo, si una clase base implementa una interfaz, la clase derivada hereda esta implementación. Se dice que la clase derivada implementa la interfaz de forma implícita.

Al igual que las clases heredan una clase base o struct, las clases y structs implementan interfaces, con dos excepciones:

- Una clase o struct puede implementar más de una interfaz.
- Cuando una clase o struct implementan una interfaz, recibe sólo los nombres de métodos y las firmas, ya que la propia interfaz no contiene ninguna implementación, como se muestra en el ejemplo siguiente.

  C#

```
public class Car : IEquatable<Car>
{
    public string Make {get; set;}
    public string Model { get; set; }
    public string Year { get; set; }

    // Implementation of IEquatable<T> interface
    public bool Equals(Car car)
    {
        if (this.Make == car.Make &&
            this.Model == car.Model &&
            this.Year == car.Year)
```

```
            {
                return true;
            }
            else
                return false;
        }
    }
```

La interfaz IEquatable(Of T) informa al usuario del objeto de que éste puede determinar si es igual que otros objetos del mismo tipo; el usuario de la interfaz no necesita saber cómo se implementa.

Para implementar un miembro de interfaz, el miembro correspondiente de la clase debe ser público, no estático y tener el mismo nombre y la misma firma que el miembro de interfaz. Las propiedades e indizadores de una clase pueden definir descriptores de acceso adicionales para una propiedad o indizador definidos en una interfaz. Por ejemplo, una interfaz puede declarar una propiedad que tiene un descriptor de acceso get. La clase que implementa la interfaz puede declarar la misma propiedad con los descriptores de acceso get y set. Sin embargo, si la propiedad o el indizador utiliza una implementación explícita, los descriptores de acceso deben coincidir. Para obtener más información sobre la implementación explícita, vea ~~Propiedades de interfaces (Guía de programación de C#)~~.

Las interfaces y los miembros de interfaz son abstractos; las interfaces no proporcionan una implementación predeterminada. Para obtener más información, vea Clases y miembros de clase abstractos y sellados (Guía de programación de C#).

Las interfaces pueden heredar otras interfaces. Es posible que una clase herede una interfaz varias veces, a través de las clases base que hereda o de interfaces heredadas por otras interfaces. Sin embargo, la clase puede implementar una interfaz solo una vez y solo si la interfaz se declara como parte de la definición de clase, como en class ClassName : InterfaceName. Si la interfaz se hereda porque se heredó una clase base que implementa esta interfaz, dicha clase base proporcionará su implementación. También es posible que una clase base implemente miembros de interfaz a través de miembros virtuales. En ese caso, una clase derivada puede cambiar el comportamiento de la interfaz al invalidar los miembros virtuales. Para obtener más información sobre miembros virtuales, vea Polimorfismo (Guía de programación de C#).

Información general sobre interfaces

Una interfaz tiene las siguientes propiedades:

- Una interfaz es como una clase base abstracta: cualquier tipo no abstracto que implementa la interfaz debe implementar todos sus miembros.
- No se pueden crear instancias directamente de una interfaz.
- Las interfaces pueden contener eventos, métodos, indizadores y propiedades.

- Las interfaces no contienen implementaciones de métodos.
- Las clases y structs pueden implementar más de una interfaz.
- Una interfaz se puede heredar de varias interfaces.

//