**Seminario de Lenguajes (.NET)**

**Práctica 3**

1. Ejecute y analice cuidadosamente el siguiente programa:

**using** System;

class Programa{

static void **Main**(){

**for**(**int** i = 0; i < Console.WindowHeight; i++ ){

Console.CursorLeft = Console.WindowWidth - 1 - i;

Console.CursorTop = i\*2;

Console.**Write**(i);

}

Console.CursorLeft=0;

Console.CursorVisible = **false**;

Console.**WriteLine**("Presione una tecla para finalizar");

Console.**ReadKey**(**true**);

}

}

¿Cuál es el efecto de cambiar el valor de las propiedades WindowHeight, WindowWidth, CursorLeft, CursorTop y CursorVisible de la clase Console?

Efecto al cambiar propiedades de la clase Console:

* WindowsHeight: Largo de la ventana consola
* WindowsHeight: Ancho de la ventana consola
* CursorLeft: Posicion donde escribe el cursor (horizontal). Al incrementar el parámetro “i” nos acercamos al lado izquierdo de la consola.
* CursorTop: Posicion donde escribe el cursor (vertical). A medida que se incrementa se aumenta el espacio vertical entre los valores impresos.
* CursorVisible: se oculta con “false”, se muestra en “true”.

2. Ejecute y analice cuidadosamente el siguiente programa:

**using** System;

class Programa{

static void **Main**(){

Console.CursorVisible=**false**;

ConsoleKeyInfo k = Console.**ReadKey**(**true**);

**while**(k.Key != ConsoleKey.End){

Console.**Clear**();

Console.**Write**("{0,-10} {1,-10} {2,-10}",k.Modifiers, k.Key, k.KeyChar);

k = Console.**ReadKey**(**true**);

}

}

}

Compruebe tipeando teclas y modificadores (shift, ctrl, alt) para apreciar de qué manera se puede acceder a esta información en el código del programa.

**Para saber que modificador fue utilizado podríamos hacer:  
if(k.Modifiers.ToString().Equals("Control")) 🡪 Realizo operación**

**El modificador puede ser “Control”, “Alt” o “Shift”.**

7. ¿Qué salida por la consola produce el siguiente código?

**int** x=0;

**try**{

Console.**WriteLine**(1.0/x);

Console.**WriteLine**(1/x);

Console.**WriteLine**("Ok");

}**catch**{

}

¿Qué se puede inferir respecto de la excepción división por cero en relación al tipo de los operandos?

Console.**WriteLine**(1.0/x); 🡪 Imprime un 8

Console.**WriteLine**(1/x); 🡪 Excepción, sale del bloque try

Console.**WriteLine**("Ok"); 🡪 Solo se muestra en caso de comentar el 1/x

La salda por consola para el código es un “8”. ????

**8.** Compruebe el funcionamiento del siguiente programa:

**using** System;

class Ejercicio3 {

static void **Main**(string[] args) {

**try**{

**metodo1**();

}**catch**{

Console.**WriteLine**("método 1 propagó una excepción no tratada");

}

Console.**ReadLine**();

}

static void **metodo1**() {

**byte** b=255;

**try**{

b++;

}**finally**{

Console.**WriteLine**("bloque finally");

}

}

}

Se produce un overflow al sumar la variable b, lo que produce la siguiente salida por consola:

Bloque finally

Metodo 1 propago una excepción no tratada

10. Consulte la documentación sobre la clase BitArray. Compile y ejecute el siguiente programa:

**using** System;

**using** System.Collections;

class Program

{

**public** static void **Main**(string[] args){

**int**[] vectorDeInt = **new int**[] {5,255};

Console.**WriteLine**("BitArray construido a partir " +

"del vector de enteros {5,255}");

BitArray ba1=**new BitArray**(vectorDeInt);

**for**(**int** i=0;i<ba1.Count;i++){

Console.**WriteLine** ("posicion {0,2} valor {1}", i, ba1[i]);

}

**byte**[] vectorDeByte = **new byte**[] {5,255};

Console.**WriteLine**("BitArray construido a partir " +

"del vector de bytes {5,255}");

BitArray ba2=**new BitArray**(vectorDeByte);

**for**(**int** i=0;i<ba2.Count;i++){

Console.**WriteLine** ("posicion {0,2} valor {1}", i, ba2[i]);

}

Console.**ReadKey**(**true**);

}

}

¿Observó la diferencia en la creación de ambos objetos Bitarray? ¿Dónde se colocan los bits más significativos?

