**EXAMEN .NET**

**MODULO2**

**PRINCIPIO1(RESPONSABILIDAD UNICA)**

**Ejercicio1 (ejemplo semana 5 dia 1 carpeta content)**

**Before**

Vamos a crear una aplicación de reservas de hotel donde necesitamos crear una clase **BookingService**, y crear un método **Create** dentro de dicha clase, donde le pasemos por argumentos un **nombre**(string), un **email**(string), y una fecha de **check-in(**dateTime) y **check-out**(dateTime). Además, debemos hacer las validaciones correspondientes de cada parámetro **(nombre no vacío, email válido, fechas válidas).** Luego, si todo es válido, se procederá a crear la reserva del hotel.

Este ejercicio será desarrollado inicialmente dentro de una sola clase -> lo cual no es correcto ya que viola el primer principio de responsabilidad única (SRP). Posteriormente, se debe **refactorizar** el código para cumplir con el principio de Responsabilidad única(SRP).

**AFTER**

Implementar una clase **Usuario**, una clase **Booking**(definir las propiedades de la reserva), una clase **ValidationResult**, y una clase **BookingServiceValidator** y la clase que realiza la reserva **BookingService**

**PRINCIPIO2(Abierto-Cerrado)**

**Ejercicio2**

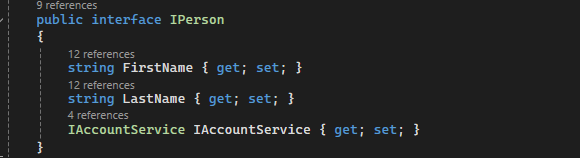
**Before**

Vamos a desarrollar un sistema de gestión de membresías en un gimnasio. Inicialmente, tendremos una serie de personas por lo que crearemos nuestra clase **Person** con los siguientes atributos: **FirstName**(string), **LastName**(string), **MembresiaType**(Enum) -> **Basico, Gold, Platinium**. Luego necesitaremos implementar la clase **Membership** con los siguientes atributos: **FirstName**(string), **LastName**(string), **Email**(string), **MembresiaType** (Enum), **Id**(sring) y **Price**(double)

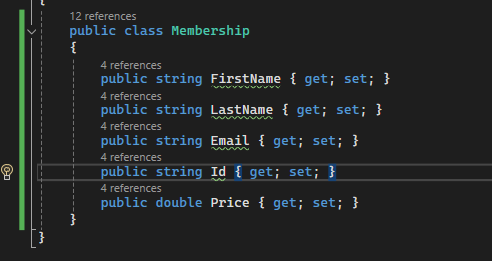
Necesitamos implementar una clase **AccountService** con un método **Create** que cree la membresía especificando un **correo** por cada tipo de membresía de cada persona y asigne un **ID** de membresía y un **precio** dependiendo del tipo de membresía.

**After**

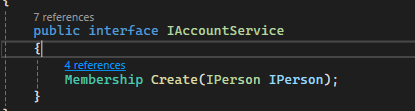
* Vamos a eliminar el enum **MembresiaType** y en vez de este enum crearemos una interfaz **IPerson** que tendrá mi interfaz **IAccountService** y las propiedades **FirstName** y **LastName** como se indica en la figura



* Crearemos nuestra clase **Membership** con las siguientes propiedades



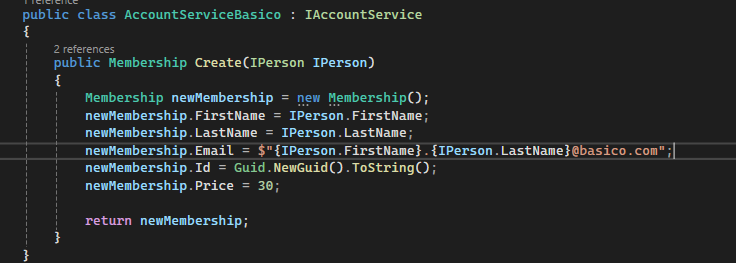
* La interfaz **IAccountService** proporcionara un método **Create** de tipo **Membreship** y que recibirá como parámetros una **Persona** y que me permitirá implementar la lógica dependiendo del tipo de menbresia ya sea **Basico**, **Gold** o **Platinium.**



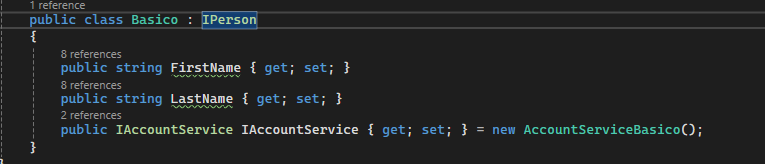
Las clases que se implementarán a partir de esta interfaz serán:

* **AccountServiceBasico**
* **AccountServiceGold**
* **AccountServicePlatinium**

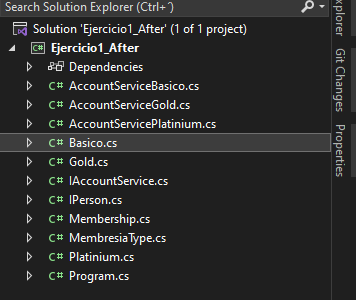
Seguir el ejemplo de implementación de lógica para cada clase:

****

Por ultimo implementaremos nuestras clases **Basico**, **Gold** y **Platinium**, como sigue:

****

La estructura de carpetas deberá ser así:

****

**PRINCIPIO3(Sustitucion de LISKOV)**

**Ejercicio3 (guía semana 5 miercoles LSP EJERCICIO 1 AFTER)**

**Before**

Tenemos un programa donde vamos a gestionar diferentes tipos de empleados y sus departamentos (areas ejp: contabilidad, finanzas). Para ello, vamos a tener una clase **Employee** que tiene los siguientes atributos: **FirstName(string)**, **LastName(string)**, **Department((Department)**, un enum **ContractType** (ReciboPorHonorarios, Planilla). y un método para obtener el departamento (**GetDepartment**), un método para asignar el departamento (**AssignDepartment**), y un método para cambiar el tipo de empleo (**SetContractType**).

La clase **Departamento** simplemente tendrá una propiedad **Name.**

Luego tendremos una clase hija **FullTime** que hereda de **Employee** y tiene un método **GetSalario** donde obtenemos el **salario fijo asignado** en caso pertenezca a un **departamento**. Luego tendremos una clase **PartTime** que también heredará de **Employee** y donde lanzará una excepción cuando se implemente **AssignDepartment** ya que un **Practicante** no necesita un departamento.

La lógica consiste en que al asignar un departamento a mi empleado de tipo **FullTime** automáticamente este pasará a estar en planilla, de lo contrario si no se asigna un departamento será recibo por honorarios

**AFTER**

Lo primero que se tendría que realizar es determinar todo lo que es común en ambas clases para tenerlo presente en la interfaz base **IEmployee**, asi luego se implementara la interfaz **IEmployeeDepartment** para las propiedades adicionales en caso se asigne un **departamento** y por ende se encuentre en **Planilla**.

La relación implica que si un empleado tiene asociado un departamento este automáticamente estará en Planilla, y por lo tanto un salario.

Como vemos la propiedad **Department**, los métodos de **obtener Department y asignar Department** son solo propias de la clase **FullTime** asi como el método **GetSalario.**