Unidad

4

DIPLOMATURA EN PROGRAMACION JAVA

Capítulo 6

Herencia, Polimorfismo e Interfaces en .Net

Herencia y Polimorfismo en .Net

Herencia

Es frecuente cuando se busca definir clases en un diseño que se encuentren muchas de ellas con características y operaciones parecidas. Sin embargo, con las técnicas de orientación a objetos no es necesario codificar una y otra vez los atributos y métodos similares en las clases ya que se puede reutilizar el código a través de la herencia.

Las definiciones teóricas explican que cuando dos objetos son del mismo tipo, son dos instancias de una misma clase o al menos pertenecen a una misma cadena de herencia.

Este tipo de razonamiento permite descubrir esta clase de relaciones, sin embargo, el código también es una fuente de información para decidir si un grupo de clases pertenecen a la misma cadena de herencia.

Supongamos que se tiene que resolver un problema para un sistema en el cual se deben manejar las clases que representan a los agentes de seguridad (que no son empleados de la empresa), las secretarias, los gerentes y los empleados de una compañía. El siguiente es un primer diseño de estas clases, teniendo en cuenta para comenzar sólo los atributos que las clases necesitan.

Ejemplo

Clase Empleado

C#

```
class Empleado
{
	private String primerNombre;

	public String PrimerNombre
	{
		get { return primerNombre; }
		set { primerNombre = value; }
	}
	private String segundoNombre;

	public String SegundoNombre
	{
		get { return segundoNombre; }
		set { segundoNombre = value; }
```

```
        }
        private String apellido;

        public String Apellido
        {
                get { return apellido; }
                set { apellido = value; }
        }
        private String documento;

        public String Documento
        {
                get { return documento; }
                set { documento = value; }
        }
        private String departamento;

        public String Departamento
        {
                get { return departamento; }
                set { departamento = value; }
        }
        private String id;

        public String Id
        {
                get { return id; }
                set { id = value; }
        }
        private String idJefe;

        public String IdJefe
        {
                get { return idJefe; }
                set { idJefe = value; }
        }
        private float sueldo;

        public float Sueldo
        {
                get { return sueldo; }
                set { sueldo = value; }
        }

        public Empleado(String primerNombre, String segundoNombre, String apellido,
                String documento, String departamento, String id, String idJefe,
                float sueldo)
        {
                this.primerNombre = primerNombre;
                this.segundoNombre = segundoNombre;
                this.apellido = apellido;
                this.documento = documento;
                this.departamento = departamento;
                this.id = id;
                this.idJefe = idJefe;
                this.sueldo = sueldo;
        }
}
```

VB

```
Public Class Empleado
    Private _primerNombre As String
    Private _segundoNombre As String
    Private _apellido As String
    Private _documento As String
    Private _departamento As String
    Private _id As String
    Private _idJefe As String
    Private _sueldo As Single

    Public Property PrimerNombre() As String
        Get
            PrimerNombre = _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property

    Public Property SegundoNombre() As String
        Get
            SegundoNombre = _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = value
        End Set
    End Property

    Public Property Apellido() As String
        Get
            Apellido = _apellido
        End Get
        Set(ByVal value As String)
            _apellido = value
        End Set
    End Property

    Public Property Documento() As String
        Get
            Documento = _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
    End Property

    Public Property Departamento() As String
        Get
            Departamento = _departamento
        End Get
        Set(ByVal value As String)
            _departamento = value
        End Set
    End Property

    Public Property Id() As String
        Get
```

```
            Id = _id
        End Get
        Set(ByVal value As String)
            _id = value
        End Set
    End Property

    Public Property IdJefe() As String
        Get
            IdJefe = _idJefe
        End Get
        Set(ByVal value As String)
            _idJefe = value
        End Set
    End Property

    Public Property Sueldo() As Single
        Get
            Sueldo = _sueldo
        End Get
        Set(ByVal value As Single)
            _sueldo = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
              ByVal d As String, ByVal departamento As String, _
              ByVal id As String, ByVal idJefe As String, ByVal sueldo As Single)
        _primerNombre = p
        _segundoNombre = s
        _apellido = a
        _documento = d
        _departamento = departamento
        _id = id
        _idJefe = idJefe
        _sueldo = sueldo
    End Sub

End Class
```

Clase AgenteDeSeguridad

C#

```
class AgenteDeSeguridad
{
       private String primerNombre;

       public String PrimerNombre
       {
              get { return primerNombre; }
              set { primerNombre = value; }
       }
       private String segundoNombre;

       public String SegundoNombre
       {
              get { return segundoNombre; }
```

```
            set { segundoNombre = value; }
        }
        private String apellido;

        public String Apellido
        {
            get { return apellido; }
            set { apellido = value; }
        }
        private String documento;

        public String Documento
        {
            get { return documento; }
            set { documento = value; }
        }
        private String nroLicencia;

        public String NroLicencia
        {
            get { return nroLicencia; }
            set { nroLicencia = value; }
        }

        public AgenteDeSeguridad(String primerNombre, String segundoNombre,
                String apellido, String documento, String nroLicencia)
        {
            this.primerNombre = primerNombre;
            this.segundoNombre = segundoNombre;
            this.apellido = apellido;
            this.documento = documento;
            this.nroLicencia = nroLicencia;
        }
}
```

VB

```
Public Class AgenteDeSeguridad
    Private _primerNombre As String
    Private _segundoNombre As String
    Private _apellido As String
    Private _documento As String
    Private _nroLicencia As String

    Public Property PrimerNombre() As String
        Get
            PrimerNombre = _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property

    Public Property SegundoNombre() As String
        Get
            SegundoNombre = _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = value
```

```
        End Set
    End Property

    Public Property Apellido() As String
        Get
            Apellido = _apellido
        End Get
        Set(ByVal value As String)
            _apellido = value
        End Set
    End Property

    Public Property Documento() As String
        Get
            Documento = _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
    End Property

    Public Property NroLicencia() As String
        Get
            NroLicencia = _nroLicencia
        End Get
        Set(ByVal value As String)
            _nroLicencia = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                   ByVal d As String, ByVal nroLicencia As String)
        _primerNombre = p
        _segundoNombre = s
        _apellido = a
        _documento = d
        _nroLicencia = nroLicencia
    End Sub

End Class
```

Clase Secretaria

C#

```
class Secretaria
{
       private String primerNombre;

       public String PrimerNombre
       {
              get { return primerNombre; }
              set { primerNombre = value; }
       }
       private String segundoNombre;

       public String SegundoNombre
       {
```

```
        get { return segundoNombre; }
        set { segundoNombre = value; }
    }
    private String apellido;

    public String Apellido
    {
        get { return apellido; }
        set { apellido = value; }
    }
    private String documento;

    public String Documento
    {
        get { return documento; }
        set { documento = value; }
    }
    private String departamento;

    public String Departamento
    {
        get { return departamento; }
        set { departamento = value; }
    }
    private String id;

    public String Id
    {
        get { return id; }
        set { id = value; }
    }
    private float sueldo;

    public float Sueldo
    {
        get { return sueldo; }
        set { sueldo = value; }
    }
    private String idJefe;

    public String IdJefe
    {
        get { return idJefe; }
        set { idJefe = value; }
    }

    public Secretaria(String primerNombre, String segundoNombre,
        String apellido, String documento, String departamento, String id,
        float sueldo, String idJefe)
    {
        this.primerNombre = primerNombre;
        this.segundoNombre = segundoNombre;
        this.apellido = apellido;
        this.documento = documento;
        this.departamento = departamento;
        this.id = id;
        this.sueldo = sueldo;
        this.idJefe = idJefe;
```

```
        }
}
```

VB

```
Public Class Secretaria
    Private _primerNombre As String
    Private _segundoNombre As String
    Private _apellido As String
    Private _documento As String
    Private _departamento As String
    Private _id As String
    Private _idJefe As String
    Private _sueldo As Single

    Public Property PrimerNombre() As String
        Get
            PrimerNombre = _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property

    Public Property SegundoNombre() As String
        Get
            SegundoNombre = _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = value
        End Set
    End Property

    Public Property Apellido() As String
        Get
            Apellido = _apellido
        End Get
        Set(ByVal value As String)
            _apellido = value
        End Set
    End Property

    Public Property Documento() As String
        Get
            Documento = _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
    End Property

    Public Property Departamento() As String
        Get
            Departamento = _departamento
        End Get
        Set(ByVal value As String)
            _departamento = value
        End Set
    End Property
```

```
    Public Property Id() As String
        Get
            Id = _id
        End Get
        Set(ByVal value As String)
            _id = value
        End Set
    End Property

    Public Property IdJefe() As String
        Get
            IdJefe = _idJefe
        End Get
        Set(ByVal value As String)
            _idJefe = value
        End Set
    End Property

    Public Property Sueldo() As Single
        Get
            Sueldo = _sueldo
        End Get
        Set(ByVal value As Single)
            _sueldo = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
              ByVal d As String, ByVal departamento As String, _
              ByVal id As String, ByVal idJefe As String, ByVal sueldo As Single)
        _primerNombre = p
        _segundoNombre = s
        _apellido = a
        _documento = d
        _departamento = departamento
        _id = id
        _idJefe = idJefe
        _sueldo = sueldo
    End Sub

End Class
```

Clase Gerente

C#

```
class Gerente
{
       private String primerNombre;

       public String PrimerNombre
       {
              get { return primerNombre; }
              set { primerNombre = value; }
       }
       private String segundoNombre;

       public String SegundoNombre
```

```
{
        get { return segundoNombre; }
        set { segundoNombre = value; }
}
private String apellido;

public String Apellido
{
        get { return apellido; }
        set { apellido = value; }
}
private String documento;

public String Documento
{
        get { return documento; }
        set { documento = value; }
}
private String departamento;

public String Departamento
{
        get { return departamento; }
        set { departamento = value; }
}

private String id;

public String Id
{
        get { return id; }
        set { id = value; }
}

private String idJefe;

public String IdJefe
{
        get { return idJefe; }
        set { idJefe = value; }
 }

private float sueldo;

public float Sueldo
{
        get { return sueldo; }
        set { sueldo = value; }
}
private int nroEmpleados;

public int NroEmpleados
{
        get { return nroEmpleados; }
        set { nroEmpleados = value; }
}

public Gerente(String primerNombre, String segundoNombre, String apellido,
```

```
            String documento, String departamento, String id, String ij,
            float sueldo, int nroEmpleados)
    {
            this.primerNombre = primerNombre;
            this.segundoNombre = segundoNombre;
            this.apellido = apellido;
            this.documento = documento;
            this.departamento = departamento;
            this.id = id;
            this.idJefe = ij;
            this.sueldo = sueldo;
            this.nroEmpleados = nroEmpleados;
    }
}
```

VB

```
Public Class Gerente
    Private _primerNombre As String
    Private _segundoNombre As String
    Private _apellido As String
    Private _documento As String
    Private _departamento As String
    Private _id As String
    Private _idJefe As String
    Private _sueldo As Single
    Private _nroEmpleados As Integer

    Public Property PrimerNombre() As String
        Get
            PrimerNombre = _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property

    Public Property SegundoNombre() As String
        Get
            SegundoNombre = _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = value
        End Set
    End Property

    Public Property Apellido() As String
        Get
            Apellido = _apellido
        End Get
        Set(ByVal value As String)
            _apellido = value
        End Set
    End Property

    Public Property Documento() As String
        Get
            Documento = _documento
        End Get
```

```
    Set(ByVal value As String)
        _documento = value
    End Set
End Property

Public Property Departamento() As String
    Get
        Departamento = _departamento
    End Get
    Set(ByVal value As String)
        _departamento = value
    End Set
End Property
Public Property Id() As String
    Get
        Id = _id
    End Get
    Set(ByVal value As String)
        _id = value
    End Set
End Property

Public Property IdJefe() As String
    Get
        IdJefe = _idJefe
    End Get
    Set(ByVal value As String)
        _idJefe = value
    End Set
End Property

Public Property Sueldo() As Single
    Get
        Sueldo = _sueldo
    End Get
    Set(ByVal value As Single)
        _sueldo = value
    End Set
End Property

Public Property NroEmpleados() As Integer
    Get
        NroEmpleados = _nroEmpleados
    End Get
    Set(ByVal value As Integer)
        _nroEmpleados = value
    End Set
End Property

Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String,
          ByVal d As String, ByVal departamento As String, ByVal id As String, _
          ByVal idJefe As String, ByVal sueldo As Single, _
          ByVal nroEmpleados As Integer)
    _primerNombre = p
    _segundoNombre = s
    _apellido = a
    _documento = d
    _departamento = departamento
```

```
        _id = id
        _idJefe = idJefe
        _sueldo = sueldo
        _nroEmpleados = nroEmpleados
    End Sub

End Class
```

Luego de examinar detenidamente el código, se puede apreciar que hay variables y métodos definidos en común para todas las clases.

