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.

```
using System;
using System.Collections.Generic;

public class Persona
{
   public string Nombre;
   public int Edad;
   public string DNI;
}

public class Program
{
   public static void Main()
   {
      List<Persona> personas = new List<Persona>();

      // Pedir datos por consola
      Console.WriteLine("Ingrese los datos de las personas en el siguiente formato: Nombre,Documento,Edad");
      Console.WriteLine("Presione Enter después de cada persona y escriba 'fin' cuando haya terminado.");

      int nro = 1;
      while (true)
      {
         string input = Console.ReadLine();
         if (input == "fin")
         {
            break;
         }

         string[] datos = input.Split(',');
         if (datos.Length != 3)
         {
            Console.WriteLine("Error: debe ingresar los datos en el formato Nombre,Documento,Edad");
            continue;
         }

         Persona persona = new Persona();
         persona.Nombre = datos[0];
         persona.DNI = datos[1];
```

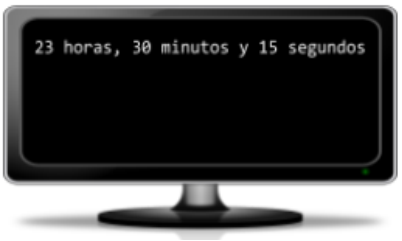
```
            if (!int.TryParse(datos[2], out persona.Edad))
            {
                Console.WriteLine("Error: la edad debe ser un número entero");
                continue;
            }

            personas.Add(persona);
            nro++;
        }

        // Imprimir listado
        Console.WriteLine("Nro) Nombre Edad DNI");
        for (int i = 0; i < personas.Count; i++)
        {
            Console.WriteLine("{0}) {1} {2} {3}", i+1, personas[i].Nombre, personas[i].Edad, personas[i].DNI);
        }
    }
}
```

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();
```




```
public class Hora
{
    private int horas;
    private int minutos;
    private int segundos;

    public Hora(int horas, int minutos, int segundos)
    {
        this.horas = horas;
        this.minutos = minutos;
        this.segundos = segundos;
    }

    public void Imprimir()
    {
        Console.WriteLine("{0:D2}:{1:D2}:{2:D2}", horas, minutos, 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();
```


```
public class Hora
{
   private int horas;
   private int minutos;
   private int segundos;

   public Hora(int horas, int minutos, int segundos)
   {
      this.horas = horas;
      this.minutos = minutos;
      this.segundos = segundos;
   }

   public Hora(double horasDecimales)
   {
      this.horas = (int)horasDecimales;
      this.minutos = (int)((horasDecimales - this.horas) * 60);
      this.segundos = (int)((((horasDecimales - this.horas) * 60) - this.minutos) * 60);
   }

   public void Imprimir()
   {
      if (segundos > 0)
      {
         Console.WriteLine("{0:D2}:{1:D2}:{2:F3}", horas, minutos, (double)segundos);
      }
      else
      {
         Console.WriteLine("{0:D2}:{1:D2}", horas, minutos);
      }
   }
}
```

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.

```
public class Ecuacion2
{
    private double a;
    private double b;
    private double c;

    public Ecuacion2(double a, double b, double c)
    {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public double GetDiscriminante()
    {
        return Math.Pow(b, 2) - 4 * a * c;
    }

    public int GetCantidadDeRaices()
    {
        double discriminante = GetDiscriminante();

        if (discriminante < 0)
        {
            return 0;
        }
        else if (discriminante == 0)
        {
            return 1;
        }
        else
        {
            return 2;
        }
    }

    public void ImprimirRaices()
    {
        double discriminante = GetDiscriminante();

        if (discriminante < 0)
        {
```

```
            Console.WriteLine("La ecuación no tiene soluciones reales");
        }
        else
        {
            double x1 = (-b + Math.Sqrt(discriminante)) / (2 * a);
            double x2 = (-b - Math.Sqrt(discriminante)) / (2 * a);

            Console.WriteLine("Las soluciones reales son:");
            Console.WriteLine("x1 = {0}", x1);

            if (GetCantidadDeRaices() == 2)
            {
                Console.WriteLine("x2 = {0}", x2);
            }
        }
    }
}
```

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());
```

```
1 3 5 7 8 10 14
3
7
14
1
```

