TP2 - Laboratorio 03

1. Agregar capa de datos para pruebas

Objetivos

Agregar un proyecto existente para la capa de datos. La misma sólo mantiene los datos en memoria y no hace persistencia porque se utilizará para realizar pruebas con las demás capas. Esta capa luego será reemplazada por una que si realice la persistencia

Duración Aproximada

5 minutos

Pasos

1. Ir a la carpeta C:\Net\Labs y crear la carpeta TP2L03.

2. Copiar la carpeta TP2 que se encuentra en C:\Net\Labs\TP2L02\ a C:\Net\Labs\TP2L03\
   De esta forma modificaremos esta copia manteniendo la versión anterior intacta.

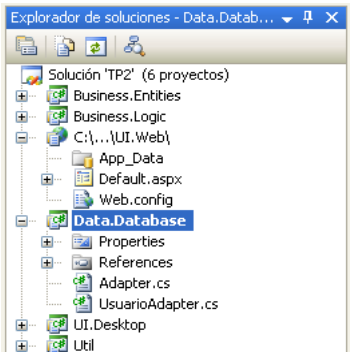
3. Dentro de la carpeta TP2 que acabamos de copiar extraemos el contenido del archivo Data.Database.zip que se encuentra junto con este enunciado. Este es un proyecto de tipo Librería de Clases que simulará una capa de datos. Podemos eliminar este archivo luego de extraerlo.

4. Hacemos doble clic sobre el archivo TP2.sln para abrir la solución

5. En el explorador de soluciones hacemos clic con el botón derecho sobre el la solución y vamos a Agregar... ? Proyecto Existente...

6. Aparece entonces una ventana para seleccionar el proyecto, en este caso vamos a seleccionar el archivo Data.Database.csproj que es el archivo de proyecto que contiene toda la información del proyecto. Para ello en la ventana nos dirigimos a la carpeta C:\Net\Labs\TP2L03\TP2\Data.Database\ y seleccionamos el archivo Data.Database.csproj

7. Veremos que aparece el nuevo proyecto en el explorador de soluciones llamado Data.Database con 2 clases: Adapter y UsuarioAdapter



2. Agregar referencias entre los proyectos

Objetivos

Agregar las referencias entre proyectos para que tengan visibilidad unos sobre otros como indica el gráfico de capas del TP2Lab01.

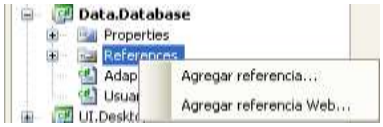
Duración Aproximada

5 minutos

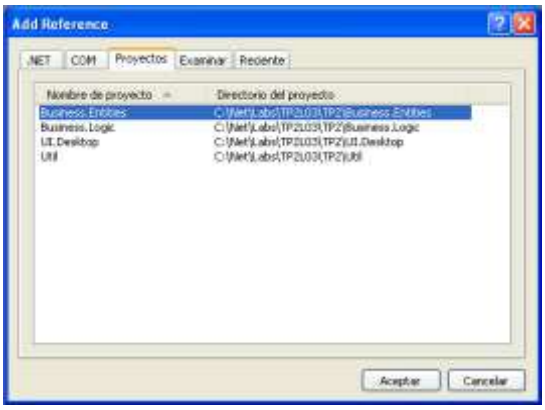
Pasos

1. Todos los proyectos deben tener visibilidad sobre la capa de entidades (Business.Entities) que es la que permite transferir información entre capas y mantener la comunicación entre ellas minimizando el acoplamiento y sobre la capa inmediata inferior.

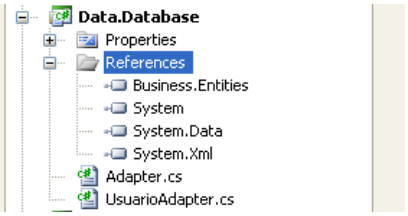
2. Empezaremos por la capa de datos que no posee capa inferior. En el explorador de soluciones, en proyecto Data.Database vamos a References y hacemos clic con el botón derecho y luego elegimos Agregar Referencia...



3. Se abrirá otra ventana y en ella seleccionamos la pestaña Proyectos, elegimos el proyecto Business.Entities y hacemos clic en Aceptar



4. Ahora veremos como en el proyecto Data.Database aparece la referencia al proyecyo Business.Entities



5. Luego repetiremos la operación agregando en la capa de la lógica de negocio (Business.Logic) referencia al proyecto Business.Entities para manipular las clases de dominio y a la capa inferior Data.Database.

6. Y agregaremos en los proyectos de UI.Desktop y UI.Web referencias a Business.Entities y a Business.Logic

3. Crear los métodos de negocio para un ABMC simple en capas

Objetivos
Desarrollar los métodos de la lógica de negocio un ABMC de usuarios en capas

Duración Aproximada
20 minutos

1. En en proyecto Business.Logic crear la clase BusinessLogic de la que todas las demás heredarán y hacerla pública

2. Crear la clase UsuarioLogic, hacerla pública y que herede de BusinessLogic.

3. Agegar los using necesarios para las clases de las capas entidades y datos.

4. Crear una propiedad UsuarioData del tipo Data.Database.UsuarioAdapter e instanciarlo en el constructor de UsuarioLogic.