Con todas estas salvedades, además se debe tener presente que para definir la herencia en .Net se depende del lenguaje utilizado, como se muestra en la siguiente tabla.

C#	<modificador> [Nombre de la Subclase] **:** [Nombre de la Superclase]
VB	<modificador> [Nombre de la Subclase] **: Inherits** [Nombre de la Superclase]

Generalización y especialización

Cuando se diseñan clases se pueden descubrir distintos objetos candidatos en un sistema los cuales pueden agruparse según sus servicios.

Otra forma de agruparlos es a través de evaluar los que tienen similares atributos y operaciones para decidir si son del mismo tipo o no.

En el caso de ser del mismo tipo se puede diseñar con ellos una cadena de herencia en la que participen clases que definan los tipos de los objetos descubiertos.

Cuando se realiza esto, se pueden dar dos casos bien definidos:

- *Dada una clase que define un objeto candidato, se descubre una clase que puede ser superclase de esta. A esto se lo llama **generalización** porque define una clase más genérica que la actual.*
- *Dada una clase que define un objeto candidato, se descubre una clase que puede ser subclase de esta. A esto se lo llama **especialización** porque define una clase más específica que la actual.*

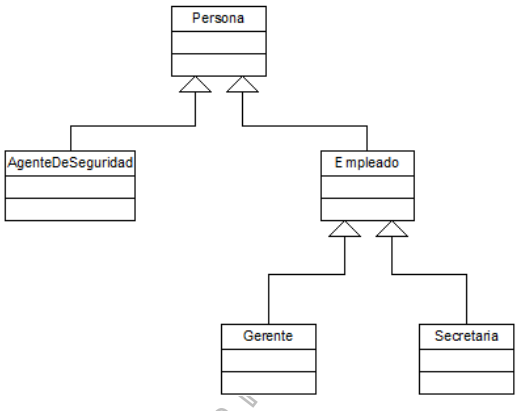
Tanto la generalización como la especialización pueden descubrirse a partir de los objetos candidatos o ser parte de ellos, ya que a través de ellos se descubren las clases que definen sus tipos. El "arte" del diseño consiste en descubrir estas relaciones, definirlas y caracterizarlas por sus tipos y, en caso de no ser parte de los objetos candidatos, definir los tipos que sean necesarios para generalizar o especializar.

Si se tiene en cuenta el código presentado hasta el momento, se puede afirmar que estas variables y métodos deberían pertenecer a una clase de la cual hereden las demás. En una primera instancia se puede seleccionar a la clase `Empleado`, por ejemplo, para poner en ella todos los atributos en común de las clases `Secretaria` y `Gerente`.

Pero se plantea un problema, los agentes de seguridad no son empleados de la empresa, sin embargo tiene atributos en común con las secretarias y los gerentes. Cuando se presenta una situación de este estilo, se debe sospechar que existe una clase más "genérica" que todavía no a sido descubierta.

Refinando nuevamente el diseño se puede otra por crear una clase llamada `Persona`, en la cual se definan atributos en común como `primerNombre`, `segundoNombre`, `apellido` o `documento`. Cuando por un refinamiento de diseño se descubre una superclase a partir de las subclases que se podrían generar a partir ella, se lo denomina "**generalización**".

Para clarificar lo expuesto se puede realizar un gráfico en lenguaje UML y luego presentar el código del formato final de las clases después que se generalizó el concepto a través de Empleado y Persona:



Constructores en la herencia y visibilidades protegidas

Un punto importante a considerar son los elementos privados declarados dentro de cada clase, porque serían inaccesibles en la subclase si estuvieran declarados con dicha visibilidad en la superclase.

Basándose en el mismo argumento, un constructor por más que sea público pertenece a la clase en donde fue declarado (en realidad no tiene sentido pensar en un constructor fuera de la clase donde se declaró porque ¿qué construiría?), por lo tanto cuando se crea una instancia de una subclase, esta debe llamar al constructor de la clase base antes de realizar cualquier otra operación (primero se construye la base y a partir de ella se construye la subclase). Si la clase base, a su vez, es subclase de otra, antes de realizar cualquier operación debe llamar al constructor de "su clase base".

Esta operatoria se propaga a lo largo de la cadena de herencia hasta llegar a la clase base de la cadena de herencia. Una vez terminada la construcción de la clase base de la cadena de herencia, se invierte el proceso construyendo las subclases a partir de ella hasta llegar al constructor de la clase de la cual se desea crear una instancia.

Ambos lenguajes, C# y VB, proveen mecanismos para realizar la invocación del constructor al que se quiere acceder en la clase base. Recordar que una clase puede tener tantos contructores como quiera declarar con la condición que el número o el tipo de los parámetros difieran de constructor a constructor. En C# se utiliza la palabra clave `base` a continuación del operador de herencia (los

dos puntos) y entre paréntesis se colocan los parámetros con los cuales el compilador determina a cuál constructor llamar. En VB se utiliza análogamente, pero como primera instrucción dentro del constructor, `MyBase.New`.

Otro problema que se plantea es que al ser privados los elementos de una clase no se pueden utilizar en otra, inclusive estando en la misma cadena de herencia. La solución que nos proveen los lenguajes es otro modificador de visibilidad: `protected` en C# y `Protected` en VB. Con este modificador se pueden definir elementos dentro de una clase con la siguiente característica:

> *Un elemento definido como `protected` dentro de una clase permanecerá oculto para el mundo exterior (cualquier otra clase cliente que quiera acceder a ese elemento) pero será accesible dentro de su cadena de herencia o el espacio de nombres en que se encuentra definido.*

Esto quiere decir que no se podrá utilizar con notación de punto luego de crear un objeto (salvo, que pertenezcan ambas clases a un mismo espacio de nombres), pero será accesible para otras clases dentro de su cadena de herencia.

Los últimos apartados sirven de fundamento para codificar el nuevo diseño de clases, generalizando cuando es conveniente y llamando al constructor de la clase base.

Ejemplo

Clase Persona

C#

```
class Persona
{
	protected String primerNombre;

	public String PrimerNombre
	{
		get { return primerNombre; }
		set { primerNombre = value; }
	}
	protected String segundoNombre;

	public String SegundoNombre
	{
		get { return segundoNombre; }
		set { segundoNombre = value; }
	}
	protected String apellido;

	public String Apellido
	{
		get { return apellido; }
		set { apellido = value; }
	}
	protected String documento;

	public String Documento
	{
```

```
            get { return documento; }
            set { documento = value; }
        }

        public Persona(String p,
            String s,
            String a,
            String d)
        {
            primerNombre = p;
            segundoNombre = s;
            apellido = a;
            documento = d;
        }
}
```

VB

```
Public Class Persona
    Protected _primerNombre As String
    Protected _segundoNombre As String
    Protected _apellido As String
    Protected _documento As String


    Public Property PrimerNombre() As String
        Get
            PrimerNombre = _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property

    Public Property SegundoNombre() As String
        Get
            SegundoNombre = _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = value
        End Set
    End Property

    Public Property Apellido() As String
        Get
            Apellido = _apellido
        End Get
        Set(ByVal value As String)
            _apellido = value
        End Set
    End Property

    Public Property Documento() As String
        Get
            Documento = _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
```

```
    End Property


    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                   ByVal _documento As String)
        _primerNombre = p
        _segundoNombre = s
        _apellido = a
        _documento = _documento
    End Sub

    Public Overridable Sub Cartel()
        System.Console.WriteLine("Estoy en Persona")
    End Sub

    Public Sub Cartel(ByVal i As Integer)
        System.Console.WriteLine("Estoy en Persona")
        System.Console.WriteLine("Estoy en Persona y el entero es {0}", i)
    End Sub

End Class
```

Clase Empleado

C#

```
class Empleado : Persona
{
        private String departamento;

        public String Departamento
        {
                get { return departamento; }
                set { departamento = value; }
        }
        private String id;

        public String Id
        {
                get { return id; }
                set { id = value; }
        }
        private String idJefe;

        public String IdJefe
        {
                get { return idJefe; }
                set { idJefe = value; }
        }
        private float sueldo;

        public float Sueldo
        {
                get { return sueldo; }
                set { sueldo = value; }
        }

        public Empleado(String p, String s, String a, String doc,
```

```
                String dep, String i, String ij, float sdo)
                : base(p, s, a, doc)
        {
                departamento = dep;
                id = i;
                idJefe = ij;
                sueldo = sdo;
        }
}
```

VB

```
Public Class Empleado : Inherits Persona
    Private _departamento As String
    Private _id As String
    Private _idJefe As String
    Private _sueldo As Single

    Public Property Departamento() As String
        Get
            Departamento = _departamento
        End Get
        Set(ByVal value As String)
            _departamento = value
        End Set
    End Property

    Public Property Id() As String
        Get
            Id = _id
        End Get
        Set(ByVal value As String)
            _id = value
        End Set
    End Property

    Public Property IdJefe() As String
        Get
            IdJefe = _idJefe
        End Get
        Set(ByVal value As String)
            _idJefe = value
        End Set
    End Property

    Public Property Sueldo() As Single
        Get
            Sueldo = _sueldo
        End Get
        Set(ByVal value As Single)
            _sueldo = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, ByVal d
                    As String, ByVal departamento As String, ByVal id As String, _
                    ByVal idJefe As String, ByVal sueldo As Single)
        MyBase.New(p, s, a, d)
         _departamento = departamento
```

```
        Me._id = id
        _idJefe = idJefe
        _sueldo = sueldo
        pepe = 5
    End Sub

    Public Overrides Sub Cartel()
        System.Console.WriteLine("estoy en Empleado")
        System.Console.WriteLine("Me voy a Persona")
        MyBase.Cartel()
    End Sub

End Class
```

Ahora se puede apreciar como se ve reducido el código gracias a la reutilización por medio de la herencia, ya que los elementos que figuran en la superclase no es necesario volver a escribirlos.

Por otro lado, se debe tener en cuenta que una subclase "es del mismo tipo" que la superclase, con lo que se puede utilizar las palabras ***es un*** / ***es una*** para verificarlo.

