TRABAJO PRACTICO

N°2

CENTRAL TELEFONICA

MACHACA GASTÓN

FUNCIONALIDAD PRETENDIDA:

Realizar llamadas al ámbito local o provincial siendo agendados en una lista que incluye todas las llamadas realizadas y el costo de cada una.

Local: Buenos Aires.

Provincial: Córdoba, Mendoza y Jujuy.

Clase 1: Introducción a .NET y C#

Se hace uso de la categoría enteros únicamente de la clase Int32 que es un entero con signo de 32 bit llamado en el alias de c# "int".

En la categoría Punto flotante se hace uso de la clase Single que vendría a ser un numero de punto flotante de simple precisión (32 bit) que en el alias de C# se denomina como "float".

Así como también el uso de valores booleanos (true o false), caracteres Unicode de tipo "char", la raíz de la jerarquía de objetos "object" y cadenas de caracteres Unicode de tamaño fijo "string".

Conversiones explicitas que se mencionaran en las siguientes clases, el uso de operadores lógicos, aritméticos.

Clase 2: Clases y métodos estáticos

Tenemos una biblioteca de clases con el nombre de Telefonía Celular que contiene 4 clases identificadas de la siguiente manera.

Estas son:

- Central (Modificador: Publico).
- Llamada (Modificador: Publico y Abstracto).
- Local (Modificador: Publico).
- Provincial (Modificador: Publico).

```
public class Local
public class Central
public abstract class Llamada
public class Provincial
```

Así como también cada clase tienen sus atributos que mencionaremos a continuación:

Sintaxis de los atributos:

[modificador] tipo identificador;

Clase llamada:

```
protected float duracion;
protected string numero;
protected string codigoArea;
```

Clase Local:

```
protected float costo;
```

Clase Central:

```
private List<Llamada> listaDeLlamadas;

protected string empresa;
```

Clase Provincial:

```
protected Franja franjaSeccion;
```

La sintaxis de los métodos es la siguiente, aunque también incluyo casos de sobrecargas de métodos que se detallaran en la Clase N°4.

```
[modificador] retorno Identificador ( [args] )
{
    // Sentencias
}
```

Ahora se pasará a mencionar los métodos de cada clase:

Clase Central:

```
/// <summary> Metodo que se encarga de calcular el costo de las llamadas.
3 referencias
private float CalcularCosto(TipoLlamada tipo)...

/// <summary> Metodo que se encarga de juntar toda la informacion de la central ...
1 referencia
private string Mostrar()...

/// <summary> Ordena la lista de llamadas por la duracion.
1 referencia
public void OrdenarLlamadas()...
/// <summary> Sobrecarga del ToString();
6 referencias
public override string ToString()...

/// <summary> Metodo que asocia las llamadas a la lista.
1 referencia
private void AgregarLlamada(Llamada nuevaLlamada)...
```

Clase Provincial:

```
/// <summary> Muestra los datos generales (numero,codigo de area,duracion)de la ...
6 referencias
protected override string Mostrar()...

/// <summary> Calcula el costo en base a la provincia seleccionada a contactar.
1 referencia
private float CalcularCosto()...

/// <summary> Publica todos los datos de la llamada provincial utilizando la fun ...
6 referencias
public override string ToString()...

/// <summary> Se encarga de preguntar si se trata de una llamada provincial.
1 referencia
public override bool Equals(object obj)...

/// <summary> Implementado para evitar la advertencia de invalidar el GetHashcod ...
1 referencia
public override int GetHashCode()...
```

Clase Local:

```
/// <summary> Metodo que se encarga de mostrar el numero,duracion,codigo de area ...
6 referencias
protected override string Mostrar()...

/// <summary> Metodo que calcula el costo de la llamada local.
1 referencia
private float CalcularCosto()...

/// <summary> Sobrecarga de Equals.
1 referencia
public override bool Equals(object obj)...

/// <summary> Sobrecarga para evitar al advertencia.
1 referencia
public override int GetHashCode()...

/// <summary> Sobrecarga del ToString()
6 referencias
public override string ToString()...
```

Clase Llamada:

```
/// <summary> Metodo que se encarga de mostrar la informacion de la llamada.
6 referencias
protected virtual string Mostrar()...

/// <summary> Ordena las llamadas por su duracion.
1 referencia
public static int OrdenarPorDuracion(Llamada llamada1, Llamada llamada2)...
```

