Seminario de Lenguajes (.NET)

Práctica 9

1) Codificar el método genérico **`Get`** para que el siguiente código produzca la salida en la consola indicada.

```
var lista = new List<object> { "hola", 7, 'A' };
string st = Get<string>(lista, 0);
int i = Get<int>(lista, 1);
char c = Get<char>(lista, 2);
Console.WriteLine($"{st} {i} {c}");
```

Salida por consola

```
hola 7 A
```

2) Codificar los métodos que faltan para que el siguiente código produzca la salida en la consola indicada.

```
int[] vector1 = new int[] { 1, 2, 3 };
bool[] vector2 = new bool[] { true, true, true };
string[] vector3= new string[] { "uno", "dos", "tres" };
Set<int>(vector1, 110, 2);
Set<bool>(vector2, false, 1);
Set<string>(vector3, "Hola Mundo!", 0);
Imprimir(vector1);
Imprimir(vector2);
Imprimir(vector3);
```

Salida por consola

```
1 2 110
True False True
Hola Mundo! dos tres
```

Debe evitarse que durante la ejecución del método **`Imprimir`** se produzca *boxing* o *unboxing*. Tip El método **`Imprimir`** también es un método genérico, no se advierte fácilmente porque no se ha explicitado el parámetro de tipo (el compilador lo infiere)

3) Codificar los métodos genéricos **`CrearArreglo`** y **`GetNuevoObjetoDelMismoTipo`** que faltan para que el siguiente código produzca la salida en la consola indicada. El método **`GetNuevoObjetoDelMismoTipo`** debe crear y devolver un nuevo elemento del mismo tipo del que recibe como parámetro. Tip: Para codificar el método **`CrearArreglo`** tener presente el uso de **`params`**

```
string[] vector1 = CrearArreglo<string>("uno", "dos");
foreach (string st in vector1) Console.Write(st + " - ");
Console.WriteLine();
double[] vector2 = CrearArreglo<double>(1, 2.3, 4.1, 6.7);
foreach (double valor in vector2) Console.Write(valor + " - ");
Console.WriteLine();

var stb = new System.Text.StringBuilder();
var a = GetNuevoObjetoDelMismoTipo(stb);
var b = GetNuevoObjetoDelMismoTipo(17);
Console.WriteLine(a.GetType());
Console.WriteLine(b.GetType());
```

Salida por consola

```
uno - dos -
1 - 2,3 - 4,1 - 6,7 -
System.Text.StringBuilder
System.Int32
```

Nota: el método **`GetNuevoObjetoDelMismoTipo`** sólo funciona con parámetros cuyo tipo debe ser no abstracto y con un constructor público sin parámetros.

4) Dada la siguiente clase genérica

```
class Nodo<T>
{
   public T Valor { get; private set; }
   public Nodo<T>? Proximo { get; set; } = null;
   public Nodo(T valor) => Valor = valor;
}
```

Utilizar la clase **`Nodo<T>`** para codificar codificar una lista enlazada genérica tal manera que el código siguiente produzca la salida indicada:

```
var lista = new ListaEnlazada<int>();
lista.AgregarAdelante(3);
lista.AgregarAdelante(100);
lista.AgregarAtras(10);
lista.AgregarAtras(11);
lista.AgregarAdelante(0);
IEnumerator<int> enumerador = lista.GetEnumerator();
while (enumerador.MoveNext())
{
   int i = enumerador.Current;
   Console.Write(i + " ");
}
Console.WriteLine();
```

Salida por consola

```
0 100 3 10 11
```

5) Tomar como base el ejercicio 7 de la práctica 4 para transformar la clase **`Nodo`** de dicho ejercicio en una clase genérica **`Nodo<T>`** de un árbol binario de búsqueda de valores de tipo **`T`**. Claramente **`T`** tiene que ser de un tipo que pueda ser ordenado (elementos comparables).
Desarrollar los siguientes métodos y propiedades

1. **`Insertar(valor)`**: Inserta valor (de tipo **`T`**) en el árbol descartándolo en caso que ya exista.
2. **`Inorden`**: devuelve un **`List<T>`** 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.

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

```
Nodo<int> n = new Nodo<int>(7);
n.Insertar(3);
n.Insertar(1);
n.Insertar(5);
n.Insertar(12);
foreach (int elem in n.InOrder)
{
    Console.Write(elem + " ");
}
Console.WriteLine();
Console.WriteLine($"Altura: {n.Altura}");
Console.WriteLine($"Cantidad: {n.CantNodos}");
Console.WriteLine($"Mínimo: {n.ValorMinimo}");
Console.WriteLine($"Máximo: {n.ValorMaximo}");

Nodo<string> n2 = new Nodo<string>("hola");
n2.Insertar("Mundo");
n2.Insertar("XYZ");
n2.Insertar("ABC");
foreach (string elem in n2.InOrder)
{
    Console.Write(elem + " ");
}
Console.WriteLine();
Console.WriteLine($"Altura: {n2.Altura}");
Console.WriteLine($"Cantidad: {n2.CantNodos}");
Console.WriteLine($"Mínimo: {n2.ValorMinimo}");
Console.WriteLine($"Máximo: {n2.ValorMaximo}");
```

Salida por consola

```
1 3 5 7 12
Altura: 2
Cantidad: 5
Mínimo: 1
Máximo: 12
ABC hola Mundo XYZ
Altura: 2
Cantidad: 4
Mínimo: ABC
Máximo: XYZ
```