Esto se evidencia cuando se comprueba con los siguientes enunciados:

- Un empleado ***es una*** persona
- Un gerente ***es un*** empleado
- Una secretaria ***es una*** empleada
- Un agente de seguridad ***es una*** persona

De esta manera las clases quedarían de la siguiente forma:

Clase Gerente

C#

```
class Gerente:Empleado
{
	private int nroEmpleados = 10;

	public int NroEmpleados
	{
		get { return nroEmpleados; }
		set { nroEmpleados = value; }
	}
	public Gerente(
		String p,
		String s,
		String a,
		String doc,
		String dep,
		String i,
		String ij,
		float sdo,
		int ne):base(p, s, a, doc, dep, i, ij, sdo)
	{
		nroEmpleados = ne;
	}
}
```

VB

```
Public Class Gerente : Inherits Empleado
    Private _nroEmpleados As Integer

    Public Property NroEmpleados() As Integer
        Get
            NroEmpleados = _nroEmpleados
        End Get
        Set(ByVal value As Integer)
            _nroEmpleados = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                   ByVal d As String, ByVal departamento As String, _
                   ByVal id As String, ByVal idJefe As String, _
                   ByVal sueldo As Single, ByVal nroEmpleados As Integer)
        MyBase.New(p, s, a, d, departamento, id, idJefe, sueldo)
        _nroEmpleados = nroEmpleados
    End Sub

End Class
```

Clase Secretaria

C#

```
class Secretaria : Empleado
{
       public Secretaria(
              String p,
              String s,
              String a,
              String doc,
              String dep,
              String i,
              String ij,
              float sdo)
              : base(p, s, a, doc, dep, i, ij, sdo)
       {
       }
}
```

VB

```
Public Class Secretaria : Inherits Empleado

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                   ByVal d As String, ByVal departamento As String, _
                   ByVal id As String, _
                   ByVal idJefe As String, ByVal sueldo As Single)
        MyBase.New(p, s, a, d, departamento, id, idJefe, sueldo)
    End Sub

End Class
```

Clase AgenteDeSeguridad

C#

```
class Secretaria : Empleado
{
       public Secretaria(
              String p,
              String s,
              String a,
              String doc,
              String dep,
              String i,
              String ij,
              float sdo)
              : base(p, s, a, doc, dep, i, ij, sdo)
       {
       }
}
```

VB

```
Public Class AgenteDeSeguridad : Inherits Persona
    Private _nroLicencia As String

    Public Property NroLicencia() As String
        Get
            NroLicencia = _nroLicencia
        End Get
        Set(ByVal value As String)
            _nroLicencia = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                   ByVal d As String, ByVal nroLicencia As String)
        MyBase.New(p, s, a, d)
        _nroLicencia = nroLicencia
    End Sub

End Class
```

Nota: En este punto es interesante plantear desde el punto de vista del diseño como se vería afectado el mismo si se incorporan a los agentes de seguridad como empleados de la empresa

Vale la pena hacer hincapié en las clases `Secretaria` y `AgenteDeSeguridad`. Como se puede apreciar en la primera, sólo se invocan los constructores de la superclase, por lo tanto se podría pensar que no es necesario definir una nueva clase de este tipo. Sin embargo, `Secretaria` es una subclase de `Empleado` y debería definirse como tal.

La pregunta en este punto es ¿vale la pena definir una nueva clase? La respuesta a esto no es trivial. En realidad lo que define si un concepto volcado a una clase es significativo tiene que ver más en su importancia como tal para el resto del sistema y su interacción con otros objetos.

Por ejemplo, supongamos que en la actualidad no poseemos información suficiente siquiera para afirmar si el objeto es importante para el sistema, pero sabemos que las definiciones sobre el mismo se ampliarán en un futuro (por ejemplo datos específicos sobre la agencia de seguridad a la que pertenece el agente de seguridad o las funciones administrativas que realiza la secretaria). Si este es el caso, la razón es suficiente como para declarar una clase independiente.

Por otra parte, la clase `AgenteDeSeguridad` no define a un `Empleado`, sin embargo esto es correcto porque la clase se diseñó con la limitación que un agente de seguridad no es empleado de la empresa, por lo tanto, no puede ser del tipo `Empleado`, pero si puede ser del tipo `Persona` y así se lo define.

Ejercicios 1 y 2

Los temas de los que se tratan en este ejercicio son los siguientes:

- Herencia
- Generalización
- Especialización

Aplicar los conocimientos adquiridos para herencia, generalización y especialización codificando diseños del módulo anterior.

Sobrescritura o rescritura de un método

En una cadena de herencia se puede redefinir un método. Esto sucede cuando un método de una subclase se debe comportar diferente de cómo lo hacia en la superclase, es decir, la operación se debe realizar de otra manera.

Para rescribir o sobre escribir un método se debe tener cuidado de escribirlo exactamente igual que en la superclase, esto implica el modificador, el valor retornado, el identificador y la lista de argumentos.

Cuando un método es sobre escrito en una subclase produce un efecto de *ocultar* el método de la clase base cuando se crea una instancia de dicha subclase. De esta manera, cada vez que se invoque el identificador en el código, se hará referencia al escrito en la subclase, pero el método de la superclase sólo esta oculto y mediante ciertas técnicas puede ser invocado. En el caso de C# se utiliza la palabra clave `base` y en el de VB `MyBase`, para acceder a la superclase, ya que estas tienen la función de resolver *visibilidades ocultas* como en este caso. Por lo tanto con estas palabras clave y la notación de punto, se puede invocar inclusive al método sobrescrito dentro del que lo sobrescribe.

En los ejemplos a continuación se pueden ver la sobrescritura en ambas clases y la utilización de la invocación de métodos ocultos utilizando las respectivas palabras clave en cada clase.

Ejemplo

Clase Persona

C#

```
class Persona
{
        protected String primerNombre;

        public String PrimerNombre
        {
                get { return primerNombre; }
                set { primerNombre = value; }
        }
        protected String segundoNombre;

        public String SegundoNombre
        {
                get { return segundoNombre; }
                set { segundoNombre = value; }
        }
        protected String apellido;

        public String Apellido
        {
                get { return apellido; }
                set { apellido = value; }
        }
        protected String documento;

        public String Documento
```

```
        {
                get { return documento; }
                set { documento = value; }
        }

        public Persona(String p,
                String s,
                String a,
                String d)
        {
                primerNombre = p;
                segundoNombre = s;
                apellido = a;
                documento = d;
        }

        public virtual void cartel()
        {
                System.Console.WriteLine("Estoy en Persona");
        }
}
```

VB

```
Public Class Persona
    Protected _primerNombre As String
    Protected _segundoNombre As String
    Protected _apellido As String
    Protected _documento As String


    Public Property PrimerNombre() As String
        Get
            PrimerNombre = _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property

    Public Property SegundoNombre() As String
        Get
            SegundoNombre = _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = value
        End Set
    End Property

    Public Property Apellido() As String
        Get
            Apellido = _apellido
        End Get
        Set(ByVal value As String)
            _apellido = value
        End Set
    End Property

    Public Property Documento() As String
```

```
        Get
            Documento = _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
    End Property


    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                   ByVal _documento As String)
        _primerNombre = p
        _segundoNombre = s
        _apellido = a
        _documento = _documento
    End Sub

    Public Overridable Sub Cartel()
        System.Console.WriteLine("Estoy en Persona")
    End Sub

End Class
```

Clase Empleado

C#

```
class Empleado : Persona
{
        private String departamento;

        public String Departamento
        {
                get { return departamento; }
                set { departamento = value; }
        }
        private String id;

        public String Id
        {
                get { return id; }
                set { id = value; }
        }
        private String idJefe;

        public String IdJefe
        {
                get { return idJefe; }
                set { idJefe = value; }
        }
        private float sueldo;

        public float Sueldo
        {
                get { return sueldo; }
                set { sueldo = value; }
        }
```

```
        public Empleado(String p, String s, String a, String doc,
               String dep, String i, String ij, float sdo)
               : base(p, s, a, doc)
        {
               departamento = dep;
               id = i;
               idJefe = ij;
               sueldo = sdo;
        }

        public override void cartel()
        {
               System.Console.WriteLine("Estoy en Empleado");
               System.Console.WriteLine("Me voy a Persona");
               base.cartel();
        }
}
```

VB

```
Public Class Empleado : Inherits Persona
    Private _departamento As String
    Private _id As String
    Private _idJefe As String
    Private _sueldo As Single

    Public Property Departamento() As String
        Get
            Departamento = _departamento
        End Get
        Set(ByVal value As String)
            _departamento = value
        End Set
    End Property
    Public Property Id() As String
        Get
            Id = _id
        End Get
        Set(ByVal value As String)
            _id = value
        End Set
    End Property

    Public Property IdJefe() As String
        Get
            IdJefe = _idJefe
        End Get
        Set(ByVal value As String)
            _idJefe = value
        End Set
    End Property

    Public Property Sueldo() As Single
        Get
            Sueldo = _sueldo
        End Get
        Set(ByVal value As Single)
            _sueldo = value
        End Set
```

```
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, ByVal d
As String, _
                   ByVal departamento As String, ByVal id As String, ByVal idJefe As
String, _
                   ByVal sueldo As Single)
        MyBase.New(p, s, a, d)
         _departamento = departamento
        Me._id = id
        _idJefe = idJefe
        _sueldo = sueldo
    End Sub

    Public Overrides Sub Cartel()
        System.Console.WriteLine("Estoy en Empleado")
        System.Console.WriteLine("Me voy a Persona")
        MyBase.Cartel()
    End Sub

End Class
```

Clase Gerente

C#

```
class Gerente:Empleado
{
       private int nroEmpleados = 10;

       public int NroEmpleados
       {
              get { return nroEmpleados; }
              set { nroEmpleados = value; }
       }
       public Gerente(
              String p,
              String s,
              String a,
              String doc,
              String dep,
              String i,
              String ij,
              float sdo,
              int ne):base(p, s, a, doc, dep, i, ij, sdo)
       {
              nroEmpleados = ne;
       }

       public override void cartel()
       {
              System.Console.WriteLine("Estoy en Gerente");
              System.Console.WriteLine("Me voy a Empleado");
              base.cartel();
       }
}
```

VB

```
Public Class Gerente : Inherits Empleado
    Private _nroEmpleados As Integer

    Public Property NroEmpleados() As Integer
        Get
            NroEmpleados = _nroEmpleados
        End Get
        Set(ByVal value As Integer)
            _nroEmpleados = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, ByVal d
As String, _
                   ByVal departamento As String, ByVal id As String, ByVal idJefe As
String, _
                   ByVal sueldo As Single, ByVal nroEmpleados As Integer)
        MyBase.New(p, s, a, d, departamento, id, idJefe, sueldo)
        _nroEmpleados = nroEmpleados
    End Sub

    Public Overrides Sub Cartel()
        System.Console.WriteLine("Estoy en Gerente")
        System.Console.WriteLine("Me voy a Empleado")
        MyBase.Cartel()
    End Sub

End Class
```


Sobrecarga de métodos

Existen situaciones en que un método se debe llamar igual a otro u otros por el tipo de servicio que brinda la operación que realiza. En estos casos, el lenguaje permite que dos o más métodos se llamen igual con tal que cumpla una o ambas de siguientes condiciones:

- La cantidad de argumentos es diferente
- El tipo de los argumentos es diferente

Se dice que un método esta sobrecargado cuando cumple con estas condiciones. Por ejemplo, se puede tener métodos donde cambie sólo el tipo de argumento:

```
Console.WriteLine()
Console.WriteLine(Boolean)
Console.WriteLine(Char)
Console.WriteLine(Char[])
Console.WriteLine(Decimal)
Console.WriteLine(Double)
Console.WriteLine(Int32)
Console.WriteLine(Int64)
Console.WriteLine(Object)
Console.WriteLine(Single)
Console.WriteLine(String)
Console.WriteLine(UInt32)
Console.WriteLine(UInt64)
Console.WriteLine(String, Object)
```

```
Console.WriteLine(String, Object[])
Console.WriteLine(Char[], Int32, Int32)
Console.WriteLine(String, Object, Object)
Console.WriteLine(String, Object, Object, Object)
Console.WriteLine(String, Object, Object, Object, Object)
```

O métodos donde cambia la cantidad de argumentos, como por ejemplo

C#

```
public void CambiaDireccion(bool esA, String nomCalle,
       int n, bool esDep, int p, String d)
public void CambiaDireccion(int n, bool esDep, int p, String d)
public void CambiaDireccion(int p, String d)
```

VB

```
Public Sub CambiaDireccion(esA As Boolean, nomCalle As String, _
              n As Integer, esDep As Boolean, p As Integer, d As String)
Public Sub CambiaDireccion(n As Integer, esDep As Boolean, p As Integer, _
              d As String)
Public Sub CambiaDireccion(p As Integer, d As String)
```

O una combinación de ambos:

C#

```
public void Dibuja(int a)
public void Dibuja(int a, int b)
public void Dibuja(char a)
public void Dibuja(int a, char b)
```

VB

```
Public Sub Dibuja(a As Integer)
Public Sub Dibuja(a As Integer, b As Integer)
Public Sub Dibuja(a As Char)
Public Sub Dibuja(a As Integer, b As Char)
```

La forma como los lenguajes diferencian los métodos es a través de la llamada **firma** del mismo, que compone la unión del nombre del método (identificador) y los tipos de datos declarados como argumentos, con lo cual, el lenguaje interpreta internamente los nombres de los métodos, por ejemplo en C#, de la siguiente manera:

```
public void Dibujaint(int a)
public void Dibujaintint(int a, int b)
public void Dibujachar(char a)
public void Dibujaintchar(int a, char b)
```

Como se puede ver, internamente todos los nombres son diferentes. Vale la pena destacar que nada se dijo del valor retornado y el motivo es que este no diferencia a un método de otro porque no es parte de la firma del mismo.

