Trabajo Práctico Seminario .Net

Diseño técnico

# Autores del programa y documento:

## Matías Iván González - Legajo: 17627/1

* Julián Daniel Gómez - Legajo: 16851/2

# Use Cases

## Clientes

Para poder crear y manipular los Use Case para Clientes, primero hay que instanciar un objeto del tipo <RepositorioClienteTXT>( que extiende de una interfaz <IRepositorio>), que será pasado como parámetro del constructor de cada uno de los Use Cases que se vayan a utilizar (Inyección de dependencia por constructor). Ejemplo:  
  
var repoCliente = new RepositorioClienteTXT();  
  
Estos son:

* + 1. AgregarClienteUseCase:  
       Requiere instanciar un objeto del tipo <AgregarClienteUseCase> y luego invocar su método Ejecutar(Cliente cli), que recibirá un cliente instanciado “cli”, el cual será insertado en el archivo de texto clientes.txt, sólo luego de validar su inexistencia previa y formato adecuado de datos.  
       Ejemplo:  
       var persona1 = new ClienteFisico("30321654")  
       ClienteUseCase agregarCliente = new AgregarClienteUseCase();

{

Nombre = "juan",

Direccion = "Diag.74 nro 123",

Telefono = "(11)502-1111"

};  
agregarCliente.Ejecutar(cli);

Este método no produce salida por consola, a menos que se trate de un log correspondiente a una excepción.

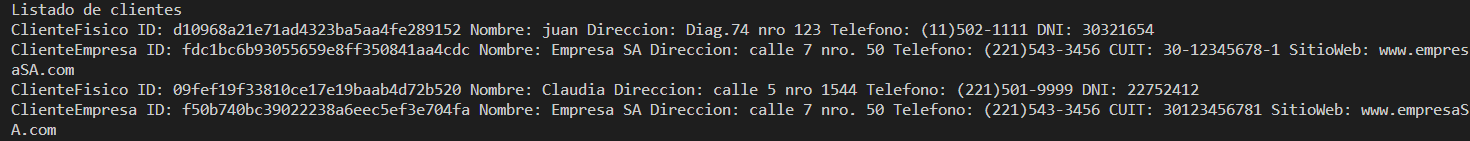
* + 1. ModificarClienteUseCase:  
       Requiere instanciar un objeto del tipo <ModificarClienteUseCase> y luego invocar su método Ejecutar(Cliente cli), que recibirá un cliente modificado “cli”, el cual será buscado y modificado en el archivo de texto clientes.txt, sólo luego de validar su existencia previa y formato adecuado de datos.  
       Ejemplo:  
         
       persona1.Nombre = "juan Perez",  
       ClienteUseCase modificarCliente = new ModificarClienteUseCase();

Direccion = "Diag.74 nro 321",

Telefono = "(11)502-0000"

modificarCliente.Ejecutar(cli);

Este método no produce salida por consola, a menos que se trate de un log correspondiente a una excepción

* + 1. EliminarClienteUseCase:  
       Requiere instanciar un objeto del tipo <EliminarClienteUseCase> y luego invocar su método Ejecutar(string id), que recibirá un id del tipo <string>, el cual será buscado dentro del archivo de texto clientes.txt, y en caso de existir, eliminado.  
       Ejemplo:  
         
       ClienteUseCase eliminarCliente = new EliminarClienteUseCase();  
       eliminarCliente.Ejecutar(persona1.Id);  
         
       Este método no produce salida por consola, a menos que se trate de un log correspondiente a una excepción
    2. ListarClientesUseCase:  
       Requiere instanciar un objeto del tipo <ListarClientesUseCase> y luego invocar su método Ejecutar(), que no recibirá ningún parámetro. Este método imprimirá por consola todos los clientes registrados dentro del archivo clientes.txt.  
       Ejemplo:  
         
       ClienteUseCase listarClientes = new ListarClientesUseCase();  
       listarClientes.Ejecutar();  
         
       Posible salida por Consola:  
       