5. Crear los métodos GetOne, GetAll, Save y Delete en UsuarioLogic.

6. El método GetAll devuelve una lista de usuarios de la capa de entidad List<Usuario> y no recibe parámetros. Por lo tanto crearemos el método así
   public List<Usuario> GetAll()

7. El método GetAll debe invocar al método GetAll de UsuarioData y devolver lo mismo que este retorne.

8. El método GetOne debe recibir un ID y devolver un objeto de la clase Business.Entities.Usuario. Para ello deberá invocar al método GetOne de UsuarioData y devolver lo mismo que este.

9. El método Delete debe recibir un ID y no debe retornar ningún tipo (void). Dentro del mismo se debe invocar al método Delete de UsuarioData.

10. El método Save debe recibir un objeto de tipo Business.Entities.Usuario, devolver void e invocar el método Save de UsuarioData.

4. Crear la UI por consola para un ABMC simple en capas

Objetivos
Crear un proyecto de consola y desarrollar un ABMC de usuarios en capas que utilice los métodos desarrollados en el punto anterior

Duración Aproximada
60 minutos

Ahora procederemos a agregar la capa de interfaz en consola

1. Agregar un nuevo proyecto de tipo consola a la solución llamado UI.Consola.

2. Agregar las referencias a los proyectos Business.Logic y Business.Entities.

3. Crear la clase Usuarios y hacerla pública.

4. Agregar los using para las capas de lógica de negocio y entidades

5. Crear la propiedad UsuarioNegocio del tipo Business.Logic.UsuarioLogic e instanciarla en el constructor

6. Crear el método Menu, el mismo debe ofrecer por consola la siguiente lista de alternativas:
   1- Listado General
   2- Consulta
   3- Agregar
   4- Modificar
   5- Eliminar
   6- Salir

   Al elegir una opción, cada uno debe invocar al método correspondiente Luego de realizar una operación el menú debe volver a mostrarse.

7. Crear los métodos ListadoGeneral, Consultar, Agregar, Modificar, Eliminar y MostrarDatos.

8. El método ListadoGeneral debe invocar al método GetAll de UsuarioNegocio y que devuelve una List<Usuario> y recorrer la lista mostrando los datos de cada usuario invocando al método MostrarDatos:

```
public void ListadoGeneral()
{
    Console.Clear();
    foreach (Usuario usr in UsuarioNegocio.GetAll())
    {
        MostrarDatos(usr);
    }

}

public void MostrarDatos(Usuario usr)
{
    Console.WriteLine("Usuario: {0}", usr.ID);
    Console.WriteLine("\t\tNombre: {0}", usr.Nombre);
    Console.WriteLine("\t\tApellido: {0}", usr.Apellido);
    Console.WriteLine("\t\tNombre de Usuario: {0}", usr.NombreUsuario);
    Console.WriteLine("\t\tClave: {0}", usr.Clave);
    Console.WriteLine("\t\tEmail: {0}", usr.EMail);
    Console.WriteLine("\t\tHabilitado: {0}", usr.Habilitado);
    // \t dentro de un string representa un TAB
    Console.WriteLine();
}
```

9. A continuación en el método Main de la clase Program escribimos esta línea para invocar el método Menú que programamos:
   `new Usuarios().Menu();`

10. Ahora presionamos F5 para ejecutar y podremos probar que el ListadoGeneral funcione correctamente.

11. Luego de asegurarnos que funcione correctamente programaremos el método Consultar, el mismo solicitará al usuario que ingrese un ID y luego invocará al método GetOne de UsuariNegocio con dicho ID. En este punto recibirá un objeto de tipo Business.Entities.Usuario y deberá mostrarlo por pantalla invocando al método MostrarDatos. Quedándonos el siguiente código:

```
public void Consultar()
{
    Console.Clear();
    Console.Write("Ingrese el ID del usuario a consultar: ");
    int ID = int.Parse(Console.ReadLine());
    this.MostrarDatos(UsuarioNegocio.GetOne(ID));

}
```

12. Presionamos F5 y probamos el Consultar con los valores de ID 1, 2, 3, 4 y X.

Si programamos correctamente el método los valores 1, 2 y 3 deberían funcionar correctamente. Pero con 4 y X no. Con 4 falla porque no existe un usuario con ese ID y con X falla al tratar de convertir un carácter en un número.

13. Para resolver esto utilizaremos un try catch. Incluimos todo el código del método consultar dentro del try y luego agregamos un catch para FormatException y otro para Exception (también podría usarse una NullReferenceException para manejar el problema específico)
    En ambos bloques del catch agregaremos código para informar al usuario del error.

14. En el FormatException mostraremos la leyenda, "La ID ingresada debe ser un número entero".

15. En la Exception, mostraremos el mensaje de la excepción.

16. Agregaremos debajo de los catch un finally que muestre le leyenda "Presione una tecla para continuar" y utilice el Console.ReadKey() para la espera.

