



Instituto Tecnológico Nacional de
México

Instituto tecnológico del Sur de
Nayarit

Programación Orientada a Objetos

Tema 4: Polimorfismo y Herencia

Interfaces en C#

Cinthia Anahí Mata Bravo

Docente

Deisi Jacqueline Ramos Rosas

Estudiante

Semestre: 2



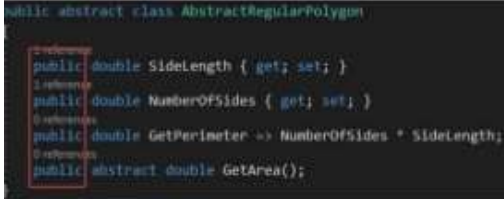
Matricula: 191140033

Fecha de Entrega: 20-marzo-2020

Índice

Concepto e Implementación de Interfaces C#.....	3
Interfaces:	4
Implementando una interfaz en C#	5
Resumen Personal.....	7

Concepto e Implementación de Interfaces C#

	Interfaces	Clases Abstractas
¿Qué son?	<p>Describen un grupo de funciones relacionadas, que pueden pertenecer a cualquier clase o estructura.</p> <p>Básicamente se trata de representar un contrato que debe cumplir cualquier clase que implemente la interfaz.</p>	<p>Son mecanismos que obligan la herencia. No se pueden instanciar, es decir, no se puede crear objetos de ellas. Se utilizan solamente para heredar de ellas (Forzar u obligar la herencia). Se antepone la palabra "abstract" al nombre de la clase.</p> <p>Se puede sobrescribir los miembros, métodos y funciones de una clase abstracta, e incluso extender la clase que herede de ella con las particularidades que consideremos oportunas.</p>
Implementación	NO poseen implementaciones.	<p>Puede tener implementaciones como se observa en la imagen1</p>  <pre> public abstract class AbstractRegularPolygon { //reference public double SideLength { get; set; } //reference public double NumberOfSides { get; set; } //reference public double GetPerimeter => NumberOfSides * SideLength; } </pre> <p><i>Imagen1</i></p>
Herencia	Herencia Múltiple	Herencia Simple
Modificadores de Acceso	<p>Automáticamente todos sus miembros son públicos, como se observa en la imagen2</p>  <pre> public interface IRegularPolygon { //reference double SideLength { get; set; } //reference double NumberOfSides { get; set; } //reference double GetPerimeter(); //reference double GetArea(); } </pre> <p><i>Imagen2</i></p>	<p>Puede tener modificadores de acceso, como se ve en la imagen3</p>  <pre> public abstract class AbstractRegularPolygon { //reference public double SideLength { get; set; } //reference public double NumberOfSides { get; set; } //reference public double GetPerimeter => NumberOfSides * SideLength; //reference public abstract double GetArea(); } </pre> <p><i>Imagen3</i></p>
Miembros Válidos	<p>Propiedades</p> <p>Métodos</p> <p>Eventos</p> <p>Índices</p>	<p>Campos, Propiedades</p> <p>Constructores, Destructores</p> <p>Métodos, Eventos, Índices</p>

Interfaces:

Una interfaz no es más que una estructura de datos que muestra únicamente las firmas de los métodos de una clase. A partir de ahí, una clase que herede de la interfaz estará obligada a “rellenar” la implementación de dichos métodos.

Ejemplo:

Interfaz Motor

Arrancar();

Detener();

Esta interfaz únicamente dice QUÉ acciones se van a realizar, pero no CÓMO se realizarán. Para saber cómo realizarlas, crearemos clases que implementen esta interfaz, obligando a “rellenar” el contenido de los métodos que declaramos previamente.

Clase Gasolina : Motor

Arrancar()

InyectarAire();

IniciarCarburacion();

EncenderMezcla(Bujía);

Detener()

DetenerCarburacion();

Clase Diesel : Motor

Arrancar()

InyectarCombustible();

InyectarAire();

Comprimir();

Detener()

DetenerInyeccion();

Ahora imagine que se tiene una clase “Coche” que necesite un “Motor”. Esta clase debe ignorar el funcionamiento interno del motor, únicamente debe saber cómo arrancarlo y cómo pararlo.

Clase Vehiculo

Motor m;

Para ello hará uso de los métodos Arrancar() y Parar(). Esta interfaz será donde se “conecte” el motor, independientemente de si es un Gasolina o un Diesel. Por ello, si quisiéramos utilizar un motor Diesel, haríamos lo siguiente:

m = new Diesel();

m.Arrancar();

Como puede observar, para un motor gasolina sería igual, salvo que declararíamos una instancia de la clase Gasolina.