## Productos

Para poder crear y manipular los Use Case para Productos, primero hay que instanciar un objeto del tipo <RepositorioProductoTXT>( que extiende de una interfaz <IRepositorio>), que será pasado como parámetro del constructor de cada uno de los Use Cases que se vayan a utilizar (Inyección de dependencia por constructor). Ejemplo:  
  
var repoProducto = new RepositorioProductoTXT();  
  
Estos son:

* + 1. AgregarProductoUseCase:  
       Requiere instanciar un objeto del tipo <AgregarProductoUseCase> y luego invocar su método Ejecutar(Producto prod), que recibirá un producto instanciado “prod”, el cual será insertado en el archivo de texto productos.txt, sólo luego de validar su inexistencia previa y formato adecuado de datos.  
       Ejemplo:  
       var producto1 = new Producto("Rodillo Grande")  
       ProductoUseCase agregarProducto = new AgregarProductoUseCase();

{

PrecioUnitario = "750.50",

Stock = "50000"

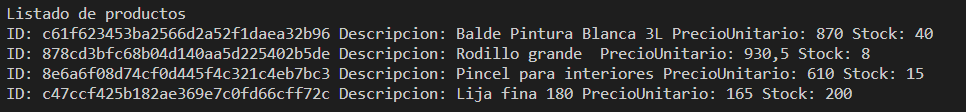
};  
agregarProducto.Ejecutar(producto1);

Este método no produce salida por consola, a menos que se trate de un log correspondiente a una excepción.

* + 1. ModificarProductoUseCase:  
       Requiere instanciar un objeto del tipo <ModificarProductoUseCase> y luego invocar su método Ejecutar(Producto prod), que recibirá un producto modificado “prod”, el cual será buscado y modificado en el archivo de texto productos.txt, sólo luego de validar su existencia previa y formato adecuado de datos.  
       Ejemplo:  
         
       producto1.PrecioUnitario = "850.30";  
       producto1.Stock = "30000";  
       ProductoUseCase modificarProducto = new ModificarProductoUseCase();

modificarProducto.Ejecutar(producto1);

Este método no produce salida por consola, a menos que se trate de un log correspondiente a una excepción

* + 1. EliminarProductoUseCase:  
       Requiere instanciar un objeto del tipo <EliminarProductoUseCase> y luego invocar su método Ejecutar(string id), que recibirá un id del tipo <string>, el cual será buscado dentro del archivo de texto productos.txt, y en caso de existir, eliminado.  
       Ejemplo:  
         
       ProductoUseCase eliminarProducto = new EliminarProductoUseCase();  
       eliminarProducto.Ejecutar(producto1.Id);  
         
       Este método no produce salida por consola, a menos que se trate de un log correspondiente a una excepción
    2. ListarProductosUseCase:  
       Requiere instanciar un objeto del tipo <ListarProductosUseCase> y luego invocar su método Ejecutar(), que no recibirá ningún parámetro. Este método imprimirá por consola todos los productos registrados dentro del archivo productos.txt.  
       Ejemplo:  
         
       ProductoUseCase listarProductos = new ListarProductosUseCase();  
       listarProductos.Ejecutar();  
         
       Posible salida por Consola:  
       

# Hashed IDs

En la solución al problema se han utilizado ciertas variables no modificables (DNI para ClienteFisico; CUIT para ClienteEmpresa y Descripción para Producto) con el fin de generar códigos hash a modo de ID único.

Se utilizan como si fueran claves primarias de una base de datos (nuestra persistencia en el repositorio de clientes.txt, productos.txt) y para ello se utilizan esos campos como entradas de datos en un algoritmo de hasheo MD5. De esta manera nos aseguramos de generar un código de ID único para cada entidad correspondiente que sea irrepetible y no genere conflictos.

**NOTA**: Se adjunta el respectivo archivo .zip. En caso de existir problemas con la descompresión del archivo, también se deja la dirección en github del proyecto:  
<https://github.com/matias-ivan-gonzalez/sistema-de-gestion-de-pintureria>