UTN-FRA TECNICATURA EN PROGRAMACIÓN

Materia: Laboratorio de programación Nº2.
Profesor: Alejandro Bongioanni.
División: 2C 2023
Fecha: 23/11/2023
Alumno: Sebastian Serrano Belloso DNI: 42810404

PARCIAL Nº2
EJERCICIO INTEGRADOR FINAL

Introducción:
La aplicación simula un contexto de negocio, representa una agencia de turismo, ofrece funcionalidades para agregar pasajeros, reservas correspondientes a cada pasajero, mostrar listas de pasajeros y de reserva. Aplica los contenidos vistos luego del primer parcial de la cursada.

A continuación adjunto una imagen y una breve explicación de cómo fue implementado cada unidad de la materia en la aplicación.

1- Excepciones.
Encontramos las excepciones definidas dentro de el proyecto Entidades->Excepciones y un ejemplo de su implementación en Entidades->BaseDeDatos-> clase "ADOPasajeros.cs"

```
/// <summary>
/// Excepción personalizada para errores relacionados con la base de datos.
/// </summary>
8 referencias
public class BaseDeDatosException : Exception
{
    /// <summary>
    /// Inicializa una nueva instancia de la clase BaseDeDatosException con un mensaje específico.
    /// </summary>
    /// <param name="message">Mensaje que describe el error.</param>
    0 referencias
    public BaseDeDatosException(string? message) : base(message)
    {

    }
    /// <summary>
    /// Inicializa una nueva instancia de la clase BaseDeDatosException con un mensaje de error específico
    /// y una referencia a la excepción interna que generó esta excepción.
    /// </summary>
    /// <param name="message">Mensaje que describe el error.</param>
    /// <param name="innerException">Excepción interna que causó esta excepción.</param>
    6 referencias
    public BaseDeDatosException(string? message, Exception? innerException) : base(message, innerException)
    {

    }
```

```
2 referencias
public bool AgregarNuevoElemento(Pasajero elemento)
{
    try
    {
        using (SqlConnection connection = new SqlConnection(this.stringConnection))
        {
            string query = "INSERT INTO pasajeros (dni_pasajero,nombre,apellido,edad)" +
             "values (@dni_pasajero,@nombre,@apellido,@edad)";

            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("dni_pasajero", elemento.Dni);
            command.Parameters.AddWithValue("nombre", elemento.Nombre);
            command.Parameters.AddWithValue("apellido", elemento.Apellido);
            command.Parameters.AddWithValue("edad", elemento.Edad);

            connection.Open();

            command.ExecuteNonQuery();

            return true;
        }

    }
    catch (Exception ex)
    {
        throw new BaseDeDatosException("Error al agregar un pasajero",ex);

    }
}
```

2- Pruebas Unitarias.

Ubicación: Carpeta TestUnitarios-> proyecto TestAgenciaViajes-> clase TestUnitariosAgencia.cs

```
/// <summary>
/// Verifica que el cálculo del promedio de precio por reserva
/// devuelve el resultado esperado.
/// </summary>
[TestMethod]
0 referencias
public void PromedioDePrecioPorReserva_SeEsperaObtener_ElPromedioCorrespondiente()
{
    // Arrange
    AgenciaViajes agenciaViajes = new AgenciaViajes();

    List<Reserva> reservas = new List<Reserva>
    {
        new Reserva { MontoFinal = 100 },
        new Reserva { MontoFinal = 150 },
        new Reserva { MontoFinal = 200 }
    };

    agenciaViajes.Reservas = reservas;

    // Act
    string resultado = agenciaViajes.PromedioDePrecioPorReserva();

    // Assert
    string valorEsperado = "El promedio de precio por reserva para 3 reservas fue de: $150,00";
    Assert.AreEqual(valorEsperado, resultado);
}
```

3- Tipos Genericos.

