Licenciatura en Gestión Tecnológica

Programación Avanzada II

Formación Continua

Fundamentos de la Programación Orientada a Objetos

Ing. Mariano Juiz

2do Cuatrimestre (2015)

Agenda

1. Conceptos Básicos
2. Elementos Básicos
3. Clase String

Conceptos Básicos

¿Qué es la Abstracción?

Es el proceso mental donde las ideas son separadas de los objetos concretos.

Es completamente subjetivo y dependiente del contexto

El objetivo es separar efectivamente los objetos concretos de la idea que queremos representar.

En la POO la abstracción se plantea en términos de similitudes entre fenómenos, conceptos, entidades, etc.

De esta manera, logramos identificar conceptos generales (persona, auto, etc.) que puedan ser traducidos a construcciones básicas (objetos) en nuestro paradigma.

Conceptos Básicos

¿Qué es un modelo?

Es una versión simplificada de algún fenómeno o entidad del mundo real.

¿Qué significa modelar?

Es un proceso de abstracción.

Tomamos una versión reducida de lo que queremos representar. Sólo especificamos aquellas cosas que son relevantes

Modelo Orientado a Objetos

Los conceptos del dominio se representan como objetos.

Los objetos se componen y colaboran con otros objetos para formar un modelo.

La ejecución de un programa OO puede verse como un modelo simulando el comportamiento de una parte del Mundo.

Introducción

¿Qué es Programación Orientada a Objetos?

Los sistemas están compuestos por un conjunto de **objetos**.

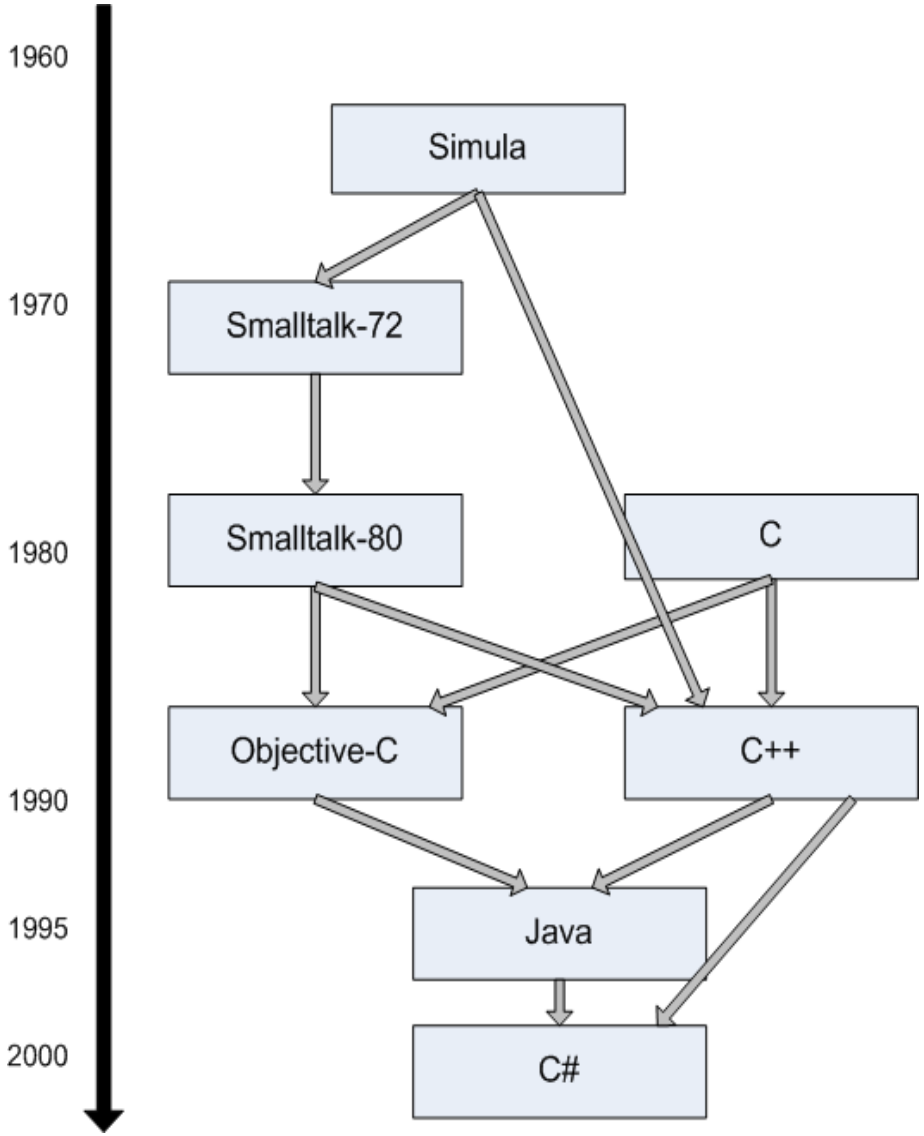
Los objetos son **responsables** de llevar a cabo ciertas acciones y **colaboran** para llevar a cabo sus responsabilidades. Además pueden **componer** otros objetos

