**TP 3 - 4 / LABORATORIO II**

Micaela Sol Vazzana


***INDICE***



***Enchulame la cámara Rental APP*** consiste en una aplicación de alquiler de equipos de cine y video destinado a productoras cinematográficas, realizadores audiovisuales independientes o estudiantes de carreras afines.
En esta aplicación podremos entonces realizar reservas accediendo a un listado de equipos que la empresa tiene en stock.
La aplicación además permite realizar modificaciones sobre esa reserva, en caso de que el cliente quiera cancelar algún equipo que haya decidido no necesitar o en cambio, sumar más equipos a su reserva (siempre que estos estén en stock).
Por otra parte, podremos dar de baja la reserva si es que el cliente decide cancelarla, o bien si el cliente procede a la devolución una vez que haya terminado su jornada de alquiler.
La aplicación también nos permite guardarnos un listado de clientes y sus datos para poder tener una base de datos que servirá para realizar mailing con promociones.

## *PARTE 1: TP 3*

### *EXCEPCIONES*

Controlo las excepciones que pueden arrojarme aquellas instancias de objetos como el ***StreamWriter***, ***StreamReader***, arrojando una nueva excepción llamada ***ArchivoException***, donde además guardo la excepción original con la propiedad InnerException.
También controlo las excepciones que me pueden arrojar los métodos como el ***Parse*** o cuando al hacer una búsqueda de índice esta esté fuera de rango. Cada posibilidad de uso de métodos que puedan arrojar excepciones controlo guardando la excepción con una excepción genérica que me permita saber qué sucedió.

### *PRUEBAS UNITARIAS*

Realizo los test de las excepciones que puedan arrojarme los métodos que gestionaran los archivos. Verifico de esta manera, que la excepción ***ArchivoException*** creada por mí sea propiamente lanzada al manipular esos métodos.
Por otra parte, realizo test para controlar que los métodos que desarrollé en mi biblioteca de clases ***Entidades*** funcionen correctamente.

### *TIPOS GENERICOS*

Realicé una clase gestora de archivos xml y json llamada ***Serializadora.*** Esta contiene los métodos que van a manejar estos distintos tipos de archivos.
Esta clase es genérica, ya que voy a utilizar los métodos que serializan con distintos tipos de objetos. Esto lo indico a la hora de utilizarlos y serán serializados o deserializados según esa indicación.

### *INTERFACES*

La utilización de interfaces está implementada por un lado, en los formularios al mostrar un mensaje de error, que será el mensaje que contendrá el mensaje de las excepciones controladas, a su vez se muestran las ***InnerException***. Estas se mostrarán a través de un ***Message.Box.*** Por otra parte, implementé un método que hace el cálculo del costo total de la reserva. Si este cliente es estudiante, el cálculo aplicará un 15% de descuento.

### *ARCHIVOS*

La creación y lectura de archivos de texto se gestionan desde la clase ***GestorArchivosTexto***. Esta es utilizada de distintas maneras. En primer lugar es utilizada para crear números auto-incrementales. Creo ids para el número de reserva y creo números de recibo. En segundo lugar, es utilizada para crear esos recibos que serán el comprobante del cliente.
Estos contienen todos los datos de la reserva. Los guardo en un subdirectorio: ***Recibos***.

### *SERIALIZACIÓN*

La creación y lectura de archivos ***xml*** y ***json*** se gestionan desde la clase ***Serializadora***.
Los archivos ***xml*** son gestionados para guardar o leer los datos de los clientes. En el caso de la lectura, solo podremos ver los mails de los clientes a través del botón ***Mails Clientes,*** ya que es lo único

que nos importa del cliente en este sistema. Al verlos podemos copiarnos los mails que necesitamos de ellos para proceder a enviar las promociones de la empresa.
Por otra parte, los archivos ***json*** son gestionados para guardar o leer los listados que contienen la información de los equipos que la empresa tiene en stock y las reservas de los clientes que se encuentran activas.

### *Algunas aclaraciones:*

La modificación de reservas reutiliza el formulario que da de alta las reservas. se envían todos los datos que contiene esa reserva al formulario para poder realizar las modificaciones pertinentes.
La baja de reservas elimina esa reserva, devolviendo el listado de equipos reservados al listado de equipos disponibles, ya que vuelven a poder ser elegibles. Estos vuelven acomodándose al final del listado.
Los mails de los clientes de momento se encuentran repetidos y sin filtrar. La idea es filtrarlos para que los muestre una única vez, a pesar de que el cliente haya venido varias veces a alquilarle a la empresa. Estos clientes pueden darse de alta muchas veces, ya que cada vez tendrán un id de reserva distinto, que me permitirá asociar a las reservas y saber qué es lo que el cliente alquiló en cada ocasión.

