

TP FINAL:

Empresa de Servicio

Segunda Parte

Grupo 12

Marcos Aguilera

Matias Castorina

Demian Nosedá

Introducción:

Se desea programar un sistema para gestionar contrataciones de servicio de Internet, así como también la gestión de abonados y facturación.

Se tiene dos tipos de internet, internet 500 e internet 100 y se puede contratar servicios para el celular, teléfono y cable.

Cada abonado puede contratar varios servicios iguales o diferentes, uno para cada uno de los domicilios que es titular.

No puede haber dos o más contrataciones con el mismo número de identificación ni tampoco el mismo domicilio.

Los abonados pueden ser de dos tipos: persona física o persona jurídica. Los medios de pagos pueden ser tres: efectivo, cheque o tarjeta.

En la segunda parte se tuvo como objetivo:

Agregar una interfaz gráfica conformada con varias ventanas para poder gestionar el sistema.

Poder registrar las siguientes acciones del abonado:

- pagar factura
- contratar nuevos servicios
- dar de baja servicios contratados

Incorporar un emulador de paso del tiempo (EPT), un botón que actualice el mes en curso.

Incorporar un gestor de facturación que realice la facturación correspondiente al mes actual de todos los abonados y agrega cada nueva factura a la lista de facturas de cada Abonado.

Emulación de “visita de AFIP”, obtiene la lista de facturas de la empresa, genera un reporte y muestra todas las facturas con su detalle. Si la visita de AFIP pretende irrumpir mientras se está dando de alta un nuevo abonado, se informa de esto con una ventana desplegable. Al terminar de dar el alta, se libera el recurso y AFIP podrá irrumpir.

El sistema debe persistirse.

Funcionalidades:

- Permite crear una empresa, asignarle un nombre y gestionar abonados con sus respectivos contratos.
- Puede generar un listado de todas las facturas emitidas con el detalle correspondiente de cada una, con su descripción y valor.
- Puede emular una visita de la AFIP la cual genera un reporte con la ganancia de la empresa.
- Permite solicitar un duplicado de cualquier objeto de tipo factura (clone()).
- Permite cambiar el servicio contratado, ya sea cambiar de internet, eliminar o agregar celulares, TVcable o teléfonos.
- Permite buscar un contrato.
- Permite eliminar un contrato.
- Permite emular el paso de los meses, emitiendo las facturas correspondientes a cada contrato activo.
- Permite al abonado pagar su factura.
- Permite persistir el sistema y cargar lo guardado al iniciarse.
- Permite visualizar los eventos que ocurren durante la ejecución.

Diseño del modelo:

La empresa tiene una relación de composición con las personas, las mismas se crean dentro de la empresa y no pueden existir fuera de ella, así también la empresa gestiona las personas, las crea, puede eliminarlas y buscar una en específico. Además la empresa puede si se solicita buscar un contrato en específico dentro de todas las personas y devolver una lista con todas las facturas de todos los abonados de la empresa.

La clase abstracta persona modela los atributos y comportamientos básicos que esperamos de las clases que heredan de la misma.

Atributos:

```
protected String nombre;  
protected int identificador;  
ArrayList<Contrato> contratos = new ArrayList<Contrato>();  
ArrayList<Factura> facturas = new ArrayList<Factura>();  
protected State estado = new SinContratacionesState(this);
```

Esta clase implementa Cloneable para obligar a todas las clases hijas a sobrescribir el método clone (), tiene un método abstracto getTasa que devuelve un double.

La persona tiene una relación de composición con los contratos y las facturas, poseen un estado encapsulados en la misma mediante el uso del patrón State. Cada persona puede gestionar sus contratos (Agregar, eliminar y modificar), utilizando para dichas acciones el comportamiento según el estado en que se encuentre la persona. También delega la gestión de sus facturas al contrato (Para generarla) y usa el patrón State para que se comporte de diversas maneras a la hora de pagarla.

En nuestro sistema existen dos clases que heredan de Persona: Persona Física y Persona Jurídica. Ambas clases implementan a su manera el getTasa mediante una delegación al Medio de Pago presente en el contrato (Pasado por parámetro). Por su parte la

persona jurídica no es clonable por lo que el método clone siempre lanza una excepción CloneNotSupportedException, y la persona física es siempre clonable.

La generación de la factura y cualquier modificación del contrato se delegan a la clase “Contrato”. Esta cuenta con los siguientes atributos:

```
private static int generadorIdContrato = 1;  
private int idContrato;  
private Domicilio domicilio;  
private PaqueteServicios paqueteServicios;  
private MedioPago medioPago;
```

El id del contrato es autogenerado en la creación del mismo.

Guarda el domicilio asociado a la contratación del servicio, el paquete con todos los servicios contratados y el medio de pago.

Las modificaciones sobre el paquete de servicios (contratación o eliminación de celulares, líneas de cable, teléfonos o cambio de internet) son delegadas del contrato al paquete de servicios.

La factura depende del contrato dado que este mismo es el encargado de su creación. Dicha factura puede ser clonada y contiene un método que devuelve un String con todos los detalles de la misma. Antes esta factura tenía una referencia a la persona, la cual fue removida dado a la hora de clonar generaba una doble referencia porque la nueva implementación las personas son quienes contienen las Facturas.

El paquete de servicios en principio lo implementamos mediante un patrón Decorator pero a la hora de recibir modificaciones y búsquedas nos generó problemas con los objetos encapsulados para retornarlos, además de este problema encontramos que no era un buen caso para implementar dicho patrón dado que agregar un servicio no implica decorar a los servicios ya existentes. En

lugar de este patrón optamos por implementarla mediante la siguiente composición:

```
Internet internet;  
ArrayList<Celular> celulares = new ArrayList<Celular> ();  
ArrayList<Telefono> telefonos = new ArrayList<Telefono> ();  
ArrayList<Cable> cables = new ArrayList<Cable> ();
```

El internet en dicho paquete es de carácter obligatorio.