Los programas están organizados en base a clases y jerarquías de herencia

La forma de pedirle a un objeto que lleve a cabo una determinada tarea es por medio del envío de un **mensaje**.

Conceptos Básicos

Historia de la Programación Orientada a Objetos



Elementos Básicos

¿Qué es un Objeto?

Los objetos son los **elementos primarios** que utilizamos para construir programas.

Todo es un objeto en la programación orientada a objetos.

Un objeto es una **abstracción** de una **entidad** del **dominio** del problema.

Cada objeto es una construcción caracterizada por su **comportamiento**, su **estado interno (encapsulamiento)** y su **identidad**.

Elementos Básicos

Comportamiento de un Objeto

Un objeto se define en términos de su comportamiento, que indica **qué** sabe hacer el objeto (**responsabilidades**).

Se especifica a través del **conjunto de mensajes** que el objeto sabe responder: protocolo

La **implementación** indica **cómo** hace el objeto para responder a sus mensajes. Es privada y se especifica a través de un conjunto de métodos.

El estado interno

Está compuesto por las **variables de instancia** del objeto, quienes pueden hacer referencia a:

- Propiedades intrínsecas del objeto (un número, un valor booleano, etc).
- Otros objetos con los cuales pueda colaborar para llevar a cabo sus responsabilidades.

Es **privado** del objeto. Ningún otro objeto puede accederlo

Elementos Básicos

Identidad

La poseen **todos** los objetos

Podemos distinguir objetos individuales dentro de un sistema

Envío de Mensajes

Para poder enviarle un mensaje a un objeto, hay que conocerlo.

Al enviarle un mensaje a un objeto, éste responde activando el método asociado a ese mensaje (siempre y cuando exista).

Como resultado del envío de un mensaje puede retornarse un objeto.

Elementos Básicos

Especificación de un Mensaje

- ¿Cómo se especifica un mensaje?
  - Se especifica con el ***nombre*** correspondiente al protocolo del objeto receptor.
  - Se indica cuáles son los ***parámetros***, es decir, la información necesaria para resolver el mensaje.

- Cada lenguaje de programación propone una sintaxis particular para indicar el envío de un mensaje.

- La sintaxis es:

```
<objeto receptor>.<nombre de mensaje> (<parámetros>);
```

- Ejemplo:
  - Decirle a una cuenta bancaria que deposite $100 se escribe como:

```
unaCuenta.depositar(100);
```

Elementos Básicos

¿Qué es una Clase?

- Es una descripción abstracta de un conjunto de objetos y es responsable de crear sus instancias
- Describe el formato y agrupan el comportamiento en común de sus instancias
- Su especificación es mediante un nombre, atributos que definen un estado interno y los métodos que definen su comportamiento
- Todas las instancias de una clase se comportan de igual manera, pero cada una de ella tendrá su propio estado interno

Elementos Básicos

¿Qué es una Clase?

