Seminario de Lenguajes (.NET)

Práctica 7

1) Codificar las clases e interfaces necesarias para modelar un sistema que trabaja con las siguientes entidades: Autos, Libros, Películas, Personas y Perros. Algunas de estas entidades pueden ser: alquilables (se pueden alquilar a una persona y ser devueltas por una persona), vendibles (se pueden vender a una persona), lavables (se pueden lavar y secar) reciclables (se pueden reciclar) y atendibles (se pueden atender). A continuación se describen estas relaciones:

- Son Alquilables: Libros y Películas
- Son Vendibles: Autos y Perros
- Son Lavables: Autos
- Son Reciclables: Libros y Autos
- Son Atendibles: Personas y Perros

Completar el código de la clase estática Procesador:

```
static class Procesador
{
    public static void Alquilar(IAlquilable x, Persona p) => x.SeAlquilaA(p);
    public static . . .
    . . .
}
```

La ejecución del siguiente código debe mostrar en la consola la salida indicada:

```
Auto auto = new Auto();
Libro libro = new Libro();
Persona persona = new Persona();
Perro perro = new Perro();
Pelicula pelicula = new Pelicula();
Procesador.Alquilar(pelicula, persona);
Procesador.Alquilar(libro, persona);
Procesador.Atender(persona);
Procesador.Atender(perro);
Procesador.Devolever(pelicula, persona);
Procesador.Devolever(libro, persona);
Procesador.Lavar(auto);
Procesador.Reciclar(libro);
Procesador.Reciclar(auto);
Procesador.Secar(auto);
Procesador.Vender(auto, persona);
Procesador.Vender(perro, persona);
```

Salida por consola

```
Alquilando película a persona
Alquilando libro a persona
Atendiendo persona
Atendiendo perro
Película devuelta por persona
Libro devuelto por persona
Lavando auto
Reciclando libro
Reciclando auto
Secando auto
Vendiendo auto a persona
Vendiendo perro a persona
```

2) Incorporar al ejercicio anterior la posibilidad también de lavar a los perros. También se debe incorporar una clase derivada de Película, las "películas clásicas" que además de alquilarse pueden venderse. Estos cambios deben poder realizarse sin necesidad de modificar la clase estática Procesador. El siguiente código debe producir la salida indicada:

```
Auto auto = new Auto();
Libro libro = new Libro();
Persona persona = new Persona();
Perro perro = new Perro();
Pelicula pelicula = new Pelicula();
Procesador.Alquilar(pelicula, persona);
Procesador.Alquilar(libro, persona);
Procesador.Atender(persona);
Procesador.Atender(perro);
Procesador.Devolever(pelicula, persona);
Procesador.Devolever(libro, persona);
Procesador.Lavar(auto);
Procesador.Reciclar(libro);
Procesador.Reciclar(auto);
Procesador.Secar(auto);
Procesador.Vender(auto, persona);
Procesador.Vender(perro, persona);
Procesador.Lavar(perro);
Procesador.Secar(perro);
PeliculaClasica peliculaClasica = new PeliculaClasica();
Procesador.Alquilar(peliculaClasica, persona);
Procesador.Devolever(peliculaClasica, persona);
Procesador.Vender(peliculaClasica, persona);
```

Salida por consola

```
Alquilando película a persona
Alquilando libro a persona
Atendiendo persona
Atendiendo perro
Película devuelta por persona
Libro devuelto por persona
Lavando auto
Reciclando libro
Reciclando auto
Secando auto
Vendiendo auto a persona
Vendiendo perro a persona
Lavando perro
Secando perro
Alquilando película clásica a persona
Película clásica devuelta por persona
Vendiendo película clásica a persona
```

3) Incorporar al ejercicio anterior las interfaces y métodos necesarios para que el siguiente código produzca la salida indicada:

```
var lista = new List<object>() {
          new Persona(),
          new Auto()
     };
foreach (IComercial c in lista)
{
   c.Importa();
}
foreach (IImportante i in lista)
{
   i.Importa();
}
(lista[0] as Persona)?.Importa();
(lista[1] as Auto)?.Importa();
```

Salida por consola

```
Persona vendiendo al exterior
Auto que se vende al exterior
Persona importante
Auto importante
Método Importar() de la clase Persona
Método Importar() de la clase Auto
```

4) Incorporar al ejercicio anterior las interfaces, propiedades y métodos necesarios para que el siguiente código produzca la salida indicada:

```
var vector = new INombrable[] {
     new Persona() {Nombre="Zulema"},
     new Perro() {Nombre="Sultán"},
     new Persona() {Nombre="Claudia"},
     new Persona() {Nombre="Carlos"},
     new Perro() {Nombre="Chopper"},
};
Array.Sort(vector);  //debe ordenar por Nombre alfabéticamente
foreach (INombrable n in vector)
{
    Console.WriteLine($"{n.Nombre}: {n}");
}
```

Salida por consola

```
Carlos: Carlos es una persona
Chopper: Chopper es un perro
Claudia: Claudia es una persona
Sultán: Sultán es un perro
Zulema: Zulema es una persona
```

5) Modificar el ejercicio anterior para que el siguiente código produzca la salida indicada:

```
var vector = new INombrable[] {
     new Persona() {Nombre="Zulema"},
     new Perro() {Nombre="Sultán"},
     new Persona() {Nombre="Claudia"},
     new Persona() {Nombre="Carlos"},
     new Perro() {Nombre="Chopper"},
};
Array.Sort(vector);  //debe ordenar por Nombre alfabéticamente
foreach (INombrable n in vector)
{
    Console.WriteLine($"{n.Nombre}: {n}");
}
```

Salida por consola

```
Carlos: Carlos es una persona
Claudia: Claudia es una persona
Zulema: Zulema es una persona
Chopper: Chopper es un perro
Sultán: Sultán es un perro
```

Es decir, el ordenamiento ahora da prioridad a las personas sobre los perros, primero se listan las personas, ordenadas alfabéticamente, y luego los perros, también ordenados alfabéticamente.
Tip: Sólo es necesario cambiar la forma en que un perro o una persona se sabe comparar contra otro objeto.