El namespace de la biblioteca de clases "TelefoniaCelular" para la organización del código y reducir los conflictos entre nombres.

```
namespace TelefoniaCelular
```

Y las directivas using usadas de las 4 clases (Local, Provincial, Llamada y Central).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Clase 3: Programación orientada a objetos

Para esta clase doy uso de los constructores de cada clase que menciono a continuación con su respectivo comentario:

Clase Provincial:

```
/// <summary>
/// Constructor designado para establecer una llamada provincial.
/// </summary>
/// <param name="origen">Codigo de area de la provincia a contactar.</param>
/// <param name="miFranja">Provincia a la que se realiza la llamada</param>
/// <param name="duracion">Tiempo de llamada realizada</param>
/// <param name="destino">Numero de celular de la persona a contactar</param>
4 referencias
public Provincial(string codigoArea, Franja miFranja, float duracion, string numero) : base(duracion, numero, codigoArea)
{
    this.franjaSeccion = miFranja;
}

/// <summary>
/// Constructor para repetir una llamada realizada a otra provincia.
/// </summary>
/// <param name="miFranja">Provincia a la que se realiza la llamada</param>
/// <param name="llamada">Datos de llamada realizada</param>
1 referencia
public Provincial(Franja miFranja, Llamada llamada) : this(llamada.CodigoArea, miFranja, llamada.Duracion, llamada.Numero)
{
}
```

Clase Llamada:

```
/// <summary>
/// Constructor que tomara el numero,la duracion y codigo de area de la llamada.
/// </summary>
/// <param name="duracion">Duracion de la llamada</param>
/// <param name="numero">Numero de la llamada</param>
/// <param name="codigoArea">Codigo de area de la llamada</param>
2 referencias
public Llamada(float duracion, string numero, string codigoArea)
{
    this.duracion = duracion;
    this.numero = numero;
    this.codigoArea = codigoArea;
}
```

Clase Central:

```
/// <summary>
/// Constructor sin parametros encargado de iniciar la lista.
/// </summary>
1 referencia
public Central()
{
    listaDeLlamadas = new List<Llamada>();
}

/// <summary>
/// Constructor que asignara el nombre de la empresa.
/// </summary>
/// <param name="nombreEmpresa">Nombre de la empresa</param>
2 referencias
public Central(string nombreEmpresa) : this()
{
    this.empresa = nombreEmpresa;
}
```

Clase Local:

```
/// <summary>
/// Constructor que recibe el codigo de area,la duracion,numero y costo de la llamada.
/// </summary>
/// <param name="codigoArea">Codigo de area</param>
/// <param name="duracion">Duracion de la llamada</param>
/// <param name="numero">Numero de la llamada</param>
/// <param name="costo">Costo de la llamada</param>
3 referencias
public Local(string codigoArea, float duracion, string numero, float costo) : base(duracion, numero, codigoArea)
{
    this.costo = costo;
}

/// <summary>
/// Constructor que toma una llamada y su costo.
/// </summary>
/// <param name="llamada">Datos de la llamada</param>
/// <param name="costo">Costo de la llamada</param>
0 referencias
public Local(Llamada llamada, float costo) : this(llamada.CodigoArea, llamada.Duracion, llamada.Numero, costo)
{
}
```

Y después puedo declarar e instanciar un objeto, en este caso, podría ser local, provincial o central. El caso de llamada no se puede invocar al ser abstracto.

Esto lo pruebo en la aplicación de consola.

