PROGRAMACIÓN II

SOBRECARGA

SOBRECARGA

CONTENIDOS

- Sobrecarga de métodos
  - Repaso
    - Métodos
    - Tipos de métodos
  - Concepto
    - Parámetros Opcionales
    - Sobrecarga
  - Reglas de sintaxis
  - Ejercicios
    - Resolución de llamadas
    - ¿Es una sobrecarga válida?
- Sobrecarga de Constructores
  - Repaso
  - Concepto y reglas
  - Ejemplo
  - Operador “This”

REPASO: MÉTODOS

¿Qué es un método?	Un bloque de código que contiene una serie de instrucciones que serán ejecutadas en la llamada al mismo.
¿Dónde pueden ser declarados?	En una clase o estructura.
¿Cómo se declaran?	[Nivel de Acceso] [Modificador Opcional] [Valor de Retorno] Nombre ([Parámetros]) { //Código }

REPASO: MÉTODOS

```
abstract class Motorcycle
{
    // Cualquiera puede llamarlo.
    public void StartEngine() {/* Sentencias del método aquí. */ }

    // Sólo las clases derivadas pueden llamarlo.
    protected void AddGas(int gallons) { /* Sentencias del método aquí. */ }

    // Las clases derivadas pueden sobrescribir la implementación del método.
    public virtual int Drive(int miles, int speed)
    {
        /* Sentencias del método aquí. */
        return 1;
    }

    // Las clases derivadas deben implementar este método.
    public abstract double GetTopSpeed();
}
```

REPASO: TIPOS DE MÉTODOS

Métodos Estáticos

- Son métodos de la clase o del tipo mismo, no de los objetos.
- Llevan el modificador “static” en su firma.
- Se los invoca usando el nombre de la clase.
- NO pueden acceder a los campos no-estáticos. No se puede usar el “this”.

Métodos de Instancia

- Son métodos del objeto.
- No llevan el modificador “static” en su firma.
- Se los invoca usando la variable del objeto.
- Pueden acceder a todos los campos no-estáticos del objeto a través de la palabra reservada “this”.

REPASO: TIPOS DE MÉTODOS

```
class Program
{
    public int Multiplicar(int x, int y) // Método de instancia de la clase Program.
    {
        int resultado = x * y;
        return resultado;
    }

    static void Main(string[] args) //Método estático
    {
        Program programObject = new Program();
        //Se debe crear una instancia de la clase para acceder al método.
        int resultado = programObject.Multiplicar(12, 12);
        Console.WriteLine(resultado);
        Console.ReadKey();
    }
}
```

REPASO: TIPOS DE MÉTODOS

```
class Program
{
    public static int Multiplicar(int x, int y) // Método estático de la clase Program.
    {
        int resultado = x * y;
        return resultado;
    }

    static void Main(string[] args) //Método estático
    {
        //No es necesario crear un objeto. Se invoca con el nombre de la clase.
        int resultado = Program.Multiplicar(12, 12);
        Console.WriteLine(resultado);
        Console.ReadKey();
    }
}
```

CONCEPTO: PARÁMETROS OPCIONALES

- Por defecto, los parámetros de un método son requeridos.
- Se puede asignar valores por defecto a los parámetros de un método, haciéndolos opcionales.
- Cuando se llama al método, si ningún argumento fue suministrado se utilizará el valor por defecto.
- Sirve para extender funcionalidad a un método sin modificar o romper las llamadas existentes.

CONCEPTO: PARÁMETROS OPCIONALES

```
public static string Suma(int requerido, int opcional = default(int), string descripcion = "Suma")
{
    return String.Format("{0}: {1} + {2} = {3}", descripcion, requerido,
        opcional, requerido + opcional);
}

static void Main(string[] args)
{
    //Los primeros dos usan argumentos posicionales (se sabe a qué parámetro corresponden por la posición).
    Console.WriteLine(Suma(10));
    Console.WriteLine(Suma(10, 2));
    /*El tercero usa un argumento nombrado para asignar un valor al parámetro "descripcion"
     al mismo tiempo que se omite el parámetro "opcional"*/
    Console.WriteLine(Suma(10, descripcion: "Suma con cero"));
    Console.ReadKey();
}
```

```
Suma: 10 + 0 = 10
Suma: 10 + 2 = 12
Suma con cero: 10 + 0 = 10
```

CONCEPTO: PARÁMETROS OPCIONALES

ALERTA

- ¡¡¡En esta materia **NO** está admitido el uso de los parámetros opcionales para la resolución de ejercicios, trabajos prácticos y/o exámenes!!!
- Se deberá usar sobrecarga.