Salida por consola:

|  |  |  |
| --- | --- | --- |
| BitArray construido a partir del vector de enteros {5,255} | | BitArray construido a partir  del vector de bytes {5,255} |
| posición 0 valor True  posición 1 valor False  posición 2 valor True  posición 3 valor False  posición 4 valor False  posición 5 valor False  posición 6 valor False  posición 7 valor False  posición 8 valor False  posición 9 valor False  posición 10 valor False  posición 11 valor False  posición 12 valor False  posición 13 valor False  posición 14 valor False  posición 15 valor False  posición 16 valor False  posición 17 valor False  posición 18 valor False  posición 19 valor False  posición 20 valor False  posición 21 valor False  posición 22 valor False  posición 23 valor False  posición 24 valor False  posición 25 valor False  posición 26 valor False  posición 27 valor False  posición 28 valor False  posición 29 valor False  posición 30 valor False  posición 31 valor False | posición 32 valor True  posición 33 valor True  posición 34 valor True  posición 35 valor True  posición 36 valor True  posición 37 valor True  posición 38 valor True  posición 39 valor True  posición 40 valor False  posición 41 valor False  posición 42 valor False  posición 43 valor False  posición 44 valor False  posición 45 valor False  posición 46 valor False  posición 47 valor False  posición 48 valor False  posición 49 valor False  posición 50 valor False  posición 51 valor False  posición 52 valor False  posición 53 valor False  posición 54 valor False  posición 55 valor False  posición 56 valor False  posición 57 valor False  posición 58 valor False  posición 59 valor False  posición 60 valor False  posición 61 valor False  posición 62 valor False  posición 63 valor False | posición 0 valor True  posición 1 valor False  posición 2 valor True  posición 3 valor False  posición 4 valor False  posición 5 valor False  posición 6 valor False  posición 7 valor False  posición 8 valor True  posición 9 valor True  posición 10 valor True  posición 11 valor True  posición 12 valor True  posición 13 valor True  posición 14 valor True  posición 15 valor True |

Se puede observar como el entero es de 32 bits, por lo tanto el BitArray tendrá 64 posiciones, mientras que si es de bytes cada numero estará compuesto solo por 8 bits, por lo que el BitArray tendrá 16 posiciones. Teniendo en cuenta que la cantidad de bytes limita el rango.

Para el caso del Byte, el bit menos significativo para el primer valor (5) es la posición 0 del array, y el más significativo será la posición 7. Para el caso del integer, la posición más significativa será la 31.

11. ¿Cómo escribiría la representación binaria de un char utilizando BitArray? Recuerde que puede hacer casting para transformar un char en un byte, por ejemplo (**byte**)'A' devuelve un byte con el valor 65.

byte[]letras=new byte[]{(byte)'A',(byte)'B'}; 🡪 El de más arriba es el bit mas significativo

BitArray ba= new BitArray(letras); 🡪 Lo muestro por consola

Salida por consola:

|  |
| --- |
| posición 0 valor True  posición 1 valor False  posición 2 valor False  posición 3 valor False  posición 4 valor False  posición 5 valor False  posición 6 valor True  posición 7 valor False  posición 8 valor False  posición 9 valor True  posición 10 valor False  posición 11 valor False  posición 12 valor False  posición 13 valor False  posición 14 valor True  posición 15 valor False |

16. Verifique si la sección opcional [:*formatString*] de formatos compuestos redondea o trunca en el caso de la impresión de números reales cuando se restringe la cantidad de decimales.

Según la teoría (no verificado en código) lo que hace es redondear cuando se tienen más decimales de los que se muestran.