

## Seminario de Lenguajes (.NET)

### Práctica 4

**Ejercicios obligatorios para el coloquio:** Los **ejercicios 9 y 10** de esta práctica forman parte del conjunto de ejercicios de programación obligatorios que el alumno debe resolver y exponer de manera oral sobre máquina el día del coloquio hacia final de la cursada.

1) Codifique la clase **Hora** de tal forma que al ejecutar el siguiente programa:

```
class Program
{
    public static void Main(string[] args)
    {
        Hora h=new Hora(23,30,15);
        h.imprimir();
        Console.ReadKey(true);
    }
}
```

se imprima por consola:

23 HORAS, 30 MINUTOS Y 15 SEGUNDOS

2) Agregue un segundo constructor a la clase **Hora** para que pueda especificarse por un solo parámetro que indique la cantidad de horas (puede tener decimales). Así el siguiente código:

```
class Program
{
    public static void Main(string[] args)
    {
        Hora h1=new Hora(23,30,15);
        Hora h2=new Hora(14.43);
        h1.imprimir();
        h2.imprimir();
        Console.ReadKey(true);
    }
}
```

produce la siguiente salida por consola:

23 HORAS, 30 MINUTOS Y 15 SEGUNDOS  
14 HORAS, 25 MINUTOS Y 47 SEGUNDOS

3) Codificar una clase llamada **Ecuacion2** para representar una ecuación de 2º grado. Esta clase debe tener 3 campos privados, los coeficientes a, b y c de tipo **double**. La única forma de establecer los valores de estos campos será en el momento de la instanciación de un objeto **Ecuacion2**.

Codificar los siguientes métodos:

- **GetDiscriminante()**: devuelve el valor del discriminante (**double**), el discriminante tiene la siguiente formula,  $(b^2) - 4*a*c$ .
- **GetCantidadDeRaices()**: devuelve 0, 1 ó 2 dependiendo de la cantidad de raíces reales que posee la ecuación. Si el discriminante es negativo no tiene raíces reales, si el discriminante es cero tiene una única raíz, si el discriminante es mayor que cero posee 2 raíces reales.
- **ImprimirRaices()**: imprime la única o las 2 posibles raíces reales de la ecuación. En caso de no poseer soluciones reales debe imprimir una leyenda que así lo especifique.

4) Defina una clase **Persona** con 3 campos públicos: **Nombre**, **Edad** y **DNI**. Escriba un algoritmo que permita al usuario ingresar en una consola una serie de datos de la forma "Nombre<TAB>Documento<TAB>Edad<ENTER>". Una vez finalizada la entrada de datos, el programa debe imprimir en la consola un listado el listado con la forma:

Nro.) Nombre (Edad) <TAB> DNI.

Ejemplo:

```
1) Juan Perez (40)          2098745
2) José García (41)        1965412
...
```

- a) Utilizando un arreglo de **Personas**. Antes de comenzar con la carga el usuario debe ingresar por consola la cantidad de personas que va cargar.
- b) Utilizando un **ArrayList**. En este caso el usuario no debe ingresar la cantidad de personas que va a cargar, simplemente el proceso de entrada finaliza con un **string** vacío.

NOTA: Puede utilizar:

```
Console.SetIn(new System.IO.StreamReader(nombreDeArchivo));
```

para cambiar la entrada estándar de la consola y poder leer directamente desde un archivo de texto.

5) Modifique el programa anterior haciendo privados todos los campos. Defina un constructor adecuado y un método público **imprimir()** que escribe en la consola los campos del objeto con el formato requerido para el listado. En el constructor utilice parámetros cuyos nombres coincidan con los campos privados del objeto

6) Modifique el programa anterior definiendo un método público **cumpleaños** que incremente en uno la edad de la persona. Utilícelo para recorrer el arreglo de personas e incrementar en uno la edad de todos cuyos nombres comienzan con vocal.

7) Modifique el programa anterior agregando un campo privado **FechaNacimiento**. Defina un nuevo constructor que reciba como parámetros el nombre, la fecha de nacimiento (un **DateTime**) y el **dni**. Utilice ambos constructores para resolver el ejercicio, permitiendo ahora que el usuario tipee las entradas como:

```
"Nombre<TAB>Documento<TAB>Edad<ENTER>"
```

o bien:

```
"Nombre<TAB>Documento<TAB>fecha de nacimiento<ENTER>"
```

Nota: Utilice un método estático que pueda llamar desde **Main()** que reciba el string tipeado por el usuario y devuelva en parámetros de salida el nombre, el dni, la edad y la fecha de nacimiento.

8) Agregue a la clase **Persona** un método **esMayorQue(Persona p)** que devuelva verdadero si la persona que recibe el mensaje es de más edad que la persona recibida como parámetro. Utilícelo para realizar un programa que compare la edad de la primera persona del arreglo respecto de todas las demás imprimiendo en la consola el resultado de dicha comparación.

9) Modelar un árbol binario de búsqueda de valores enteros (sin valores duplicados). Desarrollar los métodos métodos:

1. **Insertar(valor)**: Inserta valor en el árbol descartándolo en caso que ya exista.
2. **RecorridoInorden**: devuelve un vector con los valores ordenados en forma creciente.
3. **Altura**: devuelve la altura del árbol.
4. **CantNodos**: devuelve la cantidad de nodos que posee el árbol.
5. **ValorMáximo**: devuelve el valor máximo que contiene el árbol.
6. **ValorMínimo**: devuelve el valor mínimo que contiene el árbol.