<modificadorDeVisibilidad> class **<nombre>**

{

<miembros>

}

Ejemplo

public class **Alumno**

{

}

Elementos Básicos

Modificadores de acceso

- Los modificadores son elementos del lenguaje que se colocan delante de la definición de variables de instancia y métodos que alteran o condicionan el significado del elemento.

- Los modificadores de acceso permiten al diseñador de una clase determinar quien accede a los datos y métodos de la misma.

- En C#, por defecto, se considera que los métodos y variables de instancia solo son accesibles desde el código interno de la clase.

Elementos Básicos

Modificadores de acceso

Los miembros pueden ser:

- **public**: Pueden ser accedidos desde cualquier objeto o sección de código.
- **protected**: Desde una clase sólo puede accederse a miembros protected de objetos de esa misma clase o de subclases suyas.
- **private**: Sólo pueden ser accedidos desde el código de la clase a la que pertenece. Es lo considerado por defecto.
- **internal**: sólo pueden ser accedidos desde código perteneciente al ensamblado en que se han definido.
- **protected internal** : sólo pueden ser accedidos desde código perteneciente al ensamblado en que se han definido o desde clases que deriven de la clase donde se ha definido

Elementos Básicos

Modificadores de acceso

También es posible aplicar modificadores de acceso a las clases.

Los modificadores de acceso permitidos son:

public: Es posible acceder a la clase desde cualquier assembly.

internal: Sólo es posible acceder a la clase desde el assembly donde se declaró. Es lo considerado por defecto.

