Primer Trabajo Integrador

Seminario de Lenguajes opción .NET 2022

Importante: Desarrollar con la versión **.NET 6.0**. No deshabilitar **<ImplicitUsings>** ni **<Nullable>** en los archivos de proyecto (**.csproj**). Resolver todos los *warnings* del compilador respecto de las referencias **null**.

Fecha de publicación: 2/5/2022.
Fecha límite para la entrega: 15/5/2022.

Se debe desarrollar una biblioteca de clases (como se explicó en la teoría 4) que se utilizará para desarrollar una aplicación para un local de venta de electrodomésticos. Por el momento sólo se manejará información de clientes y empleados.

Los datos de clientes y empleados se deben persistir en los archivos de texto: ***Clientes.txt*** y ***Empleados.txt***. Cada línea de estos archivos representará la información de un cliente o empleado respectivamente y estará estructurada de la siguiente manera:

- **Clientes**: DNI|Apellido|Nombre|Direccion|FechaDeNacimiento|FechaUltimaCompra
- **Empleados**: DNI|Apellido|Nombre|Direccion|FechaDeNacimiento|Legajo

Observar que cada atributo de Cliente y Empleado se separa por una barra vertical (**|**) en el archivo de texto.

Para la implementación de la biblioteca de clases se deberán considerar los siguientes aspectos técnicos:

Se deben declarar las clases **Empleado** y **Cliente**, ambas derivadas de la clase **Persona** (No debe ser posible crear instancias de la clase Persona). De una Persona se conoce nombre, apellido, número de documento, dirección, y fecha de nacimiento. El Empleado, además de los datos de Persona, posee el número de legajo. El Cliente, además de los datos de Persona, posee la fecha de la última compra.

Se deben definir los *constructores* que permitan crear las instancias de Empleado y Cliente inicializando sus campos con datos recibidos como parámetro. También se debe implementar las propiedades para acceder a cada campo de las clases. Además, para el caso del cliente, se debe proveer de una propiedad de solo lectura que permita calcular en días, el tiempo desde la última compra del cliente. Invalidar el método ToString() en las clases que considere necesario (tener el cuenta el video con la demostración sobre cómo se debe utilizar la biblioteca de clases)

Se deben declarar las siguientes **interfaces y clases** para implementar la persistencia de datos en los archivos mencionados previamente:

- Interfaz **IRepositorioCliente** (provista por la Cátedra)
- Interfaz I**RepositorioEmpleado** definida de forma similar a IRepositorioCliente
- Clase **RepositorioClienteArchTexto** que implemente IRepositorioCliente
- Clase **RepositorioEmpleadoArchTexto** que implemente IRepositorioEmpleado

Utilizar esta definición de la interfaz **IRepositorioCliente**

```
public interface IRepositorioCliente
{
   void AgregarCliente(Cliente cliente);
   List<Cliente> GetClientes();
   Cliente? GetCliente(int DNI);
   void ModificarCliente(Cliente cliente);
   void EliminarCliente(int Dni);
}
```

Los repositorios concretos son quienes tienen la responsabilidad de la persistencia de los datos en los archivos de texto correspondientes. En particular, para el caso de **RepositorioClienteArchTexto** se detalla la funcionalidad de los métodos que deben implementarse:

- **void AgregarCliente(Cliente cliente)**: Agrega al archivo Clientes.txt los datos del cliente que se pasa por parámetro.
- **List<Cliente> GetClientes()**: obtiene del archivo Clientes.txt la lista completa de clientes guardados.
- **Cliente? GetCliente(int DNI)**: obtiene del archivo Clientes.txt el cliente cuyo DNI se pasa por parámetro. En caso de no existir devuelve null.
- **void ModificarCliente(Cliente cliente)**: modifica en el archivo Clientes.txt los datos del cliente pasado como parámetro.
- **void EliminarCliente(int Dni)**: elimina del archivo Clientes.txt al cliente que posea el DNI que se pasa por parámetro.

Implementar la funcionalidad de la aplicación a partir de los siguientes casos de uso. Cada caso de uso debe codificarse por medio de una clase distinta (por convención estas clases llevarán el sufijo UseCase).

- Clase **AgregarClienteUseCase** con el método **void Ejecutar(Cliente cli)** que agrega en el repositorio al cliente **cli** recibido como parámetro. Si el cliente (identificado por el DNI) ya existe en el repositorio se debe lanzar una excepción.
- Clase **ModificarClienteUseCase** con el método **void Ejecutar(Cliente cli)** que actualiza en el repositorio los datos del cliente **cli** recibido por parámetro (el cliente se identifica por su DNI). Si el cliente no existe en el repositorio se debe lanzar una excepción.
- Clase **EliminarClienteUseCase** con el método **void Ejecutar(int DNI)** que elimina del repositorio al cliente cuyo DNI se recibe como parámetro. En caso de no existir ningún cliente con ese DNI se debe lanzar una excepción.
- Clase **ListaDeClientesUseCase** con el método **List<Cliente> Ejecutar()** que devuelve la lista de todos los clientes.

De manera similar codificar las clases y métodos necesarios para implementar los casos de uso para los empleados.

Además implementar el siguiente caso de uso codificando la clase **ListaDePersonasUseCase** con el método **List<Persona> Ejecutar()** que devuelve todas las personas (empleados y clientes) ordenadas por apellido (primer criterio) y nombre (segundo criterio). Nota: en el ordenamiento no se debe tener en cuenta si la persona es un empleado o un cliente.

Nota1: Para actualizar los datos en los archivos de texto no se debe tener en cuenta aspectos de rendimiento. Es decir que cada vez que sea necesario actualizar la información de un empleado o cliente puede sobrescribirse completamente el archivo correspondiente.

Nota2: Observar que las clases que implementan casos de uso no son quienes acceden de manera directa a los datos persistidos, sino que lo hacen a partir de repositorios. Es una buena práctica que la lógica de la aplicación (implementada en los casos de uso) no conozca detalles sobre cómo se persisten los datos. De esta forma más adelante será fácil reemplazar los repositorios concretos **RepositorioClienteArchTexto** y **RepositorioEmpleadoArchTexto** por otros que implementen las mismas interfaces pero que persistan los datos en otro medio, por ejemplo en una base de datos relacional. Estos beneficios se van a comprender mejor cuando veamos el principio de “Inversión de Dependencias” más adelante en este curso.