## *PARTE 2: TP 4*

### *SQL y CONEXIÓN A BASES DE DATOS*

Realicé el cambio de cómo se gestionan los datos de los clientes. Estos ya no sólo puedo obtenerlos desde un archivo ***xml*** sino que los traeré también desde la ***base de datos sql enchulame_db***.
Esta gestión de datos con la base se produce en la clase ***GestorSqlCliente***.
Por otra parte, el alta de cliente, también estará estableciendo una conexión a la base y agregando los datos correspondientes tanto en la ***base de datos sql*** como en el archivo ***xml***.
Al traerlos al formulario de ***Mails Clientes***, podemos elegir desde qué fuente traeremos los emails del cliente; si lo traigo desde la ***base de datos sql*** serán incorporados a través de un método donde se realiza un consulta que agrupa los mails eliminando los duplicados. Por otra parte, si los traigo desde el archivo ***xml*** también se visualizarán los mails sin duplicados ya que utilizo el método ***Distinct*** del namespace ***System.Linq*** para poder filtrarlos. Previamente a esto son ordenados con el método sort que recibe, a través de una ***expresión lambda***, el criterio de ordenación.

**Recordemos que en el sistema los clientes pueden darse de alta muchas veces, ya que siempre contendrán un id de reserva distinto, por lo que los mails pueden estar repetidos muchas veces.*

### *DELEGADOS Y PROGRAMACIÓN MULTIHILO*

Declaré dos delegados que tendrán utilidades distintas.
En primer lugar, ***DelegadoPresupuestoTotal*** es declarado para poder realizar una ***tarea secundaria*** que consistirá en mostrar en tiempo real, el presupuesto total a medida que se eligen los equipos que el cliente alquilará. También podrá mostrar el total con el descuento correspondiente si es que se selecciona el checkbox ***Es Estudiante.***
A su vez, una tarea al utilizar el método ***Run*** pasa como primer parámetro a un delegado. En ese caso paso la referencia a un método: ***MostrarPresupuestoFinal.***
Internamente, este método, utiliza el delegado declarado. ***MostrarPresupuestoFinal*** será el encargado de modificar el label que mostrará el presupuesto. Este comprobará si la modificación se está queriendo modificar desde un hilo secundario, por lo que tendrá que utilizar el delegado para **contener la referencia a sí mismo**. Al realizar el ***BeginInvoke*** es donde la referencia será enviada.
Esta tarea efectivamente está siendo llevada a cabo desde un hilo secundario: cuando se elige un equipo, cuando se lo elimina o cuando se selecciona el checkbox ya mencionado. Todos los hilos secundarios que estén corriendo serán cancelados al momento de confirmar la reserva.
Por otra parte, ***DelegadoConfirmaModificacion*** es declarado para poder enviar un ***evento***.

### *EVENTOS*

El evento ***OnConfirmaModificacion*** se enviará cuando al querer modificar una reserva, esta termine siendo confirmada o cancelada. Al confirmar la modificación, el evento se lanzará con un booleano en ***true*** para avisar que la modificación fue confirmada, caso contrario, cuando se cancele la modificación, se enviará con un booleano en ***false***.
Quien se suscribe a este evento sera el metodo ***EliminarReserva*** quien era el encargado de eliminar la reserva elegida.
Al analizar el booleano enviado, determinará si debe proseguir o no a esa eliminación.

## *MÉTODOS DE EXTENSIÓN*

Agregue un método de extensión llamado ***ContarCantidadReservas*** que cuenta la cantidad de reservas activas. Esta es usada en el formulario de reservas y se visualiza a través de un label.
Por otra parte agregue métodos de extensión que extienden la clase **String**. Estos comprobaran que el ingreso de los campos de clientes sean correctos

## *Aclaraciones:*

Modificaciones hechas de la entrega anterior:
La interfaz ***ICostoTotal*** implementa el método ***CalcularPresupuesto*** de manera estática de manera que el cálculo sea siempre el que corresponda sin la necesidad de declarar un objeto para realizar ese cálculo.
***ArchivoException*** ahora contiene un mensaje por defecto.
Agregue ***Test Unitarios*** para comprobar los ***métodos de extensión*** agregados en esta segunda entrega.

## *Conclusiones:*

El tp me parece un gran desafío que ayuda al estudiante a terminar de asimilar los temas dados. Desde mi propia implementación debo decir que me resultó difícil y al mismo tiempo sentí que aprendí muchísimo al llevarlo a cabo. La mayor dificultad la encontré a la hora de poder establecer un diseño que pudiera cumplir en un 100% con las consignas dadas.
Es de mi parecer que algunas cuestiones planteadas en este sistema se encuentran aplicadas de manera forzada y no de una manera orgánica. Por otra parte tampoco me siento conforme con los temas aplicados de esta segunda parte en este sistema, tal vez podría estar más desarrollado aún o podría estar planteado de alguna manera que resulte más razonable. También creo que al intentar sostener los temas del tp3 en la aplicación hace que se pierda el sentido en algunas cuestiones vinculadas al diseño del sistema.