Ubicación: Entidades/Interfaces/AdministradorBaseDatos.cs

```
/// <summary>
/// Define operaciones básicas para administrar elementos genéricos en una base de datos.
/// </summary>
/// <typeparam name="T">Tipo de elemento a administrar.</typeparam>
2 referencias
public interface IAdministradorBaseDeDatos<T>
{
    /// <summary>
    /// Obtiene un elemento por su número de documento de identidad (DNI) y el paquete seleccionado(EPaquetesViaje).
    /// </summary>
    /// <param name="dniElemento">Número de documento de identidad del elemento.</param>
    /// <param name="paqueteSeleccionado">Paquete seleccionado relacionado con el elemento.</param>
    /// <returns>Retorna el elemento si coincide con dniElemento.</returns>
    7 referencias
    T ObtenerElementoPorDNI(string dniElemento, string paqueteSeleccionado);
```

Se utilizan tipos genéricos para crear una abstracción, es decir, que cada clase implemente la interfaz con su tipo de datos permitiendo la administración de diferentes tipos de elementos en una base de datos de manera flexible y reutilizable.

4- Interfaces.

Ubicación: Entidades/Interfaces/AdministradorBaseDatos.cs

```
/// <summary>
/// Define operaciones básicas para administrar elementos genéricos en una base de datos.
/// </summary>
/// <typeparam name="T">Tipo de elemento a administrar.</typeparam>
2 referencias
public interface IAdministradorBaseDeDatos<T>
{
    /// <summary>
    /// Obtiene un elemento por su número de documento de identidad (DNI) y el paquete seleccionado(EPaquetesViaje).
    /// </summary>
    /// <param name="dniElemento">Número de documento de identidad del elemento.</param>
    /// <param name="paqueteSeleccionado">Paquete seleccionado relacionado con el elemento.</param>
    /// <returns>Retorna el elemento si coincide con dniElemento.</returns>
    7 referencias
    T ObtenerElementoPorDNI(string dniElemento, string paqueteSeleccionado);

    /// <summary>
    /// Obtiene todos los elementos almacenados en la base de datos.
    /// </summary>
    /// <returns>Una lista de todos los elementos almacenados.</returns>
    10 referencias
    List<T> ObtenerTodosLosElementos();

    /// <summary>
    /// Agrega un nuevo elemento a la base de datos.
    /// </summary>
    /// <param name="elemento">Elemento a agregar.</param>
    /// <returns>True si se agregó correctamente; de lo contrario, False.</returns>
    5 referencias
    bool AgregarNuevoElemento(T elemento);

    /// <summary>
    /// Elimina un elemento por su número de documento de identidad (DNI) y el paquete seleccionado.
    /// </summary>
    /// <param name="dniElemento">Número de documento de identidad del elemento a eliminar.</param>
    /// <param name="paqueteSeleccionado">Paquete seleccionado relacionado con el elemento.</param>
    /// <returns>True si se eliminó correctamente; de lo contrario, False.</returns>
    4 referencias
    bool EliminarElementoPorDNI(string dniElemento, string paqueteSeleccionado);
```

Las clases ADOPasajeros y ADOReservas implementan el contrato de esta interfaz, cada una desarrolla una lógica personal para cumplir con este funcionamiento.

5- Archivos.

Ubicación: Entidades/Archivos/GestorArchivosAgencia.cs

```
/// <summary>
/// Guarda la lista de pasajeros en un archivo con formato JSON en la ruta especificada.
/// </summary>
/// <param name="listaDePasajeros">Lista de pasajeros a guardar.</param>
/// <param name="nombreArchivo">Nombre del archivo de salida.</param>
/// <returns>True si la operación de guardado fue exitosa; de lo contrario, False.</returns>
7 referencias
public static bool GuardarPasajerosEnArchivo(List<Pasajero> listaDePasajeros, string nombreArchivo)
{
    JsonSerializerOptions options = new JsonSerializerOptions();
    options.WriteIndented = true;

    string directorio = GestorArchivosAgencia.ObtenerRutaPorDefecto();
    string rutaCompleta = Path.Combine(directorio, $"{nombreArchivo}.json");

    try
    {

        if (!Directory.Exists(directorio))
        {
            Directory.CreateDirectory(directorio);
        }

        using (StreamWriter sw = new StreamWriter(rutaCompleta))
        {
            string listaJson = JsonSerializer.Serialize(listaDePasajeros, options);
            sw.WriteLine(listaJson);
        }

        return true;
    }
    catch (Exception ex)
    {
        throw new ManejoDeArchivosException(nombreArchivo, ex);
    }

}
```

La clase GestorArchivosAgencia contiene métodos de lectura/escritura para Pasajeros y Reservas.