10) Implemente la clase **Matriz** que se utilizará para trabajar con matrices matemáticas. Implemente los dos constructores y todos los métodos que se detallan a continuación:

```
public Matriz(int filas, int columnas)
public Matriz(double[,] matriz)
public void SetElemento(int fila, int columna, double elemento)
public double GetElemento(int fila, int columna)
public void imprimir()
public void imprimir(string formatString)
public double[] GetFila(int fila)
public double[] GetColumna(int columna)
public double[] GetDiagonalPrincipal()
public double[] GetDiagonalSecundaria()
public double[][] getArregloDeArreglo()
public void sumarle(Matriz m)
public void restarle(Matriz m)
public void multiplicarPor(Matriz m)
```

A modo de ejemplo observe la salida del siguiente fragmento de código:

```
public static void Main(string[] args)
{
    Matriz A=new Matriz(2,3);
    for(int i=0;i<6;i++) A.SetElemento(i/3,i%3,(i+1)/3.0);
    Console.WriteLine("Impresión de la matriz A");
    A.imprimir("0.000");

    double[,] aux=new double[,] {{1,2,3},{4,5,6},{7,8,9}};
    Matriz B=new Matriz(aux);
    Console.WriteLine("\nImpresión de la matriz B");
    B.imprimir();

    Console.WriteLine("\nAcceso al elemento B[1,2]={0}",B.GetElemento(1,2));

    Console.Write("\nfila 1 de A: ");
    foreach(double d in A.GetFila(1)) Console.Write("{0:0.0} ",d);

    Console.Write("\n\nColumna 0 de B: ");
    foreach(double d in B.GetColumna(0)) Console.Write("{0} ",d);

    Console.Write("\n\nDiagonal principal de B: ");
    foreach(double d in B.GetDiagonalPrincipal()) Console.Write("{0} ",d);

    Console.Write("\n\nDiagonal secundaria de B: ");
    foreach(double d in B.GetDiagonalSecundaria()) Console.Write("{0} ",d);

    A.multiplicarPor(B);
    Console.WriteLine("\n\nA multiplicado por B");
    A.imprimir();
}
```

```

C:\Documents and Settings\Administrador\Mis documentos\SharpDevelop Projects...
Impresión de la matriz A
0,333 0,667 1,000
1,333 1,667 2,000

Impresión de la matriz B
1 2 3
4 5 6
7 8 9

Acceso al elemento B[1,2]=6
fila 1 de A: 1,3 1,7 2,0
Columna 0 de B: 1 4 7
Diagonal principal de B: 1 5 9
Diagonal secundaria de B: 3 5 7

A multiplicado por B
10 12 14
22 27 32

```

11) Preste atención a los siguientes programas (ninguno funciona correctamente):

Programa 1	Programa 2
<pre> class Program1{     static Auto a;     static void Main(){         a.Velocidad=10;     } } class Auto{     public double Velocidad; } </pre>	<pre> class Program2{     static void Main(){         Auto a;         a.Velocidad=10;     } } class Auto{     public double Velocidad; } </pre>

¿Qué puede decir acerca de la inicialización de los objetos? ¿En qué casos son inicializados automáticamente y con qué valor?

12) ¿Qué puede decir en relación a la sobrecarga de métodos en este ejemplo?

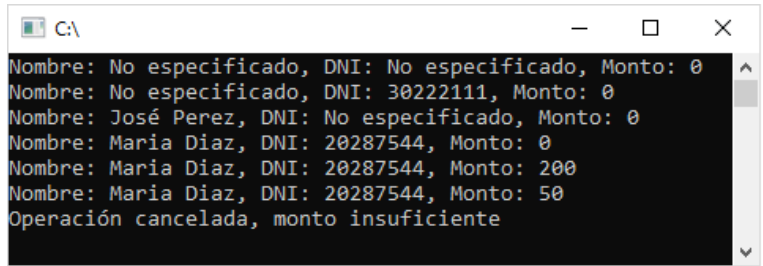
```

class A{
    public void Metodo(short n){
        Console.WriteLine("short {0} - ",n);
    }
    public void Metodo(int n){
        Console.WriteLine("int {0} - ",n);
    }
    public int Metodo(int n){
        return n+10;
    }
}

```

13) Complete la clase **Cuenta** para que el siguiente código produzca la salida indicada:

```
Cuenta cuenta = new Cuenta();
cuenta.Imprimir();
cuenta = new Cuenta(30222111);
cuenta.Imprimir();
cuenta = new Cuenta("José Perez");
cuenta.Imprimir();
cuenta = new Cuenta("Maria Diaz",20287544);
cuenta.Imprimir();
cuenta.Depositar(200);
cuenta.Imprimir();
cuenta.Extraer(150);
cuenta.Imprimir();
cuenta.Extraer(1500);
```



```
C:\
Nombre: No especificado, DNI: No especificado, Monto: 0
Nombre: No especificado, DNI: 30222111, Monto: 0
Nombre: José Perez, DNI: No especificado, Monto: 0
Nombre: Maria Diaz, DNI: 20287544, Monto: 0
Nombre: Maria Diaz, DNI: 20287544, Monto: 200
Nombre: Maria Diaz, DNI: 20287544, Monto: 50
Operación cancelada, monto insuficiente
```

```
class Cuenta
{
    private double monto;
    private int titularDNI;
    private string titularNobre;

    ...

}
```

Utilice en la medida de lo posible la sintaxis **:this** en el encabezado de los constructores para invocar a otro constructor ya definido.