CONCEPTO: SOBRECARGA

- Permite al programador definir métodos con el mismo nombre, pero distintos parámetros.
- Sirve para lo mismo que los parámetros opcionales:
  - Extender funcionalidad.
  - Unificar bajo un mismo nombre métodos similares que requieren parámetros diferentes.
- En **tiempo de compilación** se resolverá a cuál de todas las sobrecargas invocar según el tipo, número y/o orden de los argumentos utilizados en la llamada.

CONCEPTO: SOBRECARGA

```
public string Substring (int startIndex);
```

- Devuelve una subcadena de la instancia. La subcadena empieza en la posición especificada y continua hasta el final de la cadena.

```
public string Substring (int startIndex, int length);
```

- Devuelve una subcadena de la instancia. La subcadena empieza en la posición especificada y tiene una longitud determinada.

REGLAS DE SINTAXIS

Se admite sobrecarga cuando…	NO se admite sobrecarga cuando…
Se cambia el número de parámetros.	Cambia el tipo de retorno.
Se cambia el tipo de alguno de los parámetros.	Sólo se cambia el nombre de algún parámetro.
Se cambia el orden de los parámetros.	
Se cambia el modificador **out** / **ref**	

REGLAS DE SINTAXIS

¿Cómo evitamos repetir código?

- Dejamos que la versión más compleja haga todo el trabajo y la invocamos desde las versiones más simples.

REGLAS DE SINTAXIS

```
public static string Suma(
    int requerido,
    int opcional = default(int),
    string descripcion = "Suma")
{
    return String.Format(
        "{0}: {1} + {2} = {3}",
        descripcion, requerido,
        opcional, requerido + opcional);
}
```

```
public static string Suma(int requerido)
{
    return Suma(requerido, 0, "Suma");
}

public static string Suma(int requerido, int opcional)
{
    return Suma(requerido, opcional, "Suma");
}

public static string Suma(int requerido, string descripcion)
{
    return Suma(requerido, 0, descripcion);
}

public static string Suma(int requerido, int opcional, string descripcion)
{
    return String.Format("{0}: {1} + {2} = {3}", descripcion, requerido,
        opcional, requerido + opcional);
}
```

EJERCICIOS: RESOLUCIÓN DE LLAMADAS

```
public static string Suma(int requerido) // 1
{
    return Suma(requerido, 0, "Suma");
}

public static string Suma(int requerido, int opcional) // 2
{
    return Suma(requerido, opcional, "Suma");
}

public static string Suma(int requerido, string descripcion) // 3
{
    return Suma(requerido, 0, descripcion);
}

public static string Suma(int requerido, int opcional, string descripcion) // 4
{
    return String.Format("{0}: {1} + {2} = {3}", descripcion, requerido,
        opcional, requerido + opcional);
}
```

¿Qué método es llamado en cada sentencia del Main? (Unir letras con números)

```
static void Main(string[] args)
{
    Console.WriteLine(Suma(10, "Suma con cero")); //A
    Console.WriteLine(Suma(10)); //B
    Console.WriteLine(Suma(10, 5, "Suma con cero")); //C
    Console.WriteLine(Suma(10, 2)); //D
    Console.ReadKey();
}
```

EJERCICIOS: ¿ES UNA SOBRECARGA VÁLIDA?

```
// Ejercicio 1
public static string Suma(int x, int y)

public static int Suma(int x, int y)

// Ejercicio 2
public static int Suma(int x, int y)

public static int Suma(int num1, int num2)
```

EJERCICIOS: ¿ES UNA SOBRECARGA VÁLIDA?

```java
// Ejercicio 3
public int Suma(int x, int y)

public static int Suma(int num1, int num2)

// Ejercicio 4
public int Suma(int x, double y)

public int Suma(int x, int y)
```

EJERCICIOS: ¿ES UNA SOBRECARGA VÁLIDA?

```
// Ejercicio 5
public int Suma(int x, int y)

public int Suma(int x, int y, int z)

// Ejercicio 6
public string Suma(int x, int y, string descripcion)

public string Suma(string descripcion, int x, int y)
```