Elementos Básicos

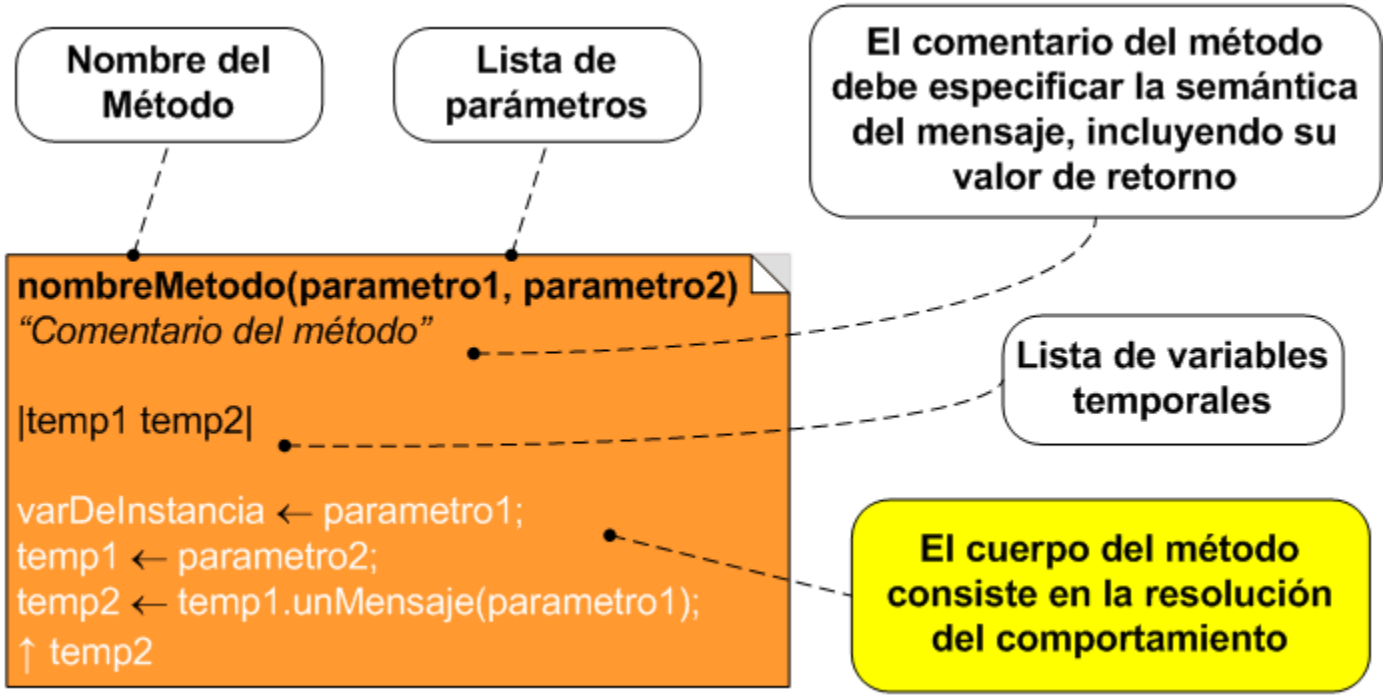
Métodos

- ¿Qué es un método?
  - Es la contraparte funcional del mensaje.
  - Expresa la forma de llevar a cabo la semántica propia de un mensaje particular (el *cómo*).

- Un método puede realizar básicamente 3 cosas:
  - Modificar el estado interno del objeto.
  - Colaborar con otros objetos (enviándoles mensajes).
  - Retornar y terminar.

Elementos Básicos

Métodos



Elementos Básicos

Encapsulamiento

- Oculta detalles de implementación.
- Protege el estado interno de los objetos.
- Un objeto sólo muestra su “cara visible” por medio de su protocolo.
  - Los cambios internos no deberían impactar en otros objetos
- Es el objeto quien decide qué se publica.
- Facilita modularidad y reutilización.
  - El acoplamiento entre objetos debería ser bajo
  - El software debería escalar mejor ante cambios
- Se establece un contrato entre objetos
  - Lo que les importa a los objetos (proveedor y consumidor) es cuál es el mensaje a proveer/consumir, qué parámetros debe recibir/enviar y cual es la semántica asociada al mensaje recibido/enviado.

Elementos Básicos

¿Qué es una Instancia?

- Es el mecanismo de creación de objetos, a partir de una clase.
- Es decir, un objeto es una instancia de una clase.
- Todas las instancias de una misma clase tendrán la misma estructura interna, responderán los mismos mensajes, pero tendrán su propio estado.
- Normalmente se utiliza la palabra **new** para instanciar nuevos objetos

Elementos Básicos

Constructores

- Son llamados cuando se quiere crear una instancia de una clase
- Tienen el mismo nombre que la clase
- Existe uno por defecto, pero pueden existir varios (sobrecarga)
  - Por defecto, es un constructor sin parámetros, que crea instancias del objeto y establece las variables miembro con los valores predeterminados.
- Inicializan valores
- Pueden sobrescribirse

Ejemplo:

```
public class Persona
{
        public Persona()
        {
                …
        }
}
```

Elementos Básicos

¿Cómo se comunican los objetos?

- El objeto consumidor debe poder nombrar al objeto proveedor. Existe una ligadura (binding) entre el nombre y el objeto.

- El binding puede ser estático o dinámico

- Existen 4 tipos de relaciones:
  - Variables de Instancia (Conocimiento Interno)
  - Parámetros (Conocimiento externo)
  - Variables Temporales (Conocimiento Temporal)
  - Seudo Variables (Conocimiento especial)

Elementos Básicos

Variables de Instancia

- Define la relación entre un objeto y sus atributos
- Se definen explícitamente como parte de la estructura de la clase
- La relación dura desde la creación del objeto hasta su destrucción

Parámetros

- Son los parámetros que se reciben junto con el mensaje
- El nombre de la relación es el nombre del método
- La relación dura el tiempo que el método se encuentra activo.
- La ligadura (binding) entre el nombre y el objeto no puede alterarse durante la ejecución del método.