6- Serialización.

Ubicación: Entidades/Archivos/GestorArchivosAgencia.cs

Como puede apreciarse en la imagen del inciso anterior(archivos), el archivo guarda una lista serializada en formato Json. El método Serialize recibe como parámetros la List<> que guardará en el archivo, y options le indica que represente cada objeto de manera indentada. (En la clase también se encuentra la lectura de un archivo Json).

7- SQL y Conexión a bases de datos.

Ubicación: Entidades/BaseDeDatos/ADOPasajeros.cs

```
public ADOPasajeros()
{
    this.stringConnection = "Server=.;Database=INTEGRADOR_AgenciaTurismo_DB;Trusted_Connection=True;";
}

/// <summary>
/// Agrega un nuevo pasajero a la base de datos.
/// </summary>
/// <param name="elemento">El pasajero a agregar.</param>
/// <returns>True si se agregó correctamente; de lo contrario, False.</returns>
2 referencias
public bool AgregarNuevoElemento(Pasajero elemento)
{
    try
    {
        using (SqlConnection connection = new SqlConnection(this.stringConnection))
        {
            string query = "INSERT INTO pasajeros (dni_pasajero,nombre,apellido,edad)" +
             "values (@dni_pasajero,@nombre,@apellido,@edad)";

            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@dni_pasajero", elemento.Dni);
            command.Parameters.AddWithValue("@nombre", elemento.Nombre);
            command.Parameters.AddWithValue("@apellido", elemento.Apellido);
            command.Parameters.AddWithValue("@edad", elemento.Edad);

            connection.Open();

            command.ExecuteNonQuery();

            return true;
        }

    }
    catch (Exception ex)
    {
        throw new BaseDeDatosException("Error al agregar un pasajero",ex);

    }
}
```

Comparte implementación de interfaz con ADOReservas. Se conecta a la base de datos con un stringConecction por defecto y realiza la query correspondiente para la funcionalidad de cada método.

8- Delegados y expresiones lambda.

Ubicación: Entidades/AgenciaViajes.cs; Formularios/FrmVista/FrmMenuPrincipal; Formularios/FrmVista/FrmAltaCliente .

En estas clases se utilizan delegados, en FrmMenuPrincipal para aplicar Concurrencia(también una expresión lambda) y en FrmAltaCliente para disparar un evento MostrarDescripcion.

En la clase AgenciaViajes: Se utiliza para disparar un evento en el formulario FrmAltaCliente.

```
// <summary>
/// Delegado utilizado para mostrar la descripción de paquetes de viaje.
/// </summary>
/// <param name="sender">El objeto que activa el evento.</param>
/// <param name="e">Argumentos del evento.</param>
public delegate void MostrarDescripcionPaquetes(object sender, EventArgs e);
```

En el formulario FrmMenuPrincipal: delegado action se le asigna función anónimas sin parámetros ni retorno.

```
/// <summary>
/// Muestra un mensaje de éxito en un label de el formulario Menu Principal,cuando se carga el registro de pasajeros.
/// </summary>
1 referencia
private void MostrarLabelExito()
{

    if (this.lblCargaDeRegistros.InvokeRequired)
    {
        Action mostrarExito = () =>
        {
            this.lblCargaDeRegistros.Visible = true;
            this.lblCargaDeRegistros.Text = "Registro de pasajeros cargado.";
        };
        this.Invoke(mostrarExito);
    }
    else
    {
        this.lblCargaDeRegistros.Visible = true;
        this.lblCargaDeRegistros.Text = "Registro de pasajeros cargado";
    }

}
```

