



Programación y Laboratorio II

Clase 02

Métodos y clases Estáticas

CONTENIDO

Clases y métodos estáticos

- Principio DRY
- Métodos estáticos
- Modificadores de acceso
- Buenas prácticas
- Clases estáticas
- Comentarios de documentación XML

Namespaces

- ¿Qué es un namespace?
- Directiva using
- Directiva alias

Trabajar con texto

- StringBuilder
- Strings interpolados
- Otros métodos de string

Funciones matemáticas

- Clase Math y sus métodos

01.

CLASES Y MÉTODOS ESTÁTICOS

Principio DRY

"Toda pieza de conocimiento debe tener una representación única, inequívoca y fidedigna dentro de un sistema."

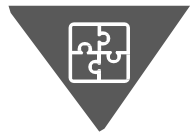


Principio DRY



PIEZA DE CONOCIMIENTO

Funcionalidad precisa
dentro del contexto de
negocio o un algoritmo
concreto.



UNICA

No debe existir
otra repetición



INEQUIVOCA

Sólo puede ser
interpretada,
entendida o
explicada de una
manera



FIDEDIGNA

Confiamos en
que es correcta

Principio DRY

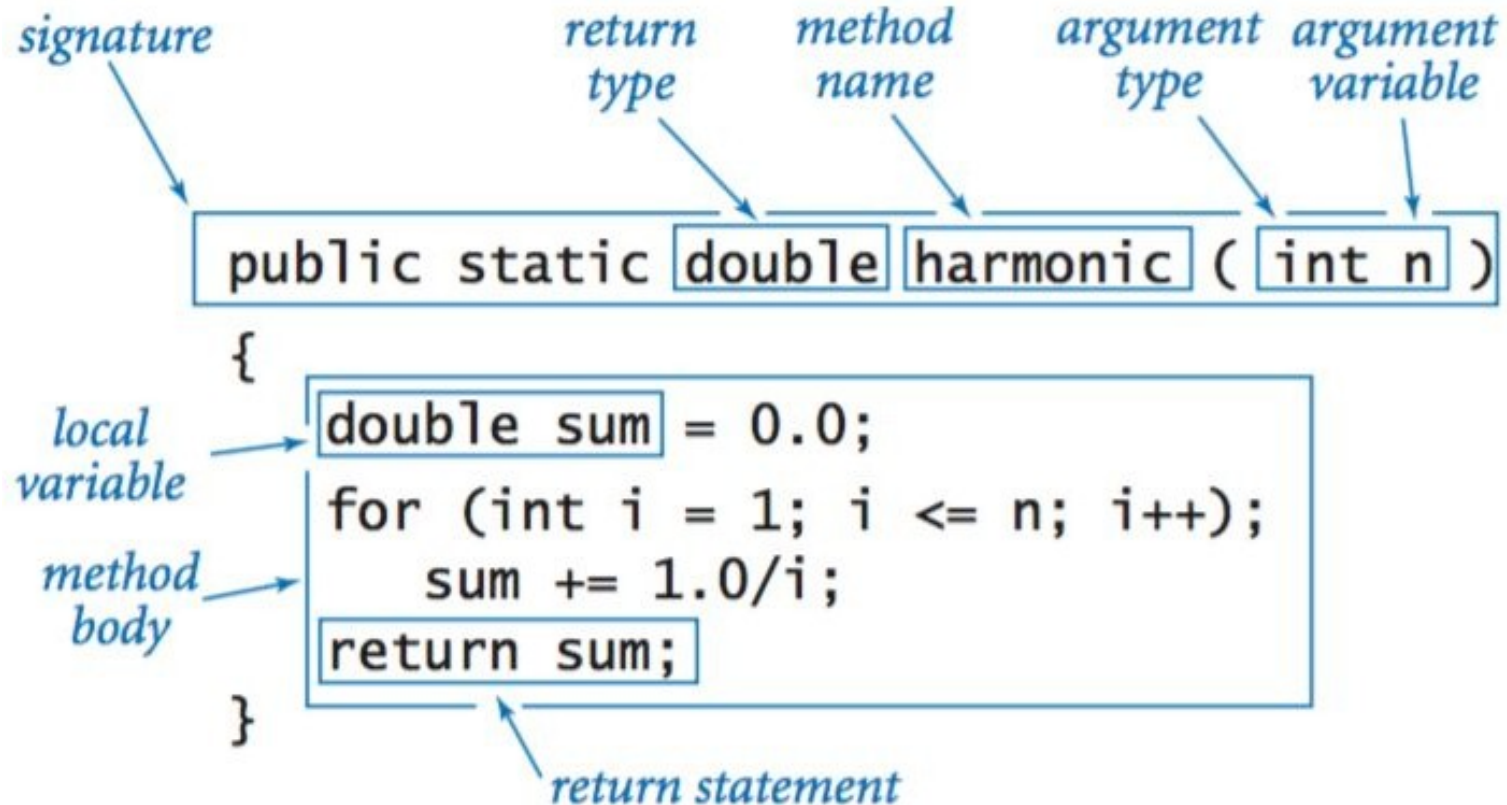
"Many people took it [the DRY principle] to refer to code only: they thought that DRY means "don't copy-and-paste lines of source." [...] DRY is about the duplication of knowledge, of intent. It's about expressing the same thing in two different places, possibly in two totally different ways."

