Clase 8 · Review

Martes, 26 Septiembre 2023

Indice

Tipos de datos

-
```
'int'    = enteros
'float'  = reales (16 bits)
'double' = reales (32 bits)
'string' = cadenas de texto
'char'   = caracteres
'enum'   = enumerations
'objeto' = tipo de un objeto
'funcion'= comportamiento del programa
'metodo' = funciones propias de un objeto de clase
```

Operadores

-
```
'+' = suma
'-' = resta
'*' = producto
'/' = division
'%' = modulo
```

Declaraciones, operadores, comparadores

-
```
'int a'   declaración de variable 'a'
'a = 5'   inicialización de variable


'a >= 3'  mayor o igual
'a < 3' menor o igual


'a == 3'  comparador de igualdad
'a != 3'  comparador de desigualdad


'a > 3 && a < 9'  Operador AND


TABLA DE VERDAD
True && True devuelve True
True && False devuelve False
False && True devuelve False
False && False devuelve False
'a != 3'  comparador de desigualdad
'a > 3 || a < 9'  Operador OR


TABLA DE VERDAD
True && True devuelve True
True && False devuelve True
False && True devuelve True
False && False devuelve False
```

Cálculo de porcentajes

- El porcentaje de un número sigue la siguiente regla:
  Parte / Todo * (tipo de tanto)

- Porcentaje en tanto por **cien**:
  Parte / Todo * (100)
- Porcentaje en tanto por **uno**:
  Parte / Todo
  Tiene la ventaja que te indica inmediatamente la proporción de las partes.

- Para aumentar un número un 20 por ciento utilizamos la siguiente fórmula:
  número * (1 + 20 / 100)
- Para disminuir un número un 20 por ciento utilizamos la siguiente fórmula:
  número * (1 - 20 / 100)

Funciones

- 'funcion'= comportamiento del programa
  'metodo' = funciones propias de un objeto de clase

Composición de una función

- Declaración de variables
  Asignaciones
  Condicionales
  Bucles
  Rupturas
  Retornos

Condicionales

- Instrucción para ser evaluada

```
'
while(condicion)
{
    código a realizar
}
'


'
if(condicion)
{
    código a realizar
}
else
{
    código a realizar
}
'

'
if(condicion 1)
{
    código a realizar
}
if(condicion 2)
{
    código a realizar
}
else
{
    si la condicion 2 NO se cumple, entonces código a realizar
    si la condicion 1 SI se cumple, pero la 2 NO se cumple,
    entonces codigo a realizar
}
'
Estas condiciones tienen el problema que no tienen en cuenta el primer if


if / else if
'
if(condicion 1)
{
    código a realizar
}
else if(condicion 2)
{
    código a realizar
}
else
{
    si la condicion 1 NO se cumple, y
    si la condicion 2 NO se cumple, entonces código a realizar
    entonces codigo a realizar
}
'

Este tipo de condicional tiene la misma estructura que la siguiente:

'
if(condicion 1)
{
    código a realizar
}
else if(condicion 2)
{
    if(condicion 2)
```

```
    {
        código a realizar
    }

    else
    {
        si la condicion 1 NO se cumple, y
        si la condicion 2 NO se cumple, entonces código a realizar
        entonces codigo a realizar
    }
}
'
```

- Operador Ternario: Realiza una comprobacion binaria en un misma linea

```
'
(condicion) ? expresion true : expresion false
'
```

Establecemos una condicion con dos posibilidades, si es true o si es false

Bucles

-

```
'
while(condicion)
{
    código a realizar
}
'
```

- Un bucle **while** se repite mientras se cumpla la condicion del parametro.

-

```
'
for(sentencia inicial; condicion; sentencia final)
{
    código a realizar
}
'
```

- Un bucle **for** se repite un número determinado de veces. Podemos utilizarlo siempre que conozcamos el numero exacto de iteraciones

Rupturas de bucle

- **break;**
  La linea de compilación salta justo al final del bucle.
- **continue;**
  La linea de compilación salta justo al inicio del bucle, sin tener en cuenta lo que resta del bucle.

Rupturas de bucle

- **break;**
  La linea de compilación salta justo al final del bucle.

Notas para tener en cuenta

- Jamás usaremos un Console.WriteLine() para hacer funciones. Excepto si nos piden explicitamente que la función imprima por pantalla.
- Jamás usaremos un return dentro de un bucle for. No tiene sentido terminar el bucle antes de que la función termine de iterar entre los valores definidos.
- Una función debe ocupar como máximo el tamaño de una pantalla. Para entenderla mejor y para mantenerla sintética.
- Es muy aconsejable dividir nuestro programa en piezas pequeñas que conforman unidades lógicas. Sistematizar el programa, crear Sistemas Funcionales que siguen una lógica. Crear el programa como las partes que son más que el todo.
- *Code Snippet*: Trozos de código que realizan una tarea demostrativa.
- Todo nuestro programa es reutilizable. Es muy positivo utilizar funciones que hemos diseñado en otras zonas de nuestro programa.

Funciones de Objeto

- Funciones de Objeto de la clase Dolphin

```
'
public class Dolphin
{
    public double life;

    public double GetLife()
    {
        return life/10;
```

```
    }
}

public class Main
{
    Dolphin d1 = new Dolphin();
    double resultLife = d1.GetLife();
}
'
```

Funciones de objeto

- Las funciones de objeto explican el comportamiento de las instancias de una clase.
- Se diseñan en el interior de la clase. Se caracterizan por no llevar 'static'. Igualmente es raro que necesiten parámetros para funcionar.
- Para invocarlas, debemos utilizar la notación por punto: d1.GetLife(). Esto significa que la función es invocada por el objeto 'd1'
- Podemos almacenar los valores resultantes de las funciones en variables que creamos a propósito en la clase Main de nuestro programa.

Orden dentro de la función

-
```
'enum' enumerations
'Atributos' Atributos de clase
'Properties' Getters y Setters
'Constructor' Inicializacion de instancias de Objeto
'Metodos' Funciones propias del Objeto
'Invocaciones'
```

Funciones de Clase

- Algunas Funciones de Clase que realizan tareas dentro de nuestro programa.

```
'
public class Utils
{
    public static double CalculateSum(double a, double b)
    {
        return a + b;
    }
}

public class Functions
{
    public static bool IsMajor(int a, int b)
    {
        return (a > b);
    }
}
'
```

Funciones de Clase

- Las funciones de clase son pequeños fragmentos de código que por si mismas realizan una función.
- Pueden diseñarse dentro de una clase creada especificamente para ellas, como 'Functions', 'Utils'
- Para utilizarlas, debemos invocarlas desde otro lugar de nuestro programa. Esto se realiza con la siguiente sintaxis:
  '(nombre de clase).(nombre de funcion)(parametros)'
  'Utils.GetSum(5, 6)'
- Como regla general , siempre devuelven un valor que luego podemos almacenar en una variable. Estas variables se usan en otras funciones o clases de nuestro programa.
- Siempre debe contener un return.
- Cada función es una pieza atómica de nuestro programa. Es conveniente que todas las piezas formen el conjunto funcional del programa.

Ejemplos de Clases tipo

- ```
  // FUNCIONES DE CLASE

  '
  public class Game
  {
      public static (tipo) (nombreFuncion)
  }
  '

  // FUNCIONES DE OBJETO

  '
  public class Student
  {
      public string name;
  }
  '
  ```

- **Palabras clave**
  Funciones de Clase
  Clase = Funciones = static = retorno = void = Categoria = Enseñanzas

- Funciones de Objeto
  Objeto = (no)static = (sin)retorno = Instancias = Objetos = registro = parametros
  ```