Otra aclaración importante que podemos hacer en este punto es que los constructores se pueden sobrecargar, y eso es lo que habilita a declarar más de uno en una clase, pero tiene las mismas

restricciones que cualquier método sobrecargado, es más, esto se nota mejor porque los constructores no devuelven ningún valor.

Si se retoman ejemplos anteriores, se puede suponer que luego de refinar nuevamente el diseño se llega a la conclusión que es mucho más fácil tener un método que permita cambiar la dirección y dejar las propiedades sólo para cambios puntuales. Los casos a tener en cuenta para este refinamiento son:

- Cambio completo de dirección
- Cambio de dirección pero se mantiene en la misma calle o avenida
- Cambio de dirección pero se mantiene en el mismo edificio

El grupo de métodos expuesto anteriormente que se llama `CambiaDireccion` cumple con estas condiciones. Con lo que la clase `Direccion` y una que la utilice, quedaría de la siguiente forma.

Ejemplo

Clase Direccion

C#

```
class Direccion
{
	private bool esAvenida;	// calle común o avenida

	public bool EsAvenida
	{
		get { return esAvenida; }
		set { esAvenida = value; }
	}
	private String nombreCalle;// nombre completo de la calle

	public String NombreCalle
	{
		get { return nombreCalle; }
		set { nombreCalle = value; }
	}
	private int nro;			// nro de la casa o departamento

	public int Nro
	{
		get { return nro; }
		set { nro = value; }
	}
	private bool esDepartamento;// es departamento o no

	public bool EsDepartamento
	{
		get { return esDepartamento; }
		set { esDepartamento = value; }
	}
	private int piso;			// número del piso

	public int Piso
```

```
{
        get { return piso; }
        set { piso = value; }
}
private String dpto;        // departamento del piso

public String Dpto
{
        get { return dpto; }
        set { dpto = value; }
}

public Direccion(
        String nombreCalle, // nombre completo de la calle
        int numero,         // nro de la casa o departamento
        bool esA,           // calle común o avenida
        String nomCalle,    // nombre completo de la calle
        int n,              // nro de la casa o departamento
        bool esDep,         // es departamento o no
        int p,              // número del piso
        String d)           // departamento del piso
{
        esAvenida = esA;
        nombreCalle = nomCalle;
        nro = n;
        esDepartamento = esDep;
        piso = p;
        dpto = d;
}

public Direccion(
        bool esA,           // calle común o avenida
        String nomCalle,    // nombre completo de la calle
        int n,              // nro de la casa o departamento
        bool esDep,         // es departamento o no
        int p,              // número del piso
        String d)           // departamento del piso
{
        esAvenida = esA;
        nombreCalle = nomCalle;
        nro = n;
        esDepartamento = esDep;
        piso = p;
        dpto = d;
}

public void CambiaDireccion(
        bool esA,           // calle común o avenida
        String nomCalle,    // nombre completo de la calle
        int n,              // nro de la casa o departamento
        bool esDep,         // es departamento o no
        int p,              // número del piso
        String d)           // departamento del piso
{
        esAvenida = esA;
        nombreCalle = nomCalle;
        nro = n;
```

```
            esDepartamento = esDep;
            piso = p;
            dpto = d;
        }
        public void CambiaDireccion(
            int n,              // nro de la casa o departamento
            bool esDep,         // es departamento o no
            int p,              // número del piso
            String d)           // departamento del piso
        {
            nro = n;
            esDepartamento = esDep;
            piso = p;
            dpto = d;
        }
        public void CambiaDireccion(
            int p,              // número del piso
            String d)           // departamento del piso
        {
            piso = p;
            dpto = d;
        }
}
```

VB

```
Public Class Direccion
    Private _nombreCalle As String
    Private _nro As Integer
    Private _esAvenida As Boolean
    Private _esDepartamento As Boolean
    Private _piso As Integer
    Private _dpto As String

    Public Property NombreCalle() As String
        Get
            NombreCalle = _nombreCalle
        End Get
        Set(ByVal value As String)
            _nombreCalle = value
        End Set
    End Property

    Public Property Nro() As String
        Get
            Nro = _nro
        End Get
        Set(ByVal value As String)
            _nro = value
        End Set
    End Property

    Public Property EsAvenida() As Boolean
        Get
            Return _esAvenida
        End Get
        Set(ByVal value As Boolean)
            _esAvenida = value
        End Set
```

```
End Property

Public Property EsDepartamento() As Boolean
    Get
        Return _esDepartamento
    End Get
    Set(ByVal value As Boolean)
        _esDepartamento = value
    End Set
End Property

Public Property Piso() As Integer
    Get
        Return _piso
    End Get
    Set(ByVal value As Integer)
        _piso = value
    End Set
End Property

Public Property Dpto() As String
    Get
        Return _dpto
    End Get
    Set(ByVal value As String)
        _dpto = value
    End Set
End Property

Public Sub New(ByVal nombreCalle As String, ByVal nro As Integer, _
   esA As Boolean, nomCalle As String, n As Integer, esDep As Boolean, _
   p As Integer, d As String)
    _nombreCalle = nombreCalle
    _nro = nro
    _esAvenida = esA
    _nombreCalle = nomCalle
    _nro = n
    _esDepartamento = esDep
    _piso = p
    _dpto = d
End Sub


Public Sub New(esA As Boolean, nomCalle As String, n As Integer, _
               esDep As Boolean, p As Integer, d As String)
    _esAvenida = esA
    _nombreCalle = nomCalle
    _nro = n
    _esDepartamento = esDep
    _piso = p
    _dpto = d
End Sub

Public Sub CambiaDireccion(esA As Boolean, nomCalle As String, _
                  n As Integer, esDep As Boolean, p As Integer, d As String)
    _esAvenida = esA
    _nombreCalle = nomCalle
    _nro = n
```

```
        _esDepartamento = esDep
        _piso = p
        _dpto = d
    End Sub

    Public Sub CambiaDireccion(n As Integer, esDep As Boolean, p As Integer, _
              d As String)
        _nro = n
        _esDepartamento = esDep
        _piso = p
        _dpto = d
    End Sub

    Public Sub CambiaDireccion(p As Integer, d As String)
        _piso = p
        _dpto = d
    End Sub
End Class
```

Clase UsaSobrecarga

C#

```
class UsaSobrecarga
{
       static void Main(string[] args)
       {
              Direccion d = new Direccion(true, "Juan De Garay", 870, true, 1, "A");
              Gerente g = new Gerente("Juan", "Pedro", "Goyena", "17.767.076", d,
                           "Ventas", "0", "GV50", 5500.0F, 20);
              Console.WriteLine("Apellido: " + g.Apellido);
              Console.WriteLine("Primer Nombre: " + g.PrimerNombre);
              Console.WriteLine("Segundo Nombre: " + g.SegundoNombre);
              Console.WriteLine("Departamento: " + g.Departamento);
              Console.WriteLine("Direccion: " + g.ObjDireccion.NombreCalle + " " +
                     g.ObjDireccion.Nro + " " + g.ObjDireccion.Piso.ToString() + "° "
                     + g.ObjDireccion.Dpto);

              Console.WriteLine();

              g.ObjDireccion.CambiaDireccion(false, "Carlos Calvo", 2222, false, 0,
                     " ");
              Console.WriteLine("Direccion: " + g.ObjDireccion.NombreCalle + " " +
                     g.ObjDireccion.Nro + " " + g.ObjDireccion.Piso.ToString() + "° "
                     + g.ObjDireccion.Dpto);

              Console.WriteLine();

              g.ObjDireccion.CambiaDireccion(245, true, 2, "C");
              Console.WriteLine("Direccion: " + g.ObjDireccion.NombreCalle + " " +
                     g.ObjDireccion.Nro + " " + g.ObjDireccion.Piso.ToString() + "° "
                     + g.ObjDireccion.Dpto);

              Console.WriteLine();

              g.ObjDireccion.CambiaDireccion(10, "B");
              Console.WriteLine("Direccion: " + g.ObjDireccion.NombreCalle + " " +
                     g.ObjDireccion.Nro + " " + g.ObjDireccion.Piso.ToString() + "° "
```

```
                    + g.ObjDireccion.Dpto);

            Console.ReadKey();
        }
}
```

Module1

VB

```
Module Module1

    Sub Main()
        Dim d As New Direccion(True, "Juan De Garay", 870, True, 1, "A")
        Dim g As Gerente = New Gerente("Juan", "Pedro", "Goyena", "17.767.076", d, _
                                       "Ventas", "0", "GV50", 5500.0F, 20)
        Console.WriteLine("Apellido: " + g.Apellido)
        Console.WriteLine("Primer Nombre: " + g.PrimerNombre)
        Console.WriteLine("Segundo Nombre: " + g.SegundoNombre)
        Console.WriteLine("Departamento: " + g.Departamento)
        Console.WriteLine("Direccion: " + g.Direccion.NombreCalle + " " + _
            g.Direccion.Nro + " " + g.Direccion.Piso.ToString() + "° " + _
            g.Direccion.Dpto)

        Console.WriteLine()

        g.Direccion.CambiaDireccion(False, "Carlos Calvo", 2222, False, 0, " ")
        Console.WriteLine("Direccion: " + g.Direccion.NombreCalle + " " +
            g.Direccion.Nro + " " + g.Direccion.Piso.ToString() + "° " +
            g.Direccion.Dpto)

        Console.WriteLine()

        g.Direccion.CambiaDireccion(245, True, 2, "C")
        Console.WriteLine("Direccion: " + g.Direccion.NombreCalle + " " +
            g.Direccion.Nro + " " + g.Direccion.Piso.ToString() + "° " +
            g.Direccion.Dpto)

        Console.WriteLine()

        g.Direccion.CambiaDireccion(10, "B")
        Console.WriteLine("Direccion: " + g.Direccion.NombreCalle + " " +
            g.Direccion.Nro + " " + g.Direccion.Piso.ToString() + "° " +
            g.Direccion.Dpto)

        Console.ReadKey()

    End Sub

End Module
```


Clases abstractas

Cuando se diseña una clase se puede presentar el caso de conocer un servicio que se deba realizar, pero los detalles de su implementación se delegan a las subclases de ella. A nivel de código esto se puede trabajar a través de las clases abstractas.

Se debe tener presente que al no saber como escribir el método, propiedad o evento, se lo declara pero no se le asigna un bloque de sentencias. Esto lo convierte en una suerte de método, propiedad o evento incompleto y no se puede crear una instancia de una clase que tenga un método incompleto, propiedad o evento.

Una clase es abstracta cuando se declara como tal y tiene al menos una operación, propiedad o evento no definido que se declara como abstracto.