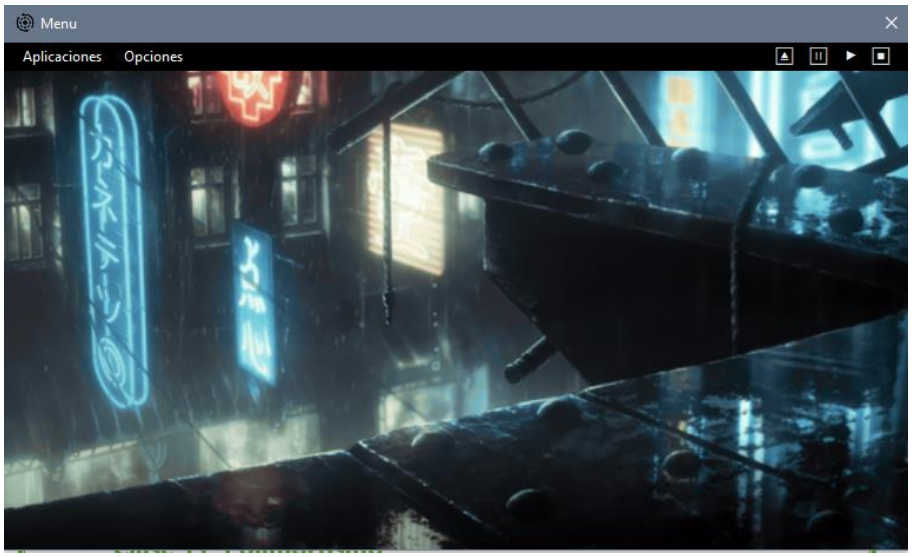
```
Central c = new Central("Movistar");
Local l1 = new Local("011", 30, "56216544", 2.65f);
Provincial l2 = new Provincial("351", Provincial.Franja.Cordoba, 21, "52166452");
Provincial l3 = new Provincial("261", Provincial.Franja.Mendoza, 44, "43921062");
Provincial l4 = new Provincial(Provincial.Franja.Jujuy, 12);
```

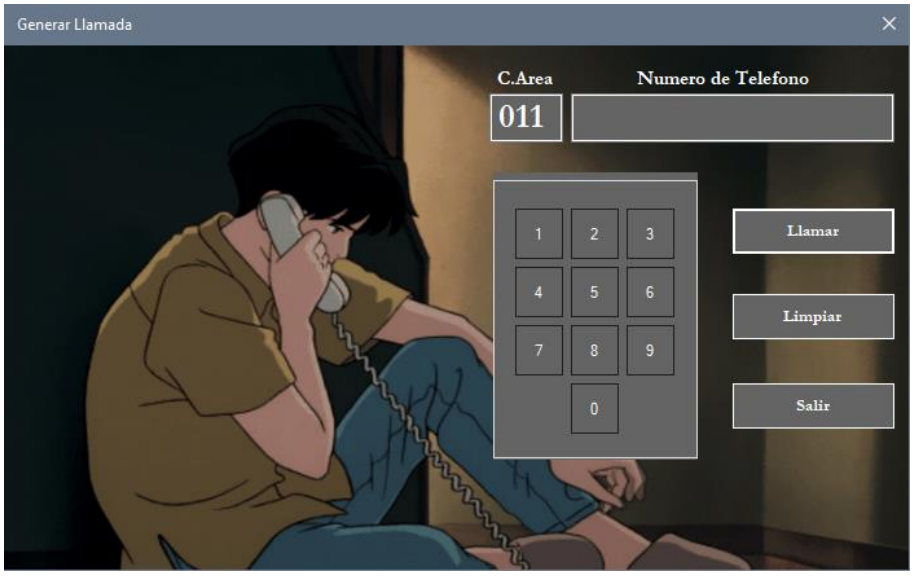
Clase 4: Sobrecarga

En el tema sobrecargas abarco todas las clases, lo utilizo para la sobrecarga del toString()en todas las clases que pertenecen a TelefoniaCelular, como así también los override de mostrar de las clases local, provincial y central y con la clase abstracta llamada me encargo de también pasarles a provincial y local el número, el código de área y la duración de la llamada.