The Pragmatic Programmer - Dave Thomas

La idea del principio *DRY* es simple:
Cuando ocurra un cambio no deberíamos necesitar actualizar múltiples cosas en paralelo



MÉTODOS ESTÁTICOS



MODIFICADORES DE ACCESO

	DESCRIPCIÓN
public	Accesible desde cualquier parte.
private	Sólo accesible desde dentro del mismo tipo.
internal	Sólo accesible desde dentro del mismo ensamblado / proyecto.
protected	Sólo accesible desde dentro del mismo tipo o tipos derivados (hijos).

BUENAS PRÁCTICAS métodos

LOS NOMBRES IMPORTAN

- Nombres descriptivos. Es preferible la legibilidad a la brevedad.
Verbos PascalCase. ToString()

HACER UNA COSA

- Solo deben hacer una cosa. Deben hacerlo bien y debe ser lo único que hagan

TAMAÑO REDUCIDO

- 20 líneas máximo con pocos niveles de anidamiento

ANIDAMIENTO

- Máximo dos niveles de anidamiento

CLASES ESTÁTICAS



```
1  public static class Cotizador
2  {
3      private const decimal pesosPorDolarComprado = 96.75M;
4      private const decimal pesosPorDolarVendido = 102.82M;
5
6      public static decimal CotizarVentaDolares(decimal montoDolaresAVender)
7      {
8          decimal costoEnPesos = montoDolaresAVender * pesosPorDolarVendido;
9
10         return costoEnPesos;
11     }
12
13     public static decimal DetizarCompraDolares(decimal montoDolaresAComprar)
14     {
15         decimal pagoEnPesos = montoDolaresAComprar * pesosPorDolarComprado;
16
17         return pagoEnPesos;
18     }
19 }
```

REGLAS DE CLEAN CODE

01

Regla del boy scout

Dejar el campamento más limpio de lo que se ha encontrado

02

Autor del código

“El código limpio se lee como prosa bien escrita”
funciones simples, claras y pequeñas

03

DRY

No pueden existir dos partes del programa que realicen la misma función.

04

Los comentarios mienten

Los códigos son modificados, los comentarios no.
Comentar solo lo necesario

05

Los nombre importan

Debe ser preciso y entregar la idea central

REFACTORIZAR

“Escribir software es como cualquier otro tipo de escritura. Cuando escribes un ensayo o un artículo, primero estructuras las ideas y después el mensaje hasta que se lee bien. El primer borrador puede ser torpe y desorganizado, así que lo retocas y mejoras hasta que se lea de la forma adecuada.

Cuando escribo funciones, suelen ser extensas y complicadas. Tienen muchas sangrías y bucles anidados. Tienen largas listas de argumentos. Los nombres son arbitrarios y hay un código duplicado. Pero también tengo una batería de test unitarios que cubren cada una de esas torpes líneas de código.