Clase Persona

C#

```
abstract class Persona
{
	private String primerNombre;

	public String PrimerNombre
	{
		get { return primerNombre; }
		set { primerNombre = value; }
	}
	private String segundoNombre;

	public String SegundoNombre
	{
		get { return segundoNombre; }
		set { segundoNombre = value; }
	}
	private String apellido;

	public String Apellido
	{
		get { return apellido; }
		set { apellido = value; }
	}
	private String documento;

	public String Documento
	{
		get { return documento; }
		set { documento = value; }
	}
	private Direccion objDireccion;

	public Direccion ObjDireccion
	{
		get { return objDireccion; }
		set { objDireccion = value; }
	}

	public Persona(String p,
		String s,
		String a,
		String d, Direccion objD)
	{
		primerNombre = p;
```

```
            segundoNombre = s;
            apellido = a;
            documento = d;
            objDireccion = objD;
        }

        private String detalles;
        public abstract String Detalles
        {
            get;
            set;
        }

        protected virtual string armaDetalles()
        {
            detalles = "Apellido: " + Apellido + " Primer Nombre: " + PrimerNombre
            + " Segundo Nombre: " + SegundoNombre + " Documento: " + Documento
            + " Direccion: " + objDireccion.NombreCalle + " " +
            objDireccion.Nro.ToString();
            return detalles;
        }
}
```

VB

```
Public MustInherit Class Persona
    Private _primerNombre As String
    Private _segundoNombre As String
    Private _apellido As String
    Private _documento As String
    Private dir As Direccion
    Private _detalles As String

    Public Property Direccion() As Direccion
        Get
            Return dir
        End Get
        Set(ByVal value As Direccion)
            dir = value
        End Set
    End Property

    Public Property PrimerNombre() As String
        Get
            PrimerNombre = _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property

    Public Property SegundoNombre() As String
        Get
            SegundoNombre = _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = value
        End Set
    End Property
```

```
    Public Property Apellido() As String
        Get
            Apellido = _apellido
        End Get
        Set(ByVal value As String)
            _apellido = value
        End Set
    End Property

    Public Property Documento() As String
        Get
            Documento = _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                   ByVal _documento As String, ByVal dir As Direccion)
        _primerNombre = p
        _segundoNombre = s
        _apellido = a
        _documento = _documento
        Me.dir = dir
    End Sub

    Public MustOverride ReadOnly Property Detalles As String

    Protected Overridable Function armaDetalles() as string
        _detalles = "Apellido: " + Apellido + " Primer Nombre: " + PrimerNombre + _
        " Segundo Nombre: " + SegundoNombre + " Documento: " + Documento + _
        " Direccion: " + dir.NombreCalle + " " + dir.Nro
        Return _detalles
    End Function
End Class
```

Ejercicio 3

Los temas de los que se tratan en este ejercicio son los siguientes:

- Clases Abstractas
- Sobrecarga
- Sobrescritura

Aplicar los conocimientos adquiridos para clases abstractas, sobrecarga y rescritura codificando diseños del módulo anterior.

Polimorfismo

Para comprender el funcionamiento del polimorfismo se debe pensar en como se realizan conversiones de tipo en las referencias. Al igual que en una expresión matemática se pueden sumar diferentes tipos y los lenguajes "promueven" los valores a los tipos que puedan albergar cada operación (como por ejemplo, si se suma un entero con un punto flotante, toda la operación se realiza como punto flotante porque se promueve el entero a este tipo), los tipos referenciados pueden promoverse en una asignación de valores a un tipo capaz de albergar dichos valores, sin perder su identidad. Para aclarar esto un poco, se puede definir:

> *Se puede declarar una variable de referencia al tipo de una clase que pertenezca a una cadena o sub cadena de herencia y almacenar en ella cualquier referencia a un objeto de una subclase de la cadena o sub cadena de herencia.*

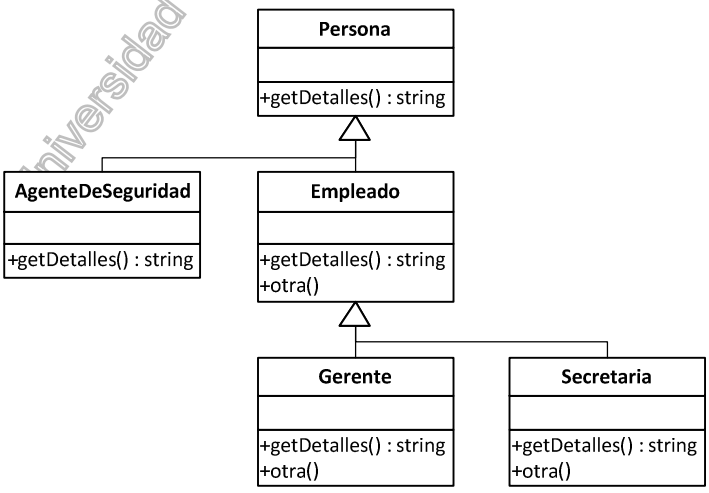
Esto quiere decir que basta con que una clase sea base de una sub cadena de herencia para que se pueda declarar una referencia a cualquier subclase a partir de ella. Sin embargo se destaca el hecho que sigue siendo del tipo de la subclase.

> *En las conversiones de tipo se afecta a lo que puede acceder la referencia (visibilidad). Los objetos por más que sus referencias hayan sido convertidas* ***no pierden su identidad****.*

Para entender profundamente lo enunciado se puede evocar al ejemplo de las clases Persona y Empleado. Si una referencia del tipo Empleado se asigna a una del tipo Persona, lo único que cambia es que a partir de la conversión, sólo se podrán acceder a los miembros visibles (no a los privados) que Persona y Empleado tengan en común, pero **el objeto en si mismo sigue siendo del tipo Empleado**, esto es, no cambia.

Analizando las conversiones de tipo

Tomando nuevamente las clases de ejemplo y considerando el diagrama que se muestra a continuación, se va a analizar el efecto de las conversiones de tipo



Si en el código se realiza la siguiente declaración

C#

```
Persona refPolimorfica;
```

VB

```
Dim refPolimorfica As Persona
```

Todas las clases que figuran en el diagrama, teniendo en cuenta la posibilidad de convertir tipos referenciados como se mencionó anteriormente, se pueden asignar a la variable de referencia al tipo de la superclase de la cadena de herencia, un objeto de cualquiera de los subtipos que figuran en el diagrama, por ejemplo, suponiendo las siguientes declaraciones

C#

```
Direccion d = new Direccion(true, "Juan De Garay", 870, true, 1, "A");

d = null;
d = new Direccion(false, "Amenabar", 1570, true, 7, "H");
Empleado e = new Empleado("Sergio", "Omar", "Perotti","20.200.555",d,
                                    "Administración","EA01","GAD50",1500.0F);
d = null;
d = new Direccion(true, "Allen", 1070, true, 5, "J");
Secretaria s = new Secretaria("Laura", "Verónica", "Diaz","20.555.555",d,
                                    "Directorio","SV01","GV50",1800.0F);
d = null;
d = new Direccion(false, "Juramento", 4561, false, 0, " ");
Gerente g = new Gerente("Juan", "Pedro", "Goyena","17.767.076",d,
                                    "Ventas","0","GV50", 5500.0F, 20);
d = null;
d = new Direccion(true, "Belgrano", 4560, true, 3, "C");
AgenteDeSeguridad a = new AgenteDeSeguridad("Hernán", "Adrián",
                                    "Perez","20.888.999", d, "207272");
```

VB

```
Dim d As Direccion = New Direccion("Juan De Garay", 870)

d = Nothing
d = New Direccion("Amenabar", 1570)
Dim e As Empleado = New Empleado("Sergio", "Omar","Perotti","20.200.555",d, _
       "Administración", "EA01", "GAD50", 1500.0F)
d = Nothing
d = New Direccion("Allen", 1070)
Dim s As Secretaria = New Secretaria("Laura", "Verónica", "Diaz", _
       "20.555.555", d, "Directorio", "SV01", "GV50", 1800.0F)
d = Nothing
d = New Direccion("Juramento", 333)
Dim g As Gerente = New Gerente("Juan", "Pedro", "Goyena", "17.767.076", d, _
       "Ventas", "0", "GV50", 5500.0F, 20)
d = Nothing
d = New Direccion("Belgrano", 444)
Dim a As AgenteDeSeguridad = New AgenteDeSeguridad("Hernán", "Adrián", _
       "Perez", "20.888.999", d, "207272")
```

Se pueden realizar las siguientes asignaciones

C#

```
refPolimorfica = e;
refPolimorfica = s;
refPolimorfica = g;
refPolimorfica = a;
```

VB

```
refPolimorfica = e
refPolimorfica = s
refPolimorfica = g
refPolimorfica = a
```

Todas las asignaciones son correctas porque cada una de las referencias apunta a objetos que son del tipo de alguna de las subclases de Persona.

Ahora bien, el diagrama sólo muestra un método en común entre todas las clases, por lo tanto es lógico pensar que es el único servicio que se quiere analizar. Con esto presente, lo único que tiene visibilidad oculta por rescritura en las diferentes clases es el método `getDetalles()`. Por lo tanto, es el único elemento de la clase que se puede invocar porque sólo esta operación es común a todas las demás (excepto por el resto de los elementos que puedan definirse en persona que no se analizan en este momento).

Para invocar un servicio, este debe ser visible o accesible. Como sólo puede acceder a los elementos definidos en Persona, si se trata de acceder a un método llamado `Otro()` que, por ejemplo, existe en una clase de la cadena de herencia como Empleado, **pero no esta declado en** Persona, luego de asignarle a `refPolimorfica` la referencia al objeto del tipo Empleado, dará un error ya que este no es accesible. El motivo es que `Otro()` no existe como servicio de Persona y una referencia a este tipo, simplemente no tiene definido como acceder a una operación que no esta declarada en esa clase o en alguna que sea superclase de ella y que además sea visible.

Por lo tanto, el objeto sigue siendo del tipo Empleado, pero debido a la conversión de tipo sólo puede acceder los mismos elemento que puede acceder un objeto del tipo Persona.

C#

```
refPolimorfica = e;
refPolimorfica.Detalles; // Correcto!!!!
refPolimorfica.otra(); // El método otra no es visible en este punto
```

VB

```
refPolimorfica = e
refPolimorfica.Detalles ' Correcto!!!!
refPolimorfica.otra()' El método otra no es visible en este punto
```

Para recuperar el acceso a los elementos declarados en Empleado, hay que revertir la conversión de tipo, de la siguiente manera

C#

```
refPolimorfica = e;
Empleado e2 = e;
```

VB

```
refPolimorfica = e
```

```
Dim e2 As Empleado = refPolimorfica
```

Como la conversión desde un tipo a un subtipo no esta definida, se la debe forzar con una conversión de tipo explícita, como se hace en (Empleado). Las conversiones explícitas se deben realizar con mucho cuidado para no convertir a un tipo diferente al que se fuerza.

Luego de la asignación, e es una referencia a un objeto del tipo Empleado y por lo tanto tiene acceso a todos sus miembros públicos (o protegidos si están en el mismo espacio de nombres). Por lo tanto no causa más un error la invocación siguiente

C#

```
e.Otro(); // En este punto Otro es visible
```

VB

```
e2.Otro() 'En este punto Otro es visible
```

Es claro que las conversiones de tipo funcionan de la misma manera cuando se llevan a cabo en un pasaje de parámetros en el llamado de una función. De esta manera si se tiene el siguiente código, se aplican los mismos conceptos

C#

```
public void muestraDetalles(Persona refPolimorfica){
        Console.WriteLine();
        Console.WriteLine("-----------------");
        Console.WriteLine(refPolimorfica.getDetalles());
        Console.WriteLine("-----------------");
        Console.ReadKey();
}
```

VB

```
Public Sub muestraDetalles(ByVal refPolimorfica As Persona)
        Console.WriteLine()
        Console.WriteLine("-----------------")
        Console.WriteLine(refPolimorfica.getDetalles())
        Console.WriteLine("-----------------")
        Console.ReadKey()
End Sub
```

Cuando se invoque a este método, siempre se va a llamar al método `getDetalles()` que pertenezca al objeto que se le pase como parámetro por lo antes expuesto.