9- Programación multihilo y concurrencia.
Ubicación: Formularios/FrmVista/FrmMenuPrincipal

```
/// <summary>
/// Inicia una tarea en segundo plano(hilo,thread) para validar los registros de la agencia de viajes.
/// </summary>
1 referencia
private void CargarListaEnSegundoPlano()
{
    Thread tareaSecundaria = new Thread(ValidarRegistrosDeAgenciaViajes);
    tareaSecundaria.Start();
}

/// <summary>
/// Método que valida los registros de pasajeros y reservas de la agencia de viajes en segundo plano.
/// Muestra un mensaje indicando el proceso y los resultados obtenidos.
/// </summary>
1 referencia
private void ValidarRegistrosDeAgenciaViajes()
{
    StringBuilder sb = new StringBuilder();

    Thread.Sleep(3000);
    MessageBox.Show("Se estan cargando registros de pasajeros en segundo plano...");

    if (this.agencia.Pasajeros is not null && this.agencia.Reservas is not null)
    {
        sb.AppendLine("Registro de pasajeros cargado con exito.");


        sb.AppendLine($"Existen {this.agencia.Pasajeros.Count} pasajeros en lista.");
        sb.AppendLine($"Existen {this.agencia.Reservas.Count} reservas registradas.");
        sb.AppendLine($"Desde el menu principal, puede acceder a Registro de Pasajeros y Registro de Reservas para verlos.");
    }
    else
    {
        sb.AppendLine("No se pudo cargar el registro de pasajeros.");
    }

    Thread.Sleep(5000);

    this.MostrarLabelExito();

    MessageBox.Show(sb.ToString(), "Tarea en segundo plano finalizada.");
}
```

Al iniciarse la aplicación, genera en un hilo secundario la corroboración de la carga de los registros de Pasajeros y Reservas. Al terminar informa por MessageBox y también hace visible un label que indica que indica que se cargó la lista(Esto lo hace utilizando el método MostrarLabelExito() que utiliza delegados y una expresión lambda; inciso 9, imagen adjunta).

10- Eventos.

Ubicación: Formularios/FrmVista/FrmAltaCliente

Delegado y Event declarado en Entidades/Agencia Viajes.cs

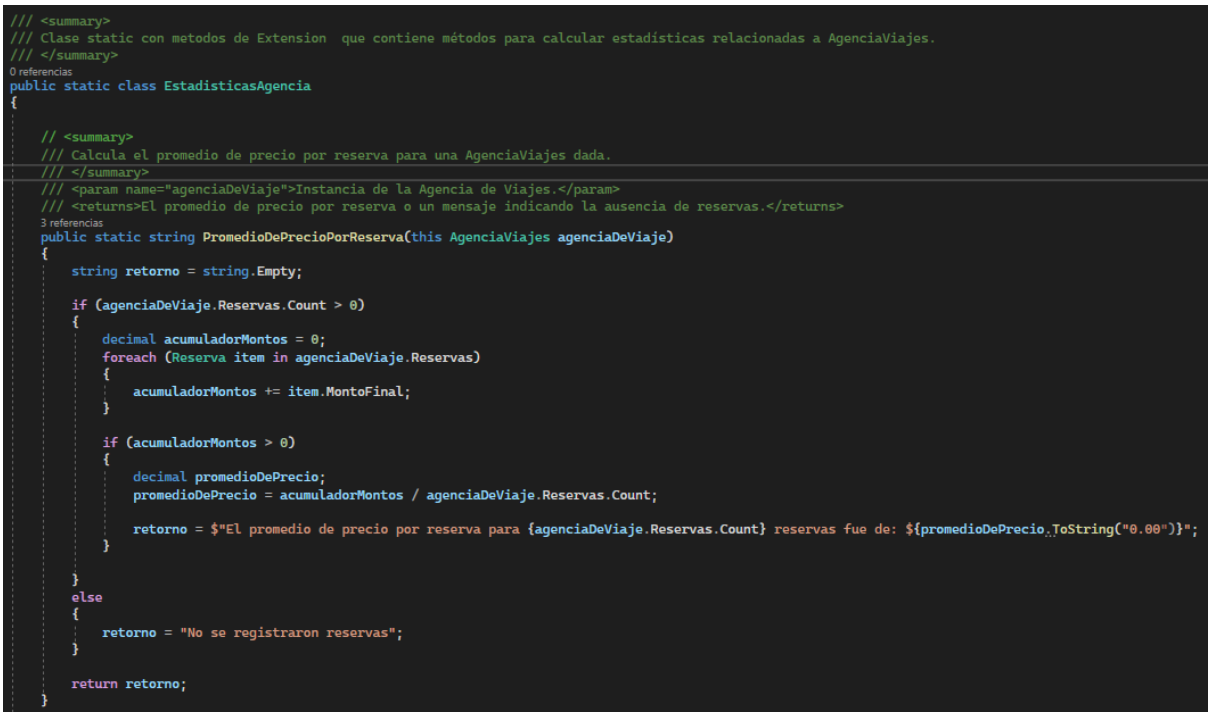
```
/// <summary>
/// Maneja el evento load del formulario "FrmAltaCliente".
/// Establece la selección predeterminada del RadioButton y suscribe un método de AgenciaViajes
/// </summary>
/// <param name="sender">Objeto que desencadenó el evento.</param>
/// <param name="e">Argumentos del evento.</param>
1 referencia
private void FrmAltaCliente_Load(object sender, EventArgs e)
{
    this.rdbBronce.Checked = true;
    this.agencia.botonMostrarDescripcionPulsado += AgenciaViajes_MostrarDescripcionPaquetes;
}
```

```
/// <summary>
/// Maneja el evento de clic en el botón para mostrar la descripción de los paquetes de viaje.
/// Invoca un método de AgenciaViajes que contiene un delegado encargado de invocar el metodo, para mostrar la descripción de los paquetes, si está disponible.
/// En caso contrario, muestra un mensaje indicando que no se ha cargado la información.
1 referencia
private void btnDescripcionPaquetes_Click(object sender, EventArgs e)
{
    try
    {
        if (this.agencia.InvocarMostrarDescripcion(sender, e))
        {
        }
        else
        {
            MessageBox.Show("No se a cargado informacion que describa los paquetes");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error al mostrar descripcion", MessageBoxButtons.OK);
    }

}

/// <summary>
/// Maneja el evento para mostrar la descripción de los paquetes de viaje.
/// Muestra la descripción de los paquetes utilizando un mensaje emergente (MessageBox).
/// </summary>
/// <param name="sender">Objeto que desencadenó el evento.</param>
/// <param name="e">Argumentos del evento.</param>
1 referencia
private void AgenciaViajes_MostrarDescripcionPaquetes(object sender, EventArgs e)
{
    MessageBox.Show(this.agencia.DescripcionPaquetes(),"Descripcion paquetes", MessageBoxButtons.OK,MessageBoxIcon.Information);
}
```

