

Seminario de Lenguajes (.NET)

Práctica 4

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

| Nro) | Nombre | Edad | DNI. |
|------|--------|------|------|
|------|--------|------|------|

Ejemplo de listado por consola:

| | | | |
|-----|-------------|----|---------|
| 1) | Juan Perez | 40 | 2098745 |
| 2) | José García | 41 | 1965412 |
| ... | | | |

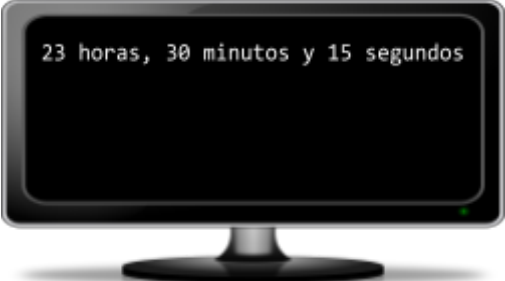
NOTA: Puede utilizar: `Console.SetIn(new System.IO.StreamReader(nombreDeArchivo));` para cambiar la entrada estándar de la consola y poder leer directamente desde un archivo de texto.

2) Modificar el programa anterior haciendo privados todos los campos. Definir un constructor adecuado y un método público **Imprimir()** que escriba en la consola los campos del objeto con el formato requerido para el listado.

3) Agregar a la clase **Persona** un método **EsMayorQue(Persona p)** que devuelva verdadero si la persona que recibe el mensaje tiene más edad que la persona enviada como parámetro. Utilizarlo para realizar un programa que devuelva la persona más joven de la lista.

4) Codificar la clase **Hora** de tal forma que el siguiente código produzca la salida por consola que se observa.

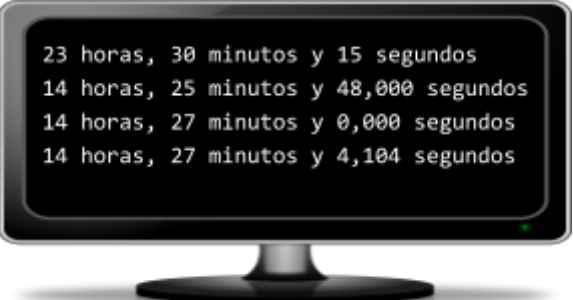
```
Hora h = new Hora(23,30,15);  
h.Imprimir();
```



23 horas, 30 minutos y 15 segundos

5) Agregar un segundo constructor a la clase **Hora** para que pueda especificarse la hora por medio de un único valor que admita decimales. Por ejemplo 3,5 indica la hora 3 y 30 minutos. Si se utiliza este segundo constructor, el método imprimir debe mostrar los segundos con tres dígitos decimales. Así el siguiente código debe producir la salida por consola que se observa.

```
new Hora(23, 30, 15).Imprimir();  
new Hora(14.43).Imprimir();  
new Hora(14.45).Imprimir();  
new Hora(14.45114).Imprimir();
```



23 horas, 30 minutos y 15 segundos
14 horas, 25 minutos y 48,000 segundos
14 horas, 27 minutos y 0,000 segundos
14 horas, 27 minutos y 4,104 segundos

6) 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.

7) Codificar la clase **Nodo** de un árbol binario de búsqueda de valores enteros. Un árbol binario de búsqueda no tiene valores duplicados y el valor de un nodo cualquiera es mayor a todos los valores de su subárbol izquierdo y es menor a todos los valores de su subárbol derecho.

Desarrollar los métodos:

1. **Insertar(valor)**: Inserta valor en el árbol descartándolo en caso que ya exista.
2. **GetInorden()**: devuelve un ArrayList con los valores ordenados en forma creciente.
3. **GetAltura()**: devuelve la altura del árbol.
4. **GetCantNodos()**: devuelve la cantidad de nodos que posee el árbol.
5. **GetValorMáximo()**: devuelve el valor máximo que contiene el árbol.
6. **GetValorMínimo()**: devuelve el valor mínimo que contiene el árbol.