Entonces el código resultante debería ser similar a este:

```
public void Consultar()
{
    try
    {
        Console.Clear();
        Console.Write("Ingrese el ID del usuario a consultar: ");
        int ID = int.Parse(Console.ReadLine());
        this.MostrarDatos(UsuarioNegocio.GetOne(ID));

    }
    catch (FormatException fe)
    {
        Console.WriteLine();
        Console.WriteLine("La ID ingresada debe ser un número entero");
    }
    catch (Exception e)
    {
        Console.WriteLine();
        Console.WriteLine(e.Message);
    }
    finally
    {
        Console.WriteLine("Presione una tecla para continuar");
        Console.ReadKey();
    }
}
```

17. Presionamos F5 y volvemos a probar consultar los usuarios con ID 1, 2, 3, 4 y X

18. Ahora procederemos a crear el método Modificar. El mismo debe solicitar al usuario el ID, recuperar el objeto usuario usuario y cada uno de los datos, modificarlos en el objeto Usuario, establecer el estado como Modified e invocar al metodo Save de UsuarioNegocio. Siguiendo la estructura de antes nos queda un código como este:

```
public void Modificar()
{
    try
    {
        Console.Clear();
        Console.Write("Ingrese el ID del usuario a modificar: ");
        int ID = int.Parse(Console.ReadLine());
        Usuario usuario = UsuarioNegocio.GetOne(ID);
        Console.Write("Ingrese Nombre: ");
        usuario.Nombre = Console.ReadLine();
        Console.Write("Ingrese Apellido: ");
        usuario.Apellido = Console.ReadLine();
        Console.Write("Ingrese Nombre de Usuario: ");
        usuario.NombreUsuario = Console.ReadLine();
        Console.Write("Ingrese Clave: ");
        usuario.Clave = Console.ReadLine();
        Console.Write("Ingrese Email: ");
        usuario.EMail = Console.ReadLine();
        Console.Write("Ingrese Habilitación de Usuario (1-Si/otro-No): ");
        usuario.Habilitado = (Console.ReadLine()=="1");
        usuario.State = BusinessEntity.States.Modified;
        UsuarioNegocio.Save(usuario);

    }
    catch (FormatException fe)
    {
        Console.WriteLine();
        Console.WriteLine("La ID ingresada debe ser un número entero");
    }
    catch (Exception e)
    {
        Console.WriteLine();
```

Nota: la estructura de los catch y el finally es igual a la del Consultar.

19. Ahora presionamos F5 y probamos que funcione correctamente y que al consultar o listar los cambios hayan quedado registrados.

20. Luego creamos el método Agregar, el mismo debe instanciar un objeto de la clase Business.Entities.Usuario, solicitar todos los datos por pantalla menos el ID (que será asignado en la capa de datos por ser automático), asignarle el estado New e invocar al método Save de UsuarioNegocio y luego mostrar el ID que se le asignó al objeto.
    Entonces debería quedar un código similar al siguiente

```
public void Agregar()
{
    Usuario usuario = new Usuario();

    Console.Clear();
    Console.Write("Ingrese Nombre: ");
    usuario.Nombre = Console.ReadLine();
    Console.Write("Ingrese Apellido: ");
    usuario.Apellido = Console.ReadLine();
    Console.Write("Ingrese Nombre de Usuario: ");
    usuario.NombreUsuario = Console.ReadLine();
    Console.Write("Ingrese Clave: ");
    usuario.Clave = Console.ReadLine();
    Console.Write("Ingrese Email: ");
    usuario.EMail = Console.ReadLine();
    Console.Write("Ingrese Habilitación de Usuario (1-Si/otro-No): ");
    usuario.Habilitado = (Console.ReadLine()=="1");
    usuario.State = BusinessEntity.States.New;
    UsuarioNegocio.Save(usuario);
    Console.WriteLine();
    Console.WriteLine("ID: {0}",usuario.ID);

}
```

21. Presionamos F5 y probamos que funcione correctamente

22. Luego creamos el método Eliminar. El mismo debe solicitar por pantalla el ID del usuario a eliminar e invocar al método Delete de UsuarioNegocio. Debe realizar los mismos manejos de excepciones que el método modificar.
    El código resultante debería ser similar a este:

```
public void Eliminar()
{
    try
    {
        Console.Clear();
        Console.Write("Ingrese el ID del usuario a eliminar: ");
        int ID = int.Parse(Console.ReadLine());
        UsuarioNegocio.Delete(ID);

    }
    catch (FormatException fe)
    {
        Console.WriteLine();
        Console.WriteLine("La ID ingresada debe ser un número entero");
    }
    catch (Exception e)
    {
        Console.WriteLine();
        Console.WriteLine(e.Message);
    }
    finally
    {
        Console.WriteLine("Presione una tecla para continuar");
        Console.ReadKey();
    }
}
```

23. Presionamos F5 y probamos que funcione correctamente

Historia de Versiones

Fecha	Versión	Autor	Descripción
01/05/2011	1	AM	Versión inicial