Entonces retoco el código, dividiendo funciones, cambiando nombres, eliminando la duplicación. Reduzco los métodos y los reordeno. A veces divido clases enteras, mientras mantengo las pruebas aprobatorias.

Al final, termino con funciones que siguen las reglas que he establecido en este capítulo. No los escribo así para empezar. No creo que nadie pueda”.



Robert Cecil Martin

BUENAS PRÁCTICAS

- 1 El nombre de la variable debe ser un nombre descriptivo que se identifique con el código
- 2 Las variables no se pueden iniciar con números
- 3 No se pueden usar caracteres especiales, ni es recomendable empezar una variable con mayúscula
- 4 Los tipos de escritura de una variable son: lowerCamelCase, UpperCamelCase, snake_case.
- 5 Tener en cuenta que hay nombres que no se pueden usar ya que se encuentran reservadas para el lenguaje.

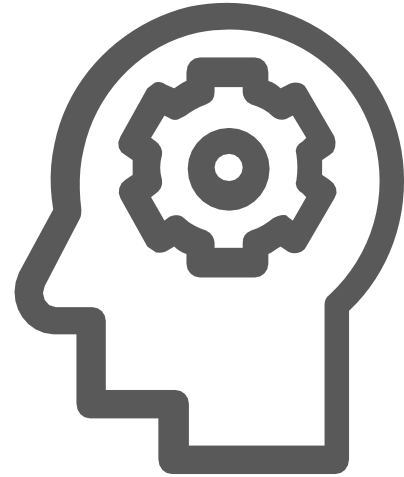
DOCUMENTACIÓN XML



```
1  /// <summary>
2  /// Cotiza la venta de un monto de dólares dado, retornando el costo en pesos.
3  /// </summary>
4  /// <param name="montoDolaresAVender">Cantidad de dólares que se desean vender.</param>
5  /// <returns>Costo en pesos de la venta.</returns>
6  public static decimal CotizarVentaDolares(decimal montoDolaresAVender)
7  {
8      decimal montoEnPesos = montoDolaresAVender * pesosPorDolarVendido;
9
10     return montoEnPesos;
11 }
```

EJERCICIOS

- I01 - Validador de rango
- I04 - La Calculadora
- A01 - Calcular un factorial



02.

NAMESPACES




¿Qué es un namespace?

Son una agrupación lógica de clases y otros elementos del código fuente.

Todo lo que declaremos deberá estar contenido dentro de un espacio de nombres.

Su función principal es la organización del código, permitiendo la reducción de conflictos por nombres duplicados y la posibilidad de trabajar en un mismo programa con componentes de distinta procedencia.



NAMESPACES



```
1 System.Console.WriteLine(";Hola mundo!");
```

DIRECTIVA USING

La directiva using permite la especificación de una llamada a un método sin el uso obligatorio de un nombre completamente cualificado (nombre completo incluyendo el espacio de nombres).

```
1  using System; //Directiva USING
2
3  public class Program
4  {
5      public static void Main()
6      {
7          Console.WriteLine("¡Hola mundo!");
8      }
9  }
```

DIRECTIVA ALIAS

permite utilizar un nombre distinto para un espacio de nombres. Se suele utilizar para abreviar nombres largos.

```
1  using SC = System.Console; //Directiva ALIAS
2
3  public class Program
4  {
5      public static void Main()
6      {
7          SC.WriteLine("¡Hola mundo!");
8      }
9  }
```

03.

STRING

STRINGBUILDER

USO

Append	Agrega información al final del StringBuilder actual
AppendLine	Agrega información al final del StringBuilder actual con un salto de línea
AppendFormat	Reemplaza el formato pasado en un string con texto formateado.
Insert	Inserta un string en el índice especificado del StringBuilder actual .

04.

MATH

EJERCICIOS

- I05 - Aprendete las tablas
- I06 - Calculadora de áreas
- I07 - Pitágoras estaría orgulloso
- I08 - El tiempo pasa