Implementando una interfaz en C#

Como ejemplo de implementación y utilización de interfaces en C#, se creará una interfaz ***ITest*** que implemente dos métodos: ***HolaMundo*** y ***DevolverCadena***. Esta interfaz será implementada por dos clases: ***Minusculas*** y ***Mayusculas***. La primera devolverá cadenas en minúsculas, mientras que la segunda, lo hará en mayúsculas. Para comenzar, crearemos tres clases: *ITest*, *Minusculas* y *Mayusculas*. Estableceremos el cuerpo de *ITest* de la siguiente forma:

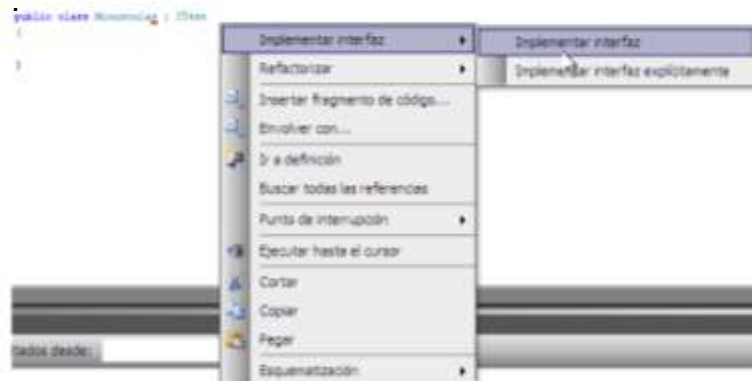
```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace Simple.Interface
6  {
7      public interface ITest
8      {
9          string HolaMundo();
10         string DevolverCadena(string s);
11     }
12 }
```

Se establecen los nombres de las funciones junto a sus parámetros (sin establecer niveles de visibilidad, como *public*, *protected* o *private*). Hecho esto, tenemos declarada una interfaz.

El segundo paso será crear las clases que la implementen. Comenzando por “*Minusculas*”. Lo único que se debe hacer, será añadir dos puntos tras el nombre de la clase, y a continuación, el nombre de la clase o interfaz de la que hereda, en este caso, *ITest*.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace Simple.Interface
6  {
7      public class Minusculas : ITest
8      {
9
10     }
11 }
```

Acto seguido, se hará click derecho sobre “*ITest*” y seleccionaremos la opción “Implementar interfaz”.



Con esto se añadirán automáticamente al código aquellos elementos que necesiten ser implementados.

```
1  #region Miembros de ITest
2
3  public string HolaMundo()
4  {
5      throw new Exception("The method or operation is not implemented.");
6  }
7
8  public string DevolverCadena(string s)
9  {
10     throw new Exception("The method or operation is not implemented.");
11 }
12
13 #endregion
```

Por último, se sustituyen las sentencias throw por nuestro propio código, que será el siguiente:

```
1  #region Miembros de ITest
2
3  public string HolaMundo()
4  {
5      return ("hola, mundo");
6  }
7
8  public string DevolverCadena(string s)
9  {
10     return s.ToLower();
11 }
12
13 #endregion
```

Con la clase *Mayusculas* se realiza la misma operación, solo que la implementación será la siguiente:

```
1  #region Miembros de ITest
2
3  public string HolaMundo()
4  {
5      return ("HOLA, MUNDO");
6  }
7
8  public string DevolverCadena(string s)
9  {
10     return s.ToUpper();
11 }
12
13 #endregion
```

Hecho esto, se le asocia una instancia de cualquiera de las clases que la implementen:

```
1  ITest minusculas = new Minusculas();
2  ITest mayusculas = new Mayusculas();
3
4  Console.WriteLine(minusculas.HolaMundo());
5  Console.WriteLine(mayusculas.HolaMundo());
6
7  Console.WriteLine("-----");
8
9  Console.WriteLine(minusculas.DevolverCadena("Esto es una Prueba de INTERFACES"));
10 Console.WriteLine(mayusculas.DevolverCadena("Esto es una Prueba de INTERFACES"));
```

Como se observa a continuación, la interfaz ITest conoce los métodos que tiene, pero no cómo ejecutarlos. Es por ello que, al asociarle una instancia que implemente la interfaz, la referencia se comportará de una forma u otra. Como se observa en la imagen3



```
hola, mundo
HOLA, MUNDO
-----
esto es una prueba de interfaces
ESTO ES UNA PRUEBA DE INTERFACES
```

Imagen3

Resumen Personal

- ✓ Las interfaces son estructuras de datos, que contienen métodos sin cuerpo, pero las clases que hereden de esta interfaz están sujetas a rellenar la implementación de dichos métodos.
- ✓ Las Interfaces dicen únicamente que acciones se van a implementar, pero no como serán implementadas
- ✓ Las Interfaces no poseen implementaciones y todos los miembros de esta; son públicos.
- ✓ Las interfaces pueden contener: Propiedades, Métodos, Eventos e Índices, pero, una interfaz no puede contener: Instancias, Constructores o finalizadores.
- ✓ Las Interfaces pueden heredar de múltiples Interfaces.

Dicho de otro modo, las interfaces describen un grupo de funciones relacionadas, que pueden pertenecer a cualquier clase o estructura.