Elementos Básicos

Variables temporales

- Definen relaciones temporales dentro de un método.
- La relación con el objeto se crea durante la ejecución del método y su nombre se define explícitamente en el método.
- La relación se mantiene dentro del contexto donde fue definida la variable.
- Durante la ejecución del método, la ligadura (binding) entre el nombre y el objeto puede alterarse

Seudo Variable (this)

- Se utiliza para que un objeto se envíe un mensaje a sí mismo
- Es como una variable, pero no se declara y no se puede modificar

Elementos Básicos

Propiedades

- Son el mecanismo que C# provee para encapsular un campo de un objeto con un método de lectura (get) y un método de escritura (set).
- Permite asociar código a ejecutar en cada asignación o lectura de su valor.
- Puede omitirse uno de los dos métodos.
- Si sólo se define el método get, la propiedad es de solo lectura
- La sintaxis de acceso es como una variable pública:

```
val = obj.prop //get
obj.prop = val //set
```

Elementos Básicos

Propiedades

```
<modificadorDeVisibilidad> <tipo> <nombre>
{
    get{    //Codigo con un valor de retorno del tipo <tipo>}
    set{    //codigo }
}
```

Ejemplo

```
private string nombre;          //variable

public string Nombre            //Propiedad
{
            get         { return nombre;}
            set         { nombre = value;}
}
```

Opcionalmente el get o el set (pero no ambos) pueden cambiar la accesibilidad del mismo

Elementos Básicos

Métodos

- Son el comportamiento que se encarga de ejecutar las acciones que se requieren para que un objeto realice su trabajo
- Puede no tener parámetros
- No se pueden escribir dos métodos con el mismo nombre, los mismos parámetros y que retornen distintos tipos de valores.
- No se pueden escribir dos métodos con el mismo nombre que devuelvan distinto tipo de datos, salvo que se utilice **Generics**
- Se pueden escribir dos o más métodos con el mismo nombre que reciban distintos parámetros (**sobrecarga de métodos**)
- .

Elementos Básicos

Métodos

<modificadorDeVisibilidad> <tipo> <nombre>(<Parámetros>)

```
    {
            //Código
     }
```

<tipo> es el tipo de retorno o **void** si no retorna nada

Ejemplos

```
public void setearSaldo(int monto)
{
            saldo = monto;
}

public int incrementarSaldo(int monto)
{
                        saldo = saldo + monto;
                        return saldo;
}
```

Elementos Básicos

Herencia

- Es uno de los pilares fundamentales en los que se basa la programación orientada a objetos.
- Es un mecanismo que permite definir nuevas clases a partir de otras ya definidas y reutilizar sus atributos y métodos (dependiendo de sus modificadores de acceso) sin declararlos en la nueva clase
- En c# no existe la herencia múltiple

Elementos Básicos

Constructores

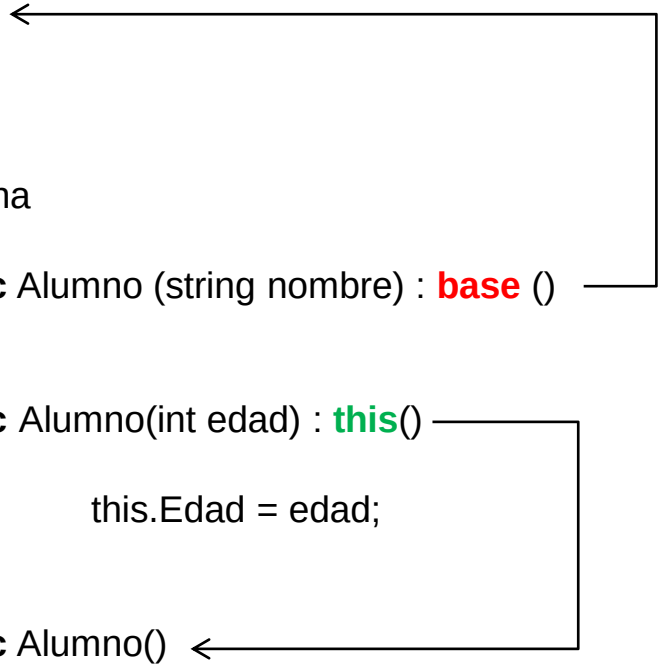
Nuevo Ejemplo:

```
public class Persona
{
        public Persona()
        { … }
}

public class Alumno : Persona
{
                public Alumno (string nombre) : base ()
                {… }

                public Alumno(int edad) : this()
                {
                        this.Edad = edad;
                }

                public Alumno()
                {
                        this.Nombre = “NN”;
                }
}
```