Clase 6: Windows Forms, Arrays, Strings

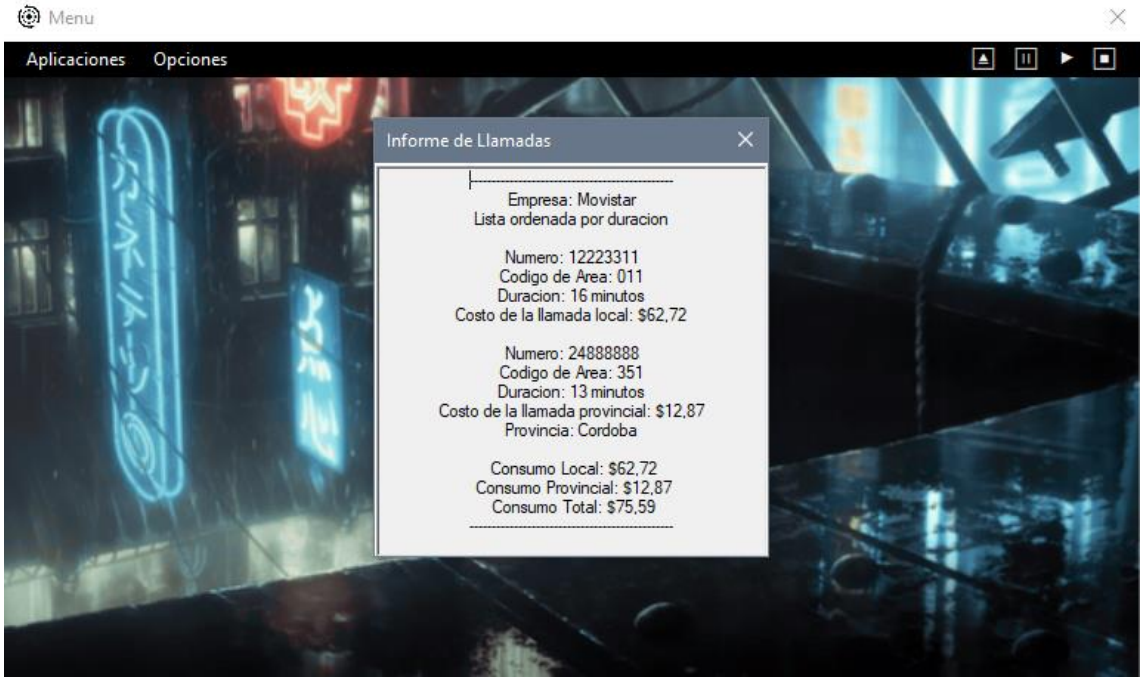
Con windows forms me encargo de hacer un formulario de ingreso de llamadas, asi como también un menú previo al acceso de las llamadas y te permite seleccionar si realizar una llamada provincial o local.





Clase 7: Colecciones

Para las colecciones me encargo de hacer una lista de llamadas que va anotando cada llamada que vas realizando, ya sea provincial o local.



Clase 8: Encapsulamiento

Para el encapsulamiento utilizo los enumerados y propiedades tales para asignar una serie de opciones de provincias a las cuales podes llamar (Mendoza, Jujuy, Córdoba) y también propiedades de solo lectura para calcular el costo de las llamadas.

```
/// <summary>
/// Opciones de provincias disponibles para su seleccion.
/// </summary>
11 referencias
public enum Franja
{
    Cordoba,
    Mendoza,
    Jujuy
}
```

```
8 referencias
public override float CostoLlamada
{
    get
    {
        return this.CalcularCosto();
    }
}

/// <summary>
```

Clase 9: Herencia

Para herencia estoy usando como base a llamada que asigna el número, el código de área y la duración de la llamada, mientras que las clases heredadas vienen a ser provincial y local que dependiendo de cuál sea vas a tener valores propios de costo a la hora de realizar las llamadas.

Clase 10 y Clase 11: Polimorfismo, Clases y métodos abstractos

Para estas dos clases solo hago uso de una clase abstracta, la clase llamada, así como también esta incluye un miembro virtual que le permite pasar la información de número, código de área y duración a cada una de las clases que la heredan.

```
/// <summary> Metodo que se encarga de mostrar la informacion de la llamada.
6 referencias
protected virtual string Mostrar()
{
    StringBuilder cadena = new StringBuilder();

    cadena.Append("\nNumero: " + this.numero + "\n");
    cadena.AppendLine("Codigo de Area: " + this.codigoArea);
    cadena.AppendFormat("Duracion: " + this.duracion + " minutos");

    return cadena.ToString();
}
```

Así como también las clases provincial y local tienen un override de costo de llamada que les permite ser independientes y tener sus propios rangos de precio.

Propiedad abstracta en la clase llamada.

```
/// <summary>
/// Propiedad de solo lectura que retornara el costo de la llamada.
/// </summary>
8 referencias
public abstract float CostoLlamada
{
    get;
}
```

Y esto lo utiliza la central para tener el rango de precio de cada llamada.