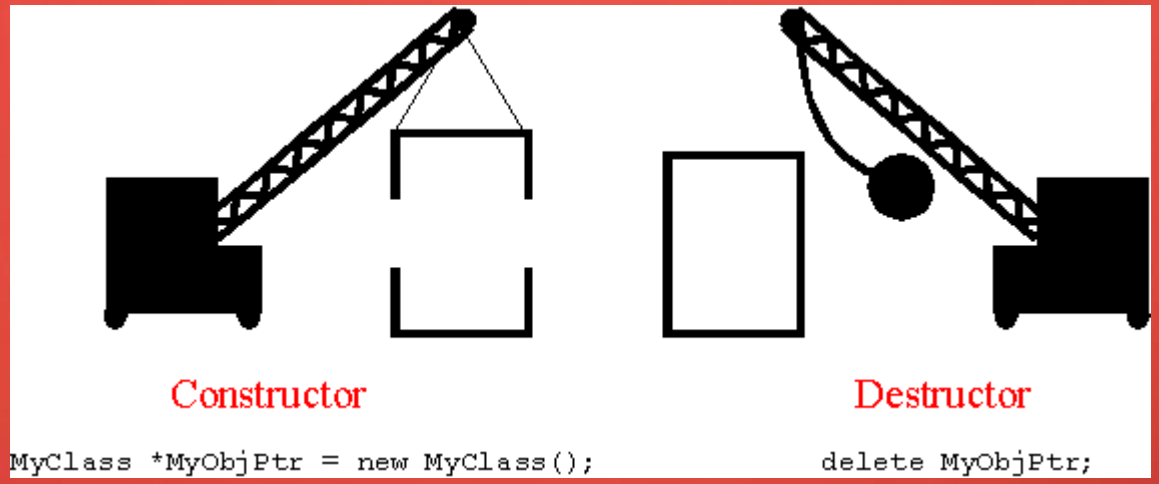
SOBRECARGA

CONTENIDOS

- Sobrecarga de métodos
  - Repaso
    - Métodos
    - Tipos de métodos
  - Concepto
    - Parámetros Opcionales
    - Sobrecarga
  - Reglas de sintaxis
  - Ejercicios
    - Resolución de llamadas
    - ¿Es una sobrecarga válida?
- Sobrecarga de Constructores
  - Repaso
  - Concepto y reglas
  - Ejemplo
  - Operador “This”

REPASO: CONSTRUCTORES

Constructores de Instancia	Constructor estático
Tienen el mismo nombre de la clase.	
Crean e inicializan los miembros de instancia del objeto.	Se utiliza para inicializar datos estáticos o para acciones que deban ser ejecutadas una única vez.
Son invocados por el operador “new” luego de que la memoria es asignada.	Es llamado automáticamente antes que la primera instancia sea creada o cualquier miembro estático sea referenciado.
Tienen modificadores de acceso y pueden tener parámetros.	No tiene modificadores de acceso ni parámetros.
Pueden ser llamados desde otros constructores y cuando instanciamos un nuevo objeto.	No puede ser llamado directamente.



GUÍA DE ESTUDIO: CONSTRUCTORES.

CONCEPTO Y REGLAS

- Los constructores se pueden sobrecargar de la misma forma que los métodos.
- La forma de hacerlo y reglas son las mismas.
- Sirve para instanciar objetos de formas diferentes.

EJEMPLO

DateTime(Int64)	Initializes a new instance of the DateTime structure to a specified number of ticks.
DateTime(Int64, DateTimeKind)	Initializes a new instance of the DateTime structure to a specified number of ticks and to Coordinated Universal Time (UTC) or local time.
DateTime(Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, and day.
DateTime(Int32, Int32, Int32, Calendar)	Initializes a new instance of the DateTime structure to the specified year, month, and day for the specified calendar.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second.
DateTime(Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)	Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and Coordinated Universal Time (UTC) or local time.

OPERADOR “THIS”

- Sirve para llamar a otros constructores de la misma clase.
- Evita repetir código.
- Sólo se pueden hacer llamadas entre constructores.
- Según el tipo, número y orden de los parámetros en el operador “this” se sabe a qué constructor llamar.

OPERADOR “THIS”

```
class Sobrecarga
{
    public Sobrecarga()
    { //… }

    public Sobrecarga(int n)
    { //… }

    public Sobrecarga(int n, int x)
    { //… }

    public Sobrecarga(string mensaje)
    { //… }
}
```

OPERADOR “THIS”

```
class Sobrecarga
{
    public Sobrecarga()
    { //… }

    public Sobrecarga(int n)
    { //… }

    public Sobrecarga(int n, int x) : this (n)
    { //… }

    public Sobrecarga(string mensaje)
    { //… }
}
```

PPT, CÓDIGO FUENTE, APUNTES, BIBLIOGRAFÍA Y GUÍA DE ESTUDIO EN:

GITHUB

github.com/mauricioCerizza/Material_Programacion_II