A modo de ejemplo, el siguiente código debe arrojar por consola la salida que se muestra.

```
Nodo n = new Nodo(7);
n.Insertar(8);
n.Insertar(10);
n.Insertar(3);
n.Insertar(1);
n.Insertar(5);
n.Insertar(14);
foreach (int i in n.GetInOrder())
{
    Console.Write(i + " ");
}
Console.WriteLine();
Console.WriteLine(n.GetAltura());
Console.WriteLine(n.GetCantNodos());
Console.WriteLine(n.GetValorMaximo());
Console.WriteLine(n.GetValorMinimo());
```



8) Implementar la clase **Matriz** que se utilizará para trabajar con matrices matemáticas. Implementar 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)
```

9) Prestar atención a los siguientes programas (ninguno funciona correctamente):

```
Foo f = new Foo();
f.Imprimir();

class Foo
{
    string? _bar;
    public void Imprimir()
    {
        Console.WriteLine(_bar.Length);
    }
}
```

```
Foo f = new Foo();
f.Imprimir();

class Foo
{
    public void Imprimir()
    {
        string? _bar;
        Console.WriteLine(_bar.Length);
    }
}
```

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

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

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

11) Qué salida produce en la consola el siguiente programa?

```
object o = 5;
Sobrecarga s = new Sobrecarga();
s.Procesar(o, o);
s.Procesar((dynamic)o, o);
s.Procesar((dynamic)o, (dynamic)o);
s.Procesar(o, (dynamic)o);
s.Procesar(5, 5);

class Sobrecarga
{
    public void Procesar(int i, object o)
    {
        Console.WriteLine($"entero: {i}    objeto:{o}");
    }
    public void Procesar(dynamic d1, dynamic d2)
    {
        Console.WriteLine($"dynamic d1: {d1}    dynamic d2:{d2}");
    }
}
```

12) Completar 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);
```



```
class Cuenta
{
    private double _monto;
    private int _titularDNI;
    private string? _titularNobre;
    . . .
}
```

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

13) Reemplazar estas líneas de código por otras equivalentes que utilicen el operador null-coalescing (**??**) y / o la asignación null-coalescing (**??=**)

```
...
if (st1 == null)
{
    if (st2 == null)
    {
        st = st3;
    }
    else
    {
        st = st2;
    }
}
else
{
    st = st1;
}
if (st4 == null)
{
    st4 = "valor por defecto";
}
...
```

14) Crear una solución con tres proyectos: una biblioteca de clases llamada **Automotores**, una biblioteca de clases llamada **Figuras** y una aplicación de consola llamada **Aplicacion**. La biblioteca **Automotores** debe contener una clase pública **Auto** (codificada de la misma manera que la vista en la teoría). La biblioteca **Figuras** debe contener las clases públicas **Circulo** y **Rectangulo**, codificadas de tal forma que el siguiente código (escrito en **Program.cs** del proyecto **Aplicacion**) produzca la siguiente salida

```
using Figuras;
using Automotores;

//El constructor de Circulo espera recibir el radio
var listaCirculos = new List<Circulo>() {
    new Circulo(1.1),
    new Circulo(3),
    new Circulo(3.2)
};

//El constructor de Rectángulo espera recibir la base y la altura
var listaRectangulos = new List<Rectangulo>() {
    new Rectangulo(3, 4),
    new Rectangulo(4.3, 4.4)
};
```

```
//La clase Auto está codificada como la vista en la teoría
var listaAutos = new List<Auto>() {
    new Auto("Nissan", 2017),
    new Auto("Ford", 2015),
    new Auto("Renault")
};

foreach(Circulo c in listaCirculos)
{
    Console.WriteLine($"Área del círculo {c.GetArea()}");
}
foreach(Rectangulo r in listaRectangulos)
{
    Console.WriteLine($"Área del rectángulo {r.GetArea()}");
}
foreach(Auto a in listaAutos)
{
    Console.WriteLine(a.GetDescripcion());
}
```