Página 2 / 6

```
using System;
using System.Collections;

class Nodo
{
    public int valor;
    public Nodo hijoIzquierdo;
    public Nodo hijoDerecho;

    public Nodo(int valor)
    {
```

```
    this.valor = valor;
    this.hijoIzquierdo = null;
    this.hijoDerecho = null;
}

public void Insertar(int valor)
{
    if (valor < this.valor)
    {
        if (this.hijoIzquierdo == null)
        {
            this.hijoIzquierdo = new Nodo(valor);
        }
        else
        {
            this.hijoIzquierdo.Insertar(valor);
        }
    }
    else if (valor > this.valor)
    {
        if (this.hijoDerecho == null)
        {
            this.hijoDerecho = new Nodo(valor);
        }
        else
        {
            this.hijoDerecho.Insertar(valor);
        }
    }
}

public ArrayList GetInorden()
{
    ArrayList lista = new ArrayList();
    if (this.hijoIzquierdo != null)
    {
        lista.AddRange(this.hijoIzquierdo.GetInorden());
    }
    lista.Add(this.valor);
    if (this.hijoDerecho != null)
    {
        lista.AddRange(this.hijoDerecho.GetInorden());
    }
    return lista;
}

public int GetAltura()
{
    int alturaIzquierdo = (this.hijoIzquierdo != null) ? this.hijoIzquierdo.GetAltura() : 0;
    int alturaDerecho = (this.hijoDerecho != null) ? this.hijoDerecho.GetAltura() : 0;
    return Math.Max(alturaIzquierdo, alturaDerecho) + 1;
}
```

```
    public int GetCantNodos()
    {
        int cantIzquierdo = (this.hijoIzquierdo != null) ? this.hijoIzquierdo.GetCantNodos() : 0;
        int cantDerecho = (this.hijoDerecho != null) ? this.hijoDerecho.GetCantNodos() : 0;
        return cantIzquierdo + cantDerecho + 1;
    }

    public int GetValorMaximo()
    {
        if (this.hijoDerecho == null)
        {
            return this.valor;
        }
        else
        {
            return this.hijoDerecho.GetValorMaximo();
        }
    }

    public int GetValorMinimo()
    {
        if (this.hijoIzquierdo == null)
        {
            return this.valor;
        }
        else
        {
            return this.hijoIzquierdo.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)
```

```
using System;

public class Matriz {
    private int filas;
```

```
private int columnas;
private double[,] matriz;

// Constructor que recibe el número de filas y columnas
public Matriz(int filas, int columnas) {
    this.filas = filas;
    this.columnas = columnas;
    matriz = new double[filas, columnas];
}

// Constructor que recibe una matriz preexistente
public Matriz(double[,] matriz) {
    this.filas = matriz.GetLength(0);
    this.columnas = matriz.GetLength(1);
    this.matriz = matriz;
}

// Método para establecer el valor de un elemento en una posición específica
public void SetElemento(int fila, int columna, double elemento) {
    matriz[fila, columna] = elemento;
}

// Método para obtener el valor de un elemento en una posición específica
public double GetElemento(int fila, int columna) {
    return matriz[fila, columna];
}

// Método para imprimir la matriz en la consola
public void Imprimir() {
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            Console.Write(matriz[i, j] + " ");
        }
        Console.WriteLine();
    }
}

// Método para imprimir la matriz en la consola con un formato específico
public void Imprimir(string formatString) {
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            Console.Write(matriz[i, j].ToString(formatString) + " ");
        }
        Console.WriteLine();
    }
}

// Método para obtener una fila de la matriz como un arreglo de dobles
```

```
public double[] GetFila(int fila) {
    double[] arreglo = new double[columnas];
    for (int j = 0; j < columnas; j++) {
        arreglo[j] = matriz[fila, j];
    }
    return arreglo;
}

// Método para obtener una columna de la matriz como un arreglo de dobles
public double[] GetColumna(int columna) {
    double[] arreglo = new double[filas];
    for (int i = 0; i < filas; i++) {
        arreglo[i] = matriz[i, columna];
    }
    return arreglo;
}

// Método para obtener la diagonal principal de la matriz como un arreglo de dobles
public double[] GetDiagonalPrincipal() {
    double[] arreglo = new double[Math.Min(filas, columnas)];
    for (int i = 0; i < arreglo.Length; i++) {
        arreglo[i] = matriz[i, i];
    }
    return arreglo;
}

// Método para obtener la diagonal secundaria de la matriz como un arreglo de dobles
public double[] GetDiagonalSecundaria() {
    double[] arreglo = new double[Math.Min(filas, columnas)];
    for (int i = 0; i < arreglo.Length; i++) {
        arreglo[i] = matriz[arreglo.Length - i - 1, i];
    }
    return arreglo;
}

// Método para obtener la matriz como un arreglo de arreglos de dobles

  public double[][] getArregloDeArreglo()
{
    double[][] arreglo = new double[filas][];
    for (int i = 0; i < filas; i++)
    {
        arreglo[i] = new double[columnas];
        for (int j = 0; j < columnas; j++)
        {
            arreglo[i][j] = matriz[i, j];
        }
    }
```

```
    return arreglo;
}
```