Interfaces

La forma más fácil de entender a una interfaz es pensarla como una clase abstracta pura, es decir, una clase cuyos servicios son todos abstractos.

Una interfaz pública es un convenio (contrato) entre la clase que la implementa, la cual deberá escribir el código que se ejecutará en la función cuyo prototipo se declara en la interfaz. Esta obligación de la clase concreta para rescribir los métodos declarados en la interfaz es la parte en la que se suele decir que “cumple con el contrato”

La interfaz de .Net es la declaración formal del contrato, de manera que en ella sólo se encuentran los prototipos de los métodos a implementar en la clase

Muchas clases no relacionadas pueden implementar una interfaz. Una clase puede implementar muchas interfaces

La sintaxis es:

```
< declaración_clase> ::=< modificador> clase < nombre>
        [extends < superclase>]
        [implements < interfaz> [,< interfaz>]* ] {
                < declaraciones>*
}
```

Una interfaz no se diseña teniendo en mente una sola clase. Como cualquier clase puede implementar una de ellas, la idea es poner dentro métodos que se esperan que una o más clases los implementen porque tienen funcionalidad en común

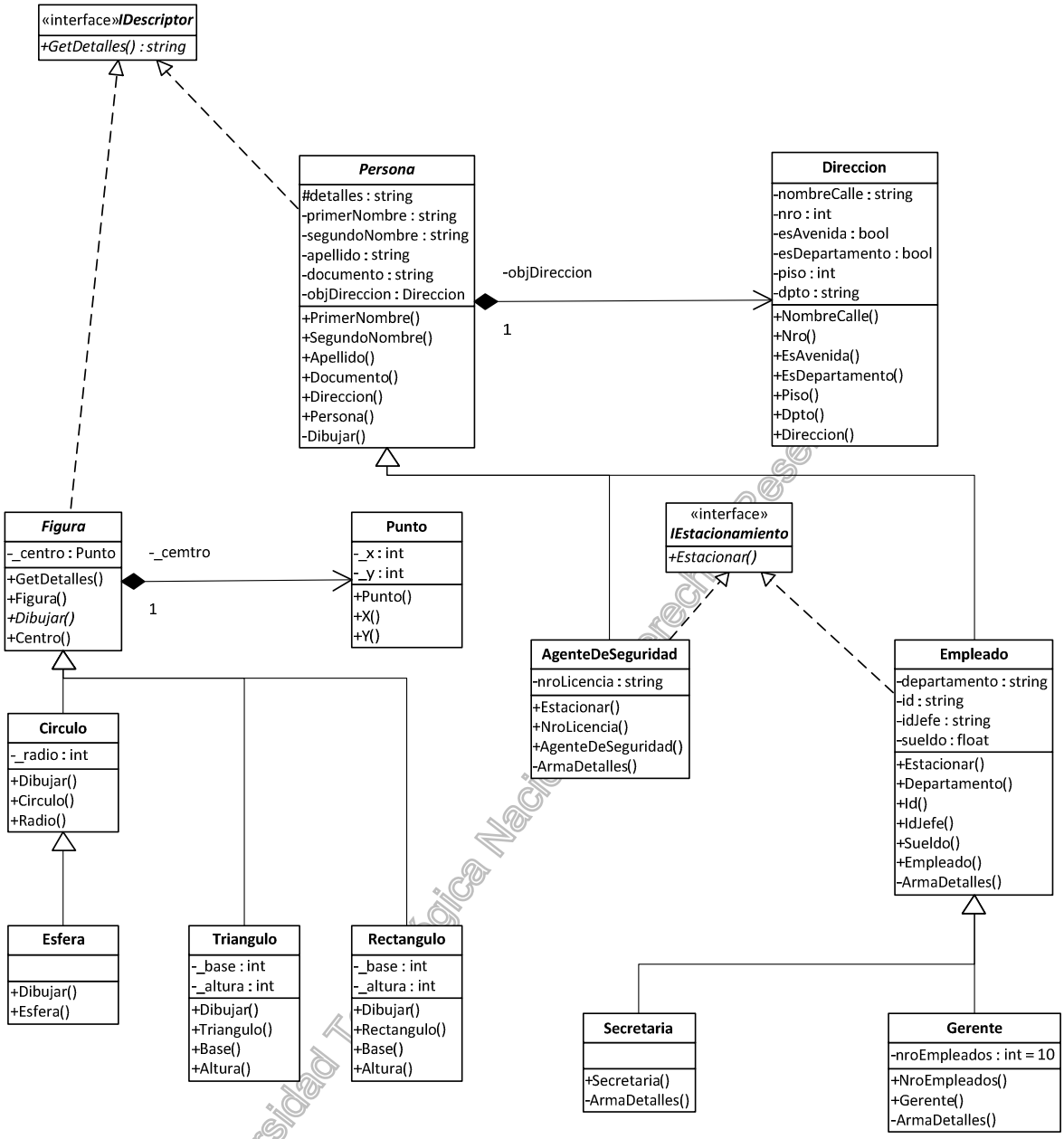
Otra gran ventaja es que conociendo sólo la interfaz y los servicios declarados en ellas, es suficiente como para declarar una referencia a esta (recordar que no se puede crear una instancia de ningún tipo de clase que tenga un elemento abstracto – esto es como tener métodos incompletos – pero si se pueden crear referencias a ellas porque apuntan a objetos **de ese tipo** y una clase que implementa la interfaz **es del tipo de la interfaz también**).

Una referencia a una interfaz puede almacenar una referencia a cualquier objeto que la implemente gracias a las conversiones de tipo. Por lo tanto, se puede usar la funcionalidad definida en la interfaz para una determinada clase sin conocer todos sus métodos accesibles o los detalles de su implementación, sólo conociendo los que están declarados en ella.

Como diferentes tipos de clases pueden implementar la misma interfaz (porque esto depende más de los servicios que están declarados dentro de ellas que de una abstracción en particular), capturan similitudes entre clases no relacionadas sin forzar una herencia concreta entre ellas

Simula la herencia múltiple porque se pueden implementar muchas interfaces en distintas clases concretas, con lo cual no está mal afirmar que es una herencia simple de clases concretas pero una herencia múltiple de clases abstractas puras.

El siguiente gráfico UML muestra lo expuesto



Sin embargo, como la clase Persona es abstracta, le permite implementar el método de la interfaz como abstracto y no afecta en nada a su comportamiento. Este no es el caso de las subclases que al ser concretas están obligadas a cumplir el contrato, por lo tanto, al menos Empleado y AgenteDeSeguridad que tienen como superclase a Persona deben rescribir obligatoriamente el método. A partir de este punto se comporta como un ejemplo de polimorfismo normal con rescritura

Un extracto del código utilizado en el programa que accede a dichas las clases diagramadas se puede ver a continuación

C#

```
static void Main(string[] args)
{
        UsaInterfaz up = new UsaInterfaz();
        IDescriptor refPolimorfica;
        Figura f;

        Direccion d = new Direccion(true, "Juan De Garay", 870, true, 1, "A");

        d = null;
        d = new Direccion(false, "Amenabar", 1570, true, 7, "H");
        Empleado e = new Empleado("Sergio", "Omar", "Perotti","20.200.555",d,
                                  "Administración","EA01","GAD50",1500.0F);
        d = null;
        d = new Direccion(true, "Allen", 1070, true, 5, "J");
        Secretaria s = new Secretaria("Laura", "Verónica", "Diaz","20.555.555",d,
                                  "Directorio","SV01","GV50",1800.0F);
        d = null;
        d = new Direccion(false, "Juramento", 4561, false, 0, " ");
        Gerente g = new Gerente("Juan", "Pedro", "Goyena","17.767.076",d,
                                  "Ventas","0","GV50",5500.0F,20);
        d = null;
        d = new Direccion(true, "Belgrano", 4560, true, 3, "C");
        AgenteDeSeguridad a = new AgenteDeSeguridad("Hernán", "Adrián",
                                  "Perez","20.888.999",d, "207272");

        refPolimorfica = e;
        Console.WriteLine(refPolimorfica.GetDetalles());
        Console.WriteLine();
        refPolimorfica = s;
        Console.WriteLine(refPolimorfica.GetDetalles());
        Console.WriteLine();
        refPolimorfica = g;
        Console.WriteLine(refPolimorfica.GetDetalles());
        Console.WriteLine();
        refPolimorfica = a;
        Console.WriteLine(refPolimorfica.GetDetalles());
        Console.WriteLine();

        Circulo c = new Circulo(3, 6, 1);
        refPolimorfica = c;
        Console.WriteLine(refPolimorfica.GetDetalles());
        f = c;
        f.Dibujar();
        Console.WriteLine();
        Rectangulo r = new Rectangulo(4, 7, 2, 3);
        refPolimorfica = r;
        Console.WriteLine(refPolimorfica.GetDetalles());
        f = r;
        f.Dibujar();
        Console.WriteLine();
        Triangulo t = new Triangulo(5, 8, 2, 3);
        refPolimorfica = t;
        Console.WriteLine(refPolimorfica.GetDetalles());
        f = t;
        f.Dibujar();
        Console.WriteLine();
        Esfera es = new Esfera(6, 9, 2);
```

```
        refPolimorfica = es;
        Console.WriteLine(refPolimorfica.GetDetalles());
        f=es ;
        f.Dibujar();
        Console.WriteLine();

        up.MuestraDetalles(e);
        up.MuestraDetalles(s);
        up.MuestraDetalles(g);
        up.MuestraDetalles(a);
        up.MuestraDetalles(c);
        up.MuestraDetalles(r);
        up.MuestraDetalles(t);
        up.MuestraDetalles(es);

        Console.WriteLine();

        Console.WriteLine("Uso del garage para el Empleado:");
        up.PermisoParaEstacionar(e);
        Console.WriteLine("Uso del garage para la Secretaria:");
        up.PermisoParaEstacionar(s);
        Console.WriteLine("Uso del garage para el Gerente:");
        up.PermisoParaEstacionar(g);
        Console.WriteLine("Uso del garage para el Agente de Seguridad:");
        up.PermisoParaEstacionar(a);

        Console.ReadKey();
}

public void MuestraDetalles(IDescriptor refPolimorfica){
        Console.WriteLine("-----------------");
        Console.WriteLine(refPolimorfica.GetDetalles());
        Console.WriteLine("-----------------");
        Console.ReadKey();
}

public void PermisoParaEstacionar(IEstacionamiento e)
{
        e.Estacionar();
}
```