El evento click llama al método InvocarMostrarDescripcion que invoca un evento a traves de el delegado botonMostrarDescripcionPulsado(declarado en la clase AgenciaViajes) al cual está suscrito MostrarDescripcionPaquetes que muestra por pantalla la información de paquetes los distintos paquetes de viaje que pueden seleccionarse al dar de alta una reserva.

11- Métodos de extensión.

Ubicación: Entidades/MetodosDeExtension/EstadisticasAgencia.cs



```
/// <summary>
/// Clase static con metodos de Extension  que contiene métodos para calcular estadísticas relacionadas a AgenciaViajes.
/// </summary>
0 referencias
public static class EstadisticasAgencia
{

    // <summary>
    /// Calcula el promedio de precio por reserva para una AgenciaViajes dada.
    /// </summary>
    /// <param name="agenciaDeViaje">Instancia de la Agencia de Viajes.</param>
    /// <returns>El promedio de precio por reserva o un mensaje indicando la ausencia de reservas.</returns>
    3 referencias
    public static string PromedioDePrecioPorReserva(this AgenciaViajes agenciaDeViaje)
    {
        string retorno = string.Empty;

        if (agenciaDeViaje.Reservas.Count > 0)
        {
            decimal acumuladorMontos = 0;
            foreach (Reserva item in agenciaDeViaje.Reservas)
            {
                acumuladorMontos += item.MontoFinal;
            }

            if (acumuladorMontos > 0)
            {
                decimal promedioDePrecio;
                promedioDePrecio = acumuladorMontos / agenciaDeViaje.Reservas.Count;

                retorno = $"El promedio de precio por reserva para {agenciaDeViaje.Reservas.Count} reservas fue de: ${promedioDePrecio.ToString("0.00")}";
            }

        }
        else
        {
            retorno = "No se registraron reservas";
        }

        return retorno;
    }
```

Contiene métodos que calculan estadísticas valiéndose de la clase AgenciaViajes que tiene los registros de pasajeros y sus correspondientes reservas.