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;
   }
}
```

Hay un problema con el código presentado, ya que el segundo método y el tercer método tienen la misma firma, es decir, tienen el mismo tipo y cantidad de parámetros. Por lo tanto, esto no es una sobrecarga válida y se producirá un error de compilación. Se debe cambiar el nombre o los parámetros de uno de los métodos para que la sobrecarga sea válida.

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}");
   }
}
```

entero: 5 objeto:5
dynamic d1: 5 dynamic d2:System.Object
dynamic d1: 5 dynamic d2:5
entero: 5 objeto:System.Object
entero: 5 objeto:5

La primera llamada al método Procesar con dos objetos de tipo object como parámetros, ejecuta el primer método sobrecargado, que espera un entero y un objeto. El valor de o se convierte implícitamente a int y la salida es "entero: 5 objeto:5".

La segunda llamada al método Procesar con un objeto de tipo object y un parámetro de tipo dynamic, ejecuta el segundo método sobrecargado, que espera dos parámetros dinámicos. El valor de o se convierte implícitamente a dynamic y la salida es "dynamic d1: 5 dynamic d2:System.Object".
La tercera llamada al método Procesar con dos parámetros de tipo dynamic, ejecuta el segundo método sobrecargado, que espera dos parámetros dinámicos. Ambos parámetros se pasan como dinámicos, por lo que no hay conversión de tipos. La salida es "dynamic d1: 5 dynamic d2:5".
La cuarta llamada al método Procesar con un objeto de tipo object y un parámetro de tipo dynamic, ejecuta el primer método sobrecargado, que espera un entero y un objeto. El valor de o se convierte implícitamente a int y la salida es "entero: 5 objeto:System.Object".
La quinta llamada al método Procesar con dos enteros, ejecuta el primer método sobrecargado, que espera un entero y un objeto. La salida es "entero: 5 objeto:5".

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";
}
...
```

st = st1 ?? st2 ?? st3;
st4 ??= "valor por defecto";
La primera línea asignará el primer valor no nulo encontrado de st1, st2 y st3 a la variable st.

La segunda línea asignará "valor por defecto" a la variable st4 solo si su valor actual es nulo.

```
using Figuras;
using Automotores;
using System.Collections.Generic;

class Program
{
   static void Main(string[] args)
   {
```

```
        var listaCirculos = new List<Circulo>() {
            new Circulo(1.1),
            new Circulo(3),
            new Circulo(3.2)
        };

        var listaRectangulos = new List<Rectangulo>() {
            new Rectangulo(3, 4),
            new Rectangulo(4.3, 4.4)
        };

        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());
        }
    }
}
```

Este código crea tres listas de objetos: listaCirculos, listaRectangulos y listaAutos. Luego, se utiliza un bucle foreach para recorrer cada lista y mostrar la descripción o el área correspondiente.

Con estos pasos, se ha creado una solución en C# con tres proyectos: Automotores, Figuras y Aplicacion. La biblioteca Automotores contiene la clase pública Auto, la biblioteca Figuras contiene las clases públicas Circulo y Rectangulo, y la aplicación Aplicacion utiliza las clases de ambas bibliotecas para mostrar la salida deseada.