6) Modificar el ejercicio anterior para que el siguiente código produzca la salida indicada:

```
var vector = new INombrable[]  {
       new Persona() {Nombre="Ana María"},
       new Perro() {Nombre="Sultán"},
       new Persona() {Nombre="Ana"},
       new Persona() {Nombre="José Carlos"},
       new Perro() {Nombre="Chopper"}
};
Array.Sort(vector, new ComparadorLongitudNombre());//ordena por longitud de Nombre
foreach (INnombrable n in vector)
{
     Console.WriteLine($"{n.Nombre.Length}: {n.Nombre}");
}
```

Salida por consola

```
3: Ana
6: Sultán
7: Chopper
9: Ana María
11: José Carlos
```

7) Proponer una forma para conseguir un ordenamiento aleatorio de todos los elementos de un vector de objetos (**`object[]`**) utilizando **`Array.Sort().`** Cada vez que se invoque debe producir un ordenamiento aleatorio diferente. (Investigar la clase **`System.Random`**)

8) Codificar usando iteradores los métodos:
`Rango(i, j, p)` que devuelve la secuencia de enteros desde **`i`** hasta **`j`** con un paso de **`p`**
`Potencia(b,k)` que devuelve la secuencia $b^1, b^2, ... b^k$
`DivisiblePor(e,i)` retorna los elementos de **`e`** que son divisibles por **`i`**
Observar la salida que debe producir el siguiente código:

```
using System.Collections;

IEnumerable rango = Rango(6, 30, 3);
IEnumerable potencias = Potencias(2, 10);
IEnumerable divisibles = DivisiblesPor(rango, 6);
foreach (int i in rango)
{
    Console.Write(i + " ");
}
Console.WriteLine();
foreach (int i in potencias)
{
    Console.Write(i + " ");
}
Console.WriteLine();
foreach (int i in divisibles)
{
    Console.Write(i + " ");
}
Console.WriteLine();
```

Salida por consola

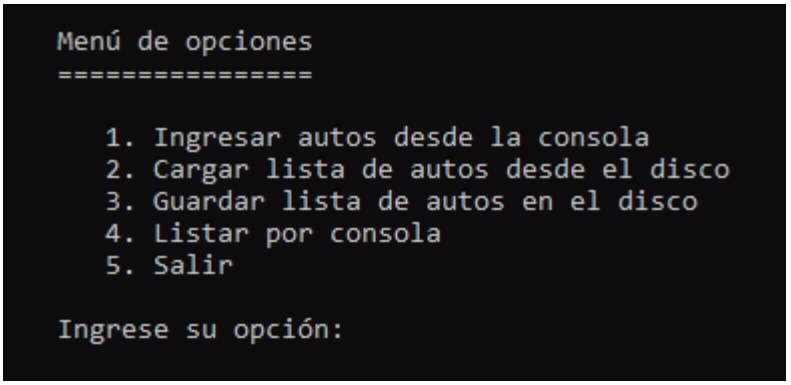
```
6 9 12 15 18 21 24 27 30
2 4 8 16 32 64 128 256 512 1024
6 12 18 24 30
```

9) Codificar un programa que permita al usuario escribir un texto por consola. El mismo puede constar de varios párrafos. Se considera el fin de la entrada cuando el usuario ingresa una línea vacía, en ese momento el programa solicitará al usuario el nombre del archivo para guardar el texto escrito. Si el usuario escribe un nombre de archivo válido, se guarda el texto ingresado en ese archivo, de lo contrario no se hace nada y termina el programa.

a) Utilizando la instrucción using

b) Sin utilizar la instrucción using

10) Codificar una aplicación para manejar una lista de autos (dos atributos: Marca y Modelo) con el siguiente menú de opciones por consola:



La opción 1 permite al usuario agregar autos a la lista actual en memoria ingresando la marca y el modelo por la consola. El final de la entrada se detecta al ingresar una marca vacía (sin caracteres).
La opción 2, carga en memoria una lista de autos previamente guardada en algún archivo de texto.
La opción 3 guarda en un archivo de texto la lista actual en memoria. Se puede implementar de infinidad de maneras, por ejemplo se podría guardar cada auto en dos líneas consecutivas, la primera para la marca y la segunda para el modelo.
La opción 4 produce un listado por consola de todos los autos en la lista actual en memoria.

Tip: La interacción con el menú puede resolverse de la siguiente manera:

```
   . . .
   ConsoleKeyInfo tecla;
   do
   {
      tecla = Console.ReadKey(true);
      switch (tecla.KeyChar)
      {
         case '1': . . .
         case '2': . . .
         case '3': . . .
         case '4': . . .
      }
   } while (tecla.KeyChar != '5');
}
```