Elementos Básicos

Herencia

<modificadorDeVisibilidad> class <nombre> : <superclase>
{
}

Ejemplo

```
public class Persona
{
    //Código
}

public class Alumno : Persona
{
    //Código
}
```

Elementos Básicos

Clases Abstractas

Permiten crear “método generales”, que recrean un comportamiento común, pero sin especificar cómo lo hacen.

Una clase abstracta **no** podrá ser instanciada.

Una clase abstracta puede contener métodos abstractos, proporcionando a sus subclases la declaración de todos los métodos necesarios para implementar.

Las clases que hereden de la clase Abstracta deben implementar todos los métodos abstractos.

Los nombre y los parámetros de los métodos, así como el tipo de dato, deben respetarse, de lo contrario se está hablando de otro método totalmente diferente.

Si la clase es abstracta puede tener métodos con implementación

Los métodos abstract no pueden ser privados

Elementos Básicos

Clases Abstractas

```
public abstract class Figura {
    public int numeroLados;
    public int area;
    public int volumen;

    abstract public void calcularArea();
    abstract public void calcularVolumen();
}

public class Esfera : Figura{
    public float radio;
    public static float pi = (float)(3.1415);

    public Esfera( int radio ){
            this.radio = radio;
            this.numeroLados = 0;
    }       // 4*pi*r²

    public void calcularArea(){
            this.area =  (4)*pi*radio*radio;
    }       //              (4/3)*pi*radio³

    public void calcularVolumen(){
            this.volumen =  (4/3)*pi*radio*radio*radio;
    }
}
```

Elementos Básicos

Interfaces

Las interfaces, tal como las clases abstractas. no se pueden instanciar. Sus métodos deben ser “re-escritos” por la clase que los **implemente**.

Las interfaces definen un “contrato” el cual deben cumplir las clases que lo implementen

Elementos Básicos

Interfaces

```
public interface MatematicaVectorial {
	public static double pi = 3.1415;
	public static double e = 2.71828;

	public double[] sumar(int[][] vectores);
	public double[] restar(int[][] vectores);
	public double productoPunto(int[][] vectores);
}

public class VectorR3 implements MatematicaVectorial{
	public double valorX;
	public double valorY;
	public double valorZ;

	public double[] sumar(int[][] vectores){
			//Implementacion
	}

	public double[] restar(int[][] vectores){
			//Implementacion
	}

	public double productoPunto(int[][] vectores)	{
			//Implementacion
	}
}
```

Elementos Básicos

Métodos virtuales

Se utilizan para indicar que los métodos de la clase base se deben redefinir.

Se utiliza la palabra **virtual** en el/los método/s involucrado/s.