VB

```
Sub Main()
        Dim refPolimorfica As IDescriptor
        Dim f As Figura

        Dim d As Direccion = New Direccion("Juan De Garay", 870)

        d = Nothing
        d = New Direccion("Amenabar", 1570)
        Dim e As Empleado = New Empleado("Sergio","Omar","Perotti","20.200.555", d, _
                "Administración", "EA01", "GAD50", 1500.0F)
        d = Nothing
        d = New Direccion("Allen", 1070)
        Dim s As Secretaria = New Secretaria("Laura", "Verónica", "Diaz", _
                "20.555.555", d, _
                "Directorio", "SV01", "GV50", 1800.0F)
        d = Nothing
```

```
d = New Direccion("Juramento", 333)
Dim g As Gerente = New Gerente("Juan", "Pedro", "Goyena", "17.767.076", d, _
       "Ventas", "0", "GV50", 5500.0F, 20)
d = Nothing
d = New Direccion("Belgrano", 444)
Dim a As AgenteDeSeguridad = New AgenteDeSeguridad("Hernán", "Adrián", _
     "Perez", "20.888.999", d, "207272")

refPolimorfica = e
Console.WriteLine(refPolimorfica.getDetalles())
Console.WriteLine()
refPolimorfica = s
Console.WriteLine(refPolimorfica.getDetalles())
Console.WriteLine()
refPolimorfica = g
Console.WriteLine(refPolimorfica.getDetalles())
Console.WriteLine()
refPolimorfica = a
Console.WriteLine(refPolimorfica.getDetalles())

Dim c As Circulo = New Circulo(3, 6, 1)
refPolimorfica = c
Console.WriteLine(refPolimorfica.getDetalles())
f = c
f.Dibujar()
Console.WriteLine()
Dim r As Rectangulo = New Rectangulo(4, 7, 2, 3)
refPolimorfica = r
Console.WriteLine(refPolimorfica.getDetalles())
f = r
f.Dibujar()
Console.WriteLine()
Dim t As Triangulo = New Triangulo(5, 8, 2, 3)
refPolimorfica = t
Console.WriteLine(refPolimorfica.getDetalles())
f = t
f.Dibujar()
Console.WriteLine()
Dim es As Esfera = New Esfera(6, 9, 2)
refPolimorfica = es
Console.WriteLine(refPolimorfica.getDetalles())
Console.WriteLine()

MuestraDetalles(e)
MuestraDetalles(s)
MuestraDetalles(g)
MuestraDetalles(a)
MuestraDetalles(c)
MuestraDetalles(r)
MuestraDetalles(t)
MuestraDetalles(es)

Console.WriteLine()

Console.WriteLine("Uso del garage para el Empleado:")
PermisoParaEstacionar(e)
Console.WriteLine("Uso del garage para la Secretaria:")
PermisoParaEstacionar(s)
```

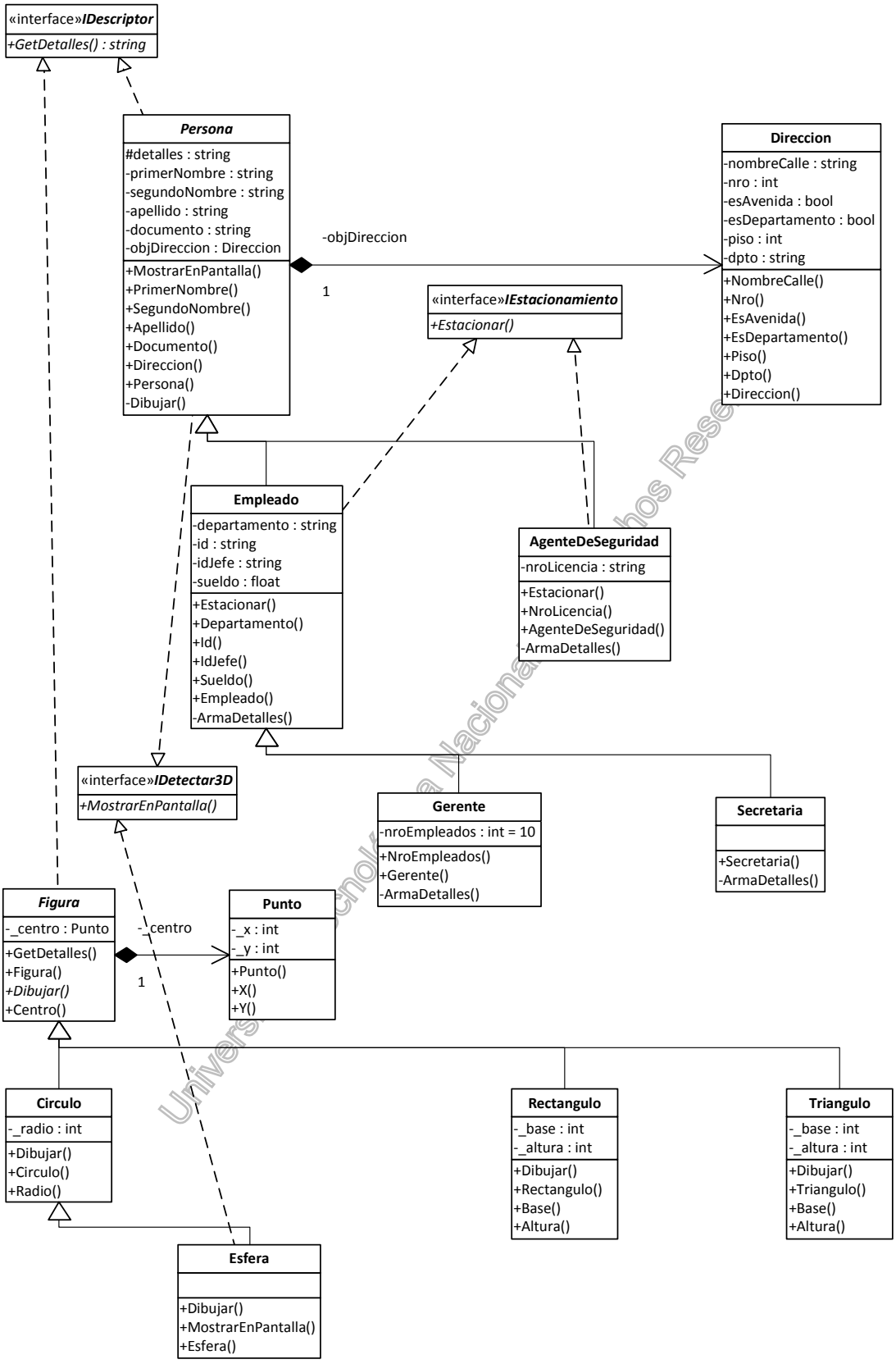
```
        Console.WriteLine("Uso del garage para el Gerente:")
        PermisoParaEstacionar(g)
        Console.WriteLine("Uso del garage para el Agente de Seguridad:")
        PermisoParaEstacionar(a)

        Console.ReadKey()
End Sub

Public Sub MuestraDetalles(ByVal refPolimorfica As IDescriptor)
        Console.WriteLine()
        Console.WriteLine("-----------------")
        Console.WriteLine(refPolimorfica.getDetalles())
        Console.WriteLine("-----------------")
        Console.ReadKey()
End Sub

Public Sub PermisoParaEstacionar(ByVal e As IEstacionamiento)
        e.estacionar()
End Sub
```

Interfaces Múltiples

Dado que .Net permite implementar más de una interfaz en una misma clase, se posee la flexibilidad de usar aquellas interfaces que se consideren necesarias para reflejar las distintas funcionalidades en común que tienen las cadenas de herencia diferentes.

Si al ejemplo anterior se incluye una funcionalidad en la cual se muestre en pantalla cuando un objeto es de 3 dimensiones, con declarar una interfaz e implementarla en las clases apropiadas se consigue, inclusive, que el compilador advierta si se quiere usar un objeto que no implemente la interfaz. Esta práctica logra que se prevengan errores e independiza el código.

El siguiente gráfico UML muestra el ejemplo propuesto


«interface»IDescriptor
+GetDetalles() : string
Persona
#detalles : string
-primerNombre : string
-segundoNombre : string
-apellido : string
-documento : string
-objDireccion : Direccion
+MostrarEnPantalla()
+PrimerNombre()
+SegundoNombre()
+Apellido()
+Documento()
+Direccion()
+Persona()
-Dibujar()
-objDireccion
1
Direccion
-nombreCalle : string
-nro : int
-esAvenida : bool
-esDepartamento : bool
-piso : int
-dpto : string
+NombreCalle()
+Nro()
+EsAvenida()
+EsDepartamento()
+Piso()
+Dpto()
+Direccion()
«interface»IEstacionamiento
+Estacionar()
Empleado
-departamento : string
-id : string
-idJefe : string
-sueldo : float
+Estacionar()
+Departamento()
+Id()
+IdJefe()
+Sueldo()
+Empleado()
-ArmaDetalles()
AgenteDeSeguridad
-nroLicencia : string
+Estacionar()
+NroLicencia()
+AgenteDeSeguridad()
-ArmaDetalles()
«interface»IDetectar3D
+MostrarEnPantalla()
Gerente
-nroEmpleados : int = 10
+NroEmpleados()
+Gerente()
-ArmaDetalles()
Secretaria
+Secretaria()
-ArmaDetalles()
Figura
-_centro : Punto
+GetDetalles()
+Figura()
+Dibujar()
+Centro()
-_centro
1
Punto
-_x : int
-_y : int
+Punto()
+X()
+Y()
Circulo
-_radio : int
+Dibujar()
+Circulo()
+Radio()
Rectangulo
-_base : int
-_altura : int
+Dibujar()
+Rectangulo()
+Base()
+Altura()
Triangulo
-_base : int
-_altura : int
+Dibujar()
+Triangulo()
+Base()
+Altura()
Esfera
+Dibujar()
+MostrarEnPantalla()
+Esfera()

Se puede apreciar como la clase Persona implementa ambas interfaces, con lo cual se puede interpretar a la clase como que es de **ambos tipos**.

Ejemplo

Clase Persona

C#

```
abstract class Persona: IDescriptor, IDetectar3D
{
	protected String detalles;

	private String primerNombre;

	public String PrimerNombre
	{
		get { return primerNombre; }
		set { primerNombre = value; }
	}
	private String segundoNombre;

	public String SegundoNombre
	{
		get { return segundoNombre; }
		set { segundoNombre = value; }
	}
	private String apellido;

	public String Apellido
	{
		get { return apellido; }
		set { apellido = value; }
	}
	private String documento;

	public String Documento
	{
		get { return documento; }
		set { documento = value; }
	}

	private Direccion objDireccion;

	public Direccion Direccion
	{
		get { return objDireccion; }
		set { objDireccion = value; }
	}


	public Persona(String p,
		String s,
		String a,
		String d,
		Direccion objD)
	{
```

```
            primerNombre = p;
            segundoNombre = s;
            apellido = a;
            documento = d;
            objDireccion = objD;
        }

        public abstract String GetDetalles();

        public void MostrarEnPantalla()
        {
            Console.WriteLine("Detectando dibujo automático en 3D.");
            Dibujar();
        }
        private void Dibujar()
        {
            Console.WriteLine("Dibujando persona");
            Console.WriteLine(GetDetalles());
        }
}
```

VB

```
Public MustInherit Class Persona
    Implements IDescriptor, IDetectar3D

    Private _primerNombre As String
    Private _segundoNombre As String
    Private _apellido As String
    Private _documento As String
    Private dir As Direccion
    Private detalles As String

    Public Property Direccion() As Direccion
        Get
            Return dir
        End Get
        Set(ByVal value As Direccion)
            dir = value
        End Set
    End Property

    Public Property PrimerNombre() As String
        Get
            PrimerNombre = _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property

    Public Property SegundoNombre() As String
        Get
            SegundoNombre = _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = value
        End Set
    End Property
```

```
    Public Property Apellido() As String
        Get
            Apellido = _apellido
        End Get
        Set(ByVal value As String)
            _apellido = value
        End Set
    End Property

    Public Property Documento() As String
        Get
            Documento = _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                   ByVal _documento As String, ByVal dir As Direccion)
        _primerNombre = p
        _segundoNombre = s
        _apellido = a
        _documento = _documento
        Me.dir = dir
    End Sub

    Private Sub ArmaDetalles()
        detalles = "Apellido: " + Apellido + " Primer Nombre: " + PrimerNombre + _
        " Segundo Nombre: " + SegundoNombre + " Documento: " + Documento + _
        " Direccion: " + dir.NombreCalle + " " + dir.Nro
    End Sub

    Public Sub MostrarEnPantalla() Implements IDetectar3D.MostrarEnPantalla
        Console.WriteLine("Detectando dibujo automático en 3D.")
        dibujar()
    End Sub

    Public Overridable Sub Dibujar()
        Console.WriteLine("Dibujando persona")
        Console.WriteLine(getDetalles())
    End Sub

   Public MustOverride Function getDetalles() As String Implements _
            IDescriptor.GetDetalles
End Class
```


Sin embargo, nuevamente, como la clase Persona es abstracta, el hecho de no implementar el método de la interfaz no afecta en nada a su comportamiento. Este no es el caso de las subclases que al ser concretas están obligadas a cumplir el contrato, por lo tanto, al menos Empleado y AgenteDeSeguridad que tienen como superclase a Persona deben rescribir obligatoriamente el método **de ambas interfaces** para que se puedan crear objetos con ellas. A partir de este punto se comporta como un ejemplo de polimorfismo normal con rescritura. Notar que se indica que la función **debe** ser sobrescrita.