6) Codificar los métodos genéricos **`Agregar`** e **`Imprimir`** de la clase **`Program`** que faltan de tal forma que el código siguiente produzca la salida en la consola que se indica. Nota: La clase **`Nodo<T>`** es la desarrollada en el ejercicio anterior

```
class Program
{
   static void Main(string[] args)
   {
       Nodo<int> nodo1 = new Nodo<int>(5);
       Agregar(nodo1, 1, 10, 3, 4, 56, 22, 31, 0, 15, 14);
       Imprimir(nodo1);
       Nodo<string> nodo2 = new Nodo<string>("hola");
       Agregar(nodo2, "Mundo", "XYZ", "ABC", "nada");
       Imprimir(nodo2);
   }
   . . .
```

Salida por consola

```
0 1 3 4 5 10 14 15 22 31 56
Altura: 5
Cantidad: 11
Mínimo: 0
Máximo: 56
ABC hola Mundo nada XYZ
Altura: 3
Cantidad: 5
Mínimo: ABC
Máximo: XYZ
```

7) Codificar una clase **`Persona`** con al menos las dos propiedades: **`Nombre`** de tipo **`string`** y **`Edad`** de tipo **`int`**. Instanciar una lista de tipo **`List<Persona>`** y obtener utilizando el método **`ConvertAll`** una segunda lista de tipo **`List<Par<string,int>>`** siendo **`Par`** la clase genérica presentada en la teoría.

Por cada objeto **persona** de la primera lista se obtiene un objeto **par** en la segunda lista guardando en las propiedades **A** y **B** de cada **par** las propiedades **Nombre** y **Edad** de cada **persona** respectivamente.

8) Solicitar al usuario que ingrese el nombre de un archivo de texto por la consola. Si el archivo existe mostrar por consola un listado con cada una de las palabras encontradas en ese archivo seguida de dos puntos (:) y de la cantidad de ocurrencias de la misma en el archivo. Además deben presentarse ordenadas alfabéticamente. Es decir si el archivo de texto tuviese este párrafo, la salida sería:

Salida por consola

```
8: 1
Además: 1
al: 1
alfabéticamente: 1
archivo: 5
cada: 1
cantidad: 1
con: 1
consola: 2
de: 8
deben: 1
decir: 1
dos: 1
el: 4
en: 2
encontradas: 1
. . .
```

Tip: Utilizar un **SortedDictionary<TKey,TValue>** para ir acumulando los contadores de cada palabra. El método **ContainsKey(clave)** de un **SortedDictionary** devuelve un valor **bool** que indica si existe la clave en el diccionario. También puede resultar muy útil el método **Split** de la clase **string**.

9) Solicitar al usuario que ingrese por consola el nombre de dos archivos de texto. Si existen ambos se debe mostrar por consola un listado con cada una de las palabras que se encuentran en ambos archivos a la vez. Por ejemplo, si el primer archivo contiene el texto del primer párrafo del punto 8, y el segundo archivo contiene este párrafo, la salida debería ser:

Salida por consola

```
8 - al - archivo - cada - con - consola - de - dos - el -
en - este - ingrese - la - las - listado - mostrar -
nombre - palabras - párrafo - por - que - salida - si -
Si - Solicitar - texto - un - una - usuario - y -
```

Nota: Observar además que las palabras están listadas en orden alfabético.
Tip: Como cabría esperar, los tipos genéricos que representan conjuntos, implementan operaciones de conjunto como la unión, la intersección, ...

10) Modificar el ejercicio anterior para que, una vez hallada la lista de palabras comunes en ambos archivos, se listen por consola todas las posiciones en las que aparece cada una de esas palabras en cada uno de los dos archivo. Por ejemplo:

Salida por consola

```
Palabra: "8"
   |--Posiciones en Texto1:--> 0
   |--Posiciones en Texto2:--> 295

Palabra: "al"
   |--Posiciones en Texto1:--> 13 158 313 388
   |--Posiciones en Texto2:--> 13 160 346

Palabra: "archivo"
   |--Posiciones en Texto1:--> 52 91 185 269 345
   |--Posiciones en Texto2:--> 65 195 240 311

Palabra: "cada"
   |--Posiciones en Texto1:--> 141
   |--Posiciones en Texto2:--> 143

Palabra: "con"
   |--Posiciones en Texto1:--> 76 118 137 168
   |--Posiciones en Texto2:--> 50 120 139 248 319

Palabra: "consola"
   |--Posiciones en Texto1:--> 76 118
   |--Posiciones en Texto2:--> 50 120
...
```

Aunque se puede resolver de muchas maneras, se pretende que los alumnos hagan el mayor uso posible de las colecciones genéricas. Se debe convertir (usando el método **`ConvertAll`**) un **`List<string>`** que contiene las palabras comunes a ambos archivos en un **`List<PalabraPosiciones>`** que es finalmente la lista que se imprimirá en la consola. La clase **`PalabraPosiciones`** debe ser capaz de guardar una palabra más la lista de posiciones de dicha palabra en un determinado texto. En realidad vamos a generalizarlo para trabajar con *n* textos (en este ejercicio alcanzaría con *n*=2) por eso, para guardar las posiciones vamos a usar una lista de listas de enteros.

```
class PalabraPosiciones
{
    public string Palabra { private set; get; }
    public List<List<int>> Posiciones { private set; get; } = new List<List<int>>();
    . . .
}
```

En **`posiciones[0]`** se guardará una lista de enteros con las posiciones de las ocurrencias de **`Palabra`** en el primer texto. En **`Posiciones[1]`** se guardará una lista de enteros con las posiciones de las ocurrencias de **`Palabra`** en el segundo texto.
Tip: Para hallar la posición de un string dentro de otro se puede utilizar una sobrecarga del método **`IndexOf`** de la clase **`string`**.