La sintaxis es la siguiente:

<modificadorDeVisibilidad> **virtual** <tipoRetorno> <nombreMétodo>(<parámetros>)
{
    //código
}

En la clase derivada, el modificador **override** extiende el método de la clase base

Elementos Básicos

override

En la clase derivada, el modificador **override** extiende el método de la clase base

La sintaxis es la siguiente:

<modificadorDeVisibilidad> **override** <tipoRetorno> <nombreMétodo>(<parámetros>)
{
    //código
}

new

En la clase derivada, el modificador **new** crea un nuevo método con el mismo nombre y oculta el método original

La sintaxis es la siguiente:

<modificadorDeVisibilidad> **new** <tipoRetorno> <nombreMétodo>(<parámetros>)
{
    //código
}

Elementos Básicos

Polimorfismo

Permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto que se referencia en ese momento

Se utiliza en clases derivadas

Binding

Es el tiempo en que se liga el envío de un mensaje con la implementación del método correspondiente

- Estático: se resuelve en tiempo de compilación
- Dinámico: se resuelve en tiempo de ejecución, nos permite programar despreocupándonos del mecanismo de selección del método adecuado

Clase String

Clase String

¿Qué es un String?

En C# es una colección de Tipos de Datos **char**

Es de Tipo Referencia (se crea en el HEAP)

string cadena = “Soy un String”;

Tipo de Dato C#	Tipo de Dato System	Tamaño en Bytes
String	System.String	20 (como mínimo)

Clase String

¿Qué es un String?

En C# es una colección de Tipos de Datos **char**

Es de Tipo Referencia (se crea en el HEAP)

string es un alias de **System.String**

Se recomienda el uso de **string** para referirse a un **objeto** y el uso de **System.String** para referirse a la **clase String**

Ejemplo:

string cadena = “Mundo”;

string mensaje = String.Format(“Hola {0}!", cadena);

Tipo de Dato C#	Tipo de Dato System	Tamaño en Bytes
string	System.String	20 (como mínimo)

Clase String

Uso de Caracteres Especiales en Cadenas

Para identificar un caracter Escape, hay dos alternativas:

Identificar el caracter Escape doblemente (“ \\ *equivale a* \ *)*

string pathArchivo = “\\\\servidor\\Archivo.pdf”;

O bien, utilizar el carácter arroba (“@”) para identificar una cadena literal, y usar el caracter Escape normalmente

string pathArchivo = @“\\servidor\Archivo.pdf”;

Clase String

Concatenar Cadenas

Los Strings son INMUTABLES

Una vez que un valor se asigna a un String, NO se puede cambiar. Se genera un NUEVO String.

La Concatenación repetitiva tienda a utilizar el 100% de la CPU….

Clase String

Concatenar Cadenas

```
string cadena1 = “Hola”;
string cadena2 = “Mundo”;
string cadenaFinal = cadena1 + “ “ + cadena2;
```

O bien, utilizar el método **String.Format**

```
string cadena1 = “Hola”;
string cadena2 = “Mundo”;
string cadenaFinal = String.Format("{0} {1}", cadena1, cadena2);
```

O resumiendo…

```
string cadena = String.Format("{0} {1}", “Hola”, “Mundo”);
```

O utilizando el método **String.Concat**

```
string cadena = ("{0}", String.Concat(“Hola ”, “Mundo”));
```

Clase String

Concatenar Cadenas

Para concatenar en una iteración (for, while, etc), se recomienda el uso de **StringBuilder**

```
StringBuilder cadena = new StringBuilder();

while (condicion)
{
    cadena.Append(valor);
}
```

Clase String

Cadenas Delimitadas

A partir de un String delimitado, se puede utilizar el método String.Split para obtener un Array de String, o bien, un string de una posición del string delimitado

string cadena = “1,2,3,5,8,13,21”;

string[] stringArray= cadena.Split(‘,’);

Acceder a elemento del array: stringArray[2]

O bien

string elemento = cadena.Split(‘,’)[2];

Clase String

Cadenas Delimitadas

A partir de un Array, se puede generar un String Delimitado con el método **String.Join**

string[] stringArray = { “uno", “dos", “tres" };

string stringDelimitado = String.Join(“,”, stringArray);

Licenciatura en Gestión Tecnológica

Programación Avanzada II

Formación Continua

Muchas Gracias

Ing. Mariano Juiz

2do Cuatrimestre (2015)