Clase Empleado

C#

```
class Empleado : Persona, IEstacionamiento
{
	private String departamento;

	public String Departamento
	{
		get { return departamento; }
		set { departamento = value; }
	}
	private String id;

	public String Id
	{
		get { return id; }
		set { id = value; }
	}
	private String idJefe;

	public String IdJefe
	{
		get { return idJefe; }
		set { idJefe = value; }
	}
	private float sueldo;

	public float Sueldo
	{
		get { return sueldo; }
		set { sueldo = value; }
	}

	public Empleado(String p, String s, String a, String doc, Direccion objDir,
		String dep, String i, String ij, float sdo):base(p, s, a, doc, objDir)
	{
		departamento = dep;
		id = i;
		idJefe = ij;
		sueldo = sdo;
	}

	public override String GetDetalles()
	{
		ArmaDetalles();
		return detalles;
	}

	private void ArmaDetalles()
	{
		detalles = "Apellido: " + Apellido + " Primer Nombre: " + PrimerNombre+
		" Segundo Nombre: " + SegundoNombre + " Documento: " + Documento +
		" Direccion: " + Direccion.NombreCalle + " " + Direccion.Nro +
		" " + Direccion.Piso + "°" + Direccion.Dpto +
		" Departamento: " + Departamento + " ID:" + Id +
		" ID del Jefe:" + IdJefe + " Sueldo:" + Sueldo;
```

```
        }

        public virtual void Estacionar()
        {
                Console.WriteLine(
                        "Esta persona no puede estacionar en el garage de la empresa");
        }
}
```

VB

```
Public Class Empleado : Inherits Persona : Implements IEstacionamiento
        Private _departamento As String
        Private _id As String
        Private _idJefe As String
        Private _sueldo As Single
        Private detalles As String

        Public Property Departamento() As String
                Get
                        Departamento = _departamento
                End Get
                Set(ByVal value As String)
               _departamento = value
                End Set
        End Property

        Public Property Id() As String
                Get
                        Id = _id
                End Get
                Set(ByVal value As String)
                        _id = value
                End Set
        End Property

        Public Property IdJefe() As String
                Get
                        IdJefe = _idJefe
                End Get
                Set(ByVal value As String)
                        _idJefe = value
                End Set
        End Property

        Public Property Sueldo() As Single
                Get
                        Sueldo = _sueldo
                End Get
                Set(ByVal value As Single)
                        _sueldo = value
                End Set
        End Property

        Public Sub New(ByVal p As String, ByVal s As String, ByVal a As String, _
                        ByVal d As String, ByVal dir As Direccion, _
                        ByVal departamento As String, ByVal id As String, _
                        ByVal idJefe As String, ByVal sueldo As Single)
                MyBase.New(p, s, a, d, dir)
```

```
            _departamento = departamento
            Me._id = id
            _idJefe = idJefe
            _sueldo = sueldo
        End Sub

        Public Overrides Function GetDetalles() As String
            ArmaDetalles()
            Return detalles
        End Function

        Private Sub ArmaDetalles()
            Dim s As String = Sueldo.ToString

            detalles = "Apellido: " + Apellido +" Primer Nombre: "+PrimerNombre + _
                " Segundo Nombre: " + SegundoNombre+" Documento: "+Documento + _
                " Direccion: " + Direccion.NombreCalle + " " + Direccion.Nro + _
                " Departamento: " + Departamento + " ID:" + Id + _
                " ID del Jefe:" + IdJefe + " Sueldo:" + s
        End Sub

        Public Overridable Sub Estacionar() Implements IEstacionamiento.Estacionar
            Console.WriteLine( _
                "Esta persona no puede estacionar en el garage de la empresa")
        End Sub
End Class
```

Clase AgenteDeSeguridad

C#

```
class Empleado : Persona, IEstacionamiento
{
        private String departamento;

        public String Departamento
        {
            get { return departamento; }
            set { departamento = value; }
        }
        private String id;

        public String Id
        {
            get { return id; }
            set { id = value; }
        }
        private String idJefe;

        public String IdJefe
        {
            get { return idJefe; }
            set { idJefe = value; }
        }
        private float sueldo;

        public float Sueldo
        {
```

```
            get { return sueldo; }
            set { sueldo = value; }
        }

        public Empleado(String p, String s, String a, String doc, Direccion objDir,
            String dep, String i, String ij, float sdo):base(p, s, a, doc, objDir)
        {
            departamento = dep;
            id = i;
            idJefe = ij;
            sueldo = sdo;
        }

        public override String GetDetalles()
        {
            ArmaDetalles();
            return detalles;
        }

        private void ArmaDetalles()
        {
            detalles = "Apellido: " + Apellido + " Primer Nombre: " +PrimerNombre +
                " Segundo Nombre: " + SegundoNombre + " Documento: " +Documento+
                " Direccion: " + Direccion.NombreCalle + " " + Direccion.Nro +
                " " + Direccion.Piso + "°" + Direccion.Dpto +
                " Departamento: " + Departamento + " ID:" + Id +
                " ID del Jefe:" + IdJefe + " Sueldo:" + Sueldo;
        }

        public virtual void Estacionar()
        {
            Console.WriteLine(
                "Esta persona no puede estacionar en el garage de la empresa");
        }
}
```

VB

```
Public Class AgenteDeSeguridad : Inherits Persona : Implements IEstacionamiento
    Private _nroLicencia As String
    Private detalles As String

    Public Property NroLicencia() As String
        Get
            NroLicencia = _nroLicencia
        End Get
        Set(ByVal value As String)
            _nroLicencia = value
        End Set
    End Property

    Public Sub New(ByVal p As String, ByVal s As String, _
            ByVal a As String, ByVal d As String, _
            ByVal dir As Direccion, ByVal nroLicencia As String)
        MyBase.New(p, s, a, d, dir)
        _nroLicencia = nroLicencia
    End Sub

    Public Overrides Function GetDetalles() As String
```

```
            ArmaDetalles()
            Return detalles
        End Function

        Private Sub ArmaDetalles()
            Dim n As String = _nroLicencia.ToString

            detalles = "Apellido: " + Apellido + " Primer Nombre: "+PrimerNombre+ _
                " Segundo Nombre: " + SegundoNombre +" Documento: "+Documento+ _
                " Direccion: " + Direccion.NombreCalle + " " + Direccion.Nro + _
                " Nro de Licencia:" + n
        End Sub

        Public Sub Estacionar() Implements IEstacionamiento.Estacionar
            Console.WriteLine( _
                "Esta persona no puede estacionar en el garage de la empresa")
        End Sub
End Class
```

Clase Esfera

C#

```
public class Esfera : Circulo, IDetectar3D
{
        public Esfera(int centroX, int centroY, int radio)
            : base(centroX, centroY, radio)
        {

        }

        public void MostrarEnPantalla()
        {
            Console.WriteLine("Detectando dibujo automático en 3D.");
            Dibujar();
        }

        public override void Dibujar()
        {
            Console.WriteLine("Dibujando esfera");
            GetDetalles();
        }

        public override string GetDetalles()
        {
            return "Esfera. " + base.GetDetalles();
        }
}
```

VB

```
Public Class Esfera
    Inherits Circulo
    Implements IDetectar3D

    Public Sub New(ByVal x As Integer, ByVal y As Integer, ByVal radio As Integer)
        MyBase.New(x, y, radio)

    End Sub
```

```
    Public Overrides Function getDetalles() As String
        Return "Esfera. " + MyBase.getDetalles()
    End Function

    Public Overrides Sub Dibujar()
        Console.WriteLine("Dibujo automático en 2D. Dibujando rectángulo")
        Console.WriteLine(getDetalles())
    End Sub

    Public Sub MostrarEnPantalla1() Implements IDetectar3D.MostrarEnPantalla
        Console.WriteLine("Detectando dibujo automático en 3D.")
        Dibujar()
    End Sub
End Class
```

El extracto de código que muestra la utilización de la interfaz múltiple para los objetos del tipo Persona (las declaraciones e inicializaciones de los objetos son iguales al ejemplo anterior) es la siguiente

C#

```
        up.MuestraDetalles(e);
        up.MuestraDetalles(s);
        up.MuestraDetalles(g);
        up.MuestraDetalles(a);
        up.MuestraDetalles(c);
        up.MuestraDetalles(r);
        up.MuestraDetalles(t);
        up.MuestraDetalles(es);

        up.Dibujo3D(e);
        up.Dibujo3D(s);
        up.Dibujo3D(g);
        up.Dibujo3D(a);
        up.Dibujo3D(es);

        Console.WriteLine();

        Console.WriteLine("Uso del garage para el Empleado:");
        up.PermisoParaEstacionar(e);
        Console.WriteLine("Uso del garage para la Secretaria:");
        up.PermisoParaEstacionar(s);
        Console.WriteLine("Uso del garage para el Gerente:");
        up.PermisoParaEstacionar(g);
        Console.WriteLine("Uso del garage para el Agente de Seguridad:");
        up.PermisoParaEstacionar(a);

        Console.ReadKey();

}

public void MuestraDetalles(IDescriptor refPolimorfica){
              Console.WriteLine("-----------------");
              Console.WriteLine(refPolimorfica.GetDetalles());
              Console.WriteLine("-----------------");
              Console.ReadKey();
}
```

```
public void PermisoParaEstacionar(IEstacionamiento e)
{
      e.Estacionar();
}

public void Dibujo3D(IDetectar3D d)
{
            Console.WriteLine("-----------------");
            d.MostrarEnPantalla();
            Console.WriteLine("-----------------");
            Console.ReadKey();
}
```

VB

```
        muestraDetalles(e)
        muestraDetalles(s)
        muestraDetalles(g)
        muestraDetalles(a)
        muestraDetalles(c)
        muestraDetalles(r)
        muestraDetalles(t)
        muestraDetalles(es)

        Dibujo3D(e)
        Dibujo3D(s)
        Dibujo3D(g)
        Dibujo3D(a)
        Dibujo3D(es)

        Console.WriteLine()

        Console.WriteLine("Uso del garage para el Empleado:")
        permisoParaEstacionar(e)
        Console.WriteLine("Uso del garage para la Secretaria:")
        permisoParaEstacionar(s)
        Console.WriteLine("Uso del garage para el Gerente:")
        permisoParaEstacionar(g)
        Console.WriteLine("Uso del garage para el Agente de Seguridad:")
        permisoParaEstacionar(a)

        Console.ReadKey()

    End Sub

    Public Sub muestraDetalles(ByVal refPolimorfica As IDescriptor)
        Console.WriteLine()
        Console.WriteLine("-----------------")
        Console.WriteLine(refPolimorfica.GetDetalles())
        Console.WriteLine("-----------------")
        Console.ReadKey()
    End Sub

    Public Sub permisoParaEstacionar(ByVal e As IEstacionamiento)
        e.Estacionar()
    End Sub

    Private Sub Dibujo3D(d As IDetectar3D)
        Console.WriteLine("-----------------")
```

```
        d.MostrarEnPantalla()
        Console.WriteLine("------------------")
        Console.ReadKey()
    End Sub
```

Ejercicios 4 y 5

Los temas de los que se tratan en este ejercicio son los siguientes:

- Polimorfismo
- Interfaces

Aplicar los conocimientos adquiridos para polimorfismo e interfaces codificando los diseños del módulo anterior.