Este paquete de servicios implementa búsqueda y agregación y eliminación para todos los servicios complementarios que lo requieren. También es capaz de devolver el costo de la suma de sus servicios, así como una lista con todos los detalles de los mismos.

Todos los servicios implementan una interfaz servicio, la cual los obliga a implementar un método que retorna el detalle y otro que retorna el costo.

El servicio internet es una clase abstracta de la que heredan todos los tipos de internet de la empresa, en nuestro caso internet500 e internet100, estas clases hacían uso de un patrón Singleton, lo quitamos debido a que todo el paquete de servicios debe ser clonable, además que si solo existe una sola instancia de internet y la misma cambia de precio con el tiempo, también cambiaría el precio en las facturas viejas.

Los servicios de Celular y Teléfono (línea fija) cuentan con un generador de números automático de forma que no se repitan y un getter de dicho número, encontramos un problema a la hora de persistir dado que el generador de números utiliza un parámetro static el cual no se puede persistir por lo que se reinicia en cada entrada del sistema. Por su parte el servicio Cable solo implementa los métodos obligatorios por el uso de la iServicio y no tiene ningún agregado.

La clase abstracta MedioPago obliga a las clases heredadas a implementar un método que retorne la tasa física y otro que retorne la tasa jurídica. Tarjeta, Cheque y Efectivo son clases extendidas de MedioPago implementan los métodos para retornar los dos tipos de tasa correspondientes.

La Clase AFIP contiene una referencia a la empresa que puede visitar y una lista de facturas que recibirá de la misma, tiene una relación de agregación con dicha empresa, puede generar un reporte y un listado de factura del momento en que se instanció la misma.

Existe un EMP (Emulador de paso del tiempo) que simula el paso de los meses, es observable para poder ejecutar la facturación automática de la empresa así como actualizar el estado de las personas en la misma. Esta clase inicialmente iba a implementar singleton pero, dado que la persistencia necesaria del mismo es una forma de clonación y un objeto que implementa singleton no puede clonarse, al intentar reiniciar el sistema, los meses volvían a empezar, por esta razón no se implementó dicho patrón.

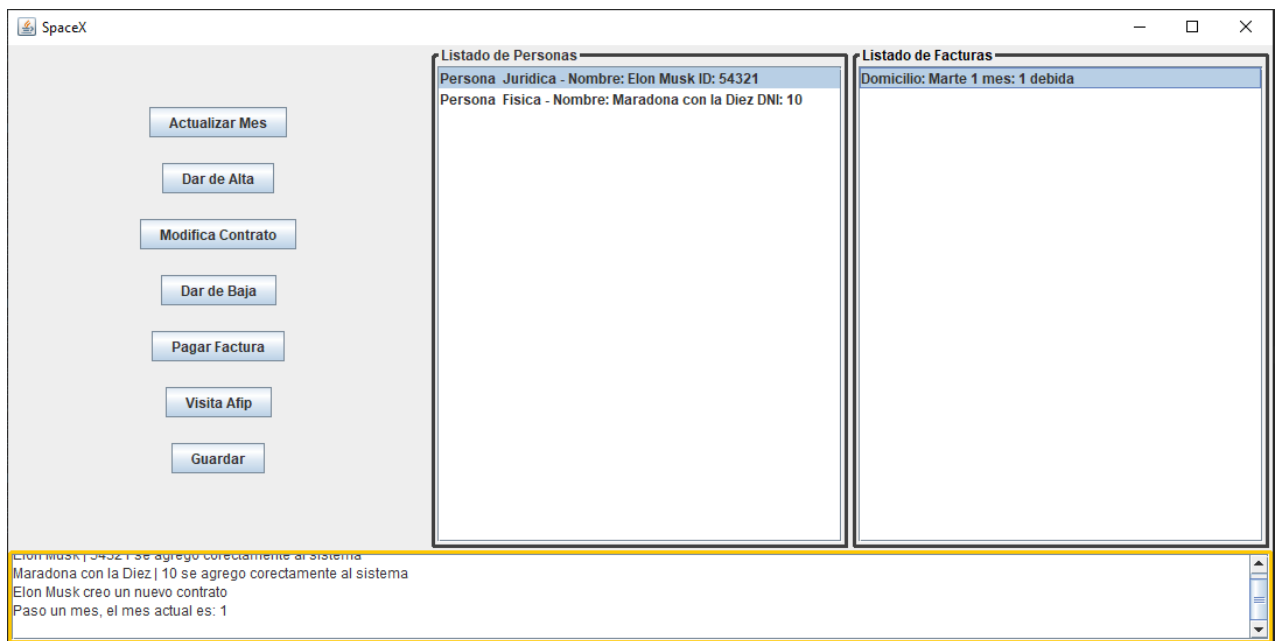
La clase Observador implementa la interface observer y contiene un emulador del paso del tiempo que será el objeto observado por la misma, además de una empresa de la que ejecutara la facturación y actualización de estados cuando haya cambios en el emulador del paso del tiempo.

Todo este modelo puede persistirse utilizando serialización binaria, además varios métodos del sistema pueden lanzar excepciones de las cuales algunas son controladas en el mismo entorno del modelo y el resto se delega a quien lo utilice.

Diseño de interfaz gráfica:

Se diseñó la interfaz gráfica utilizando el patrón MVC, por lo que se intenta mantener la vista independiente del modelo así como al modelo de la vista, gracias al uso del controlador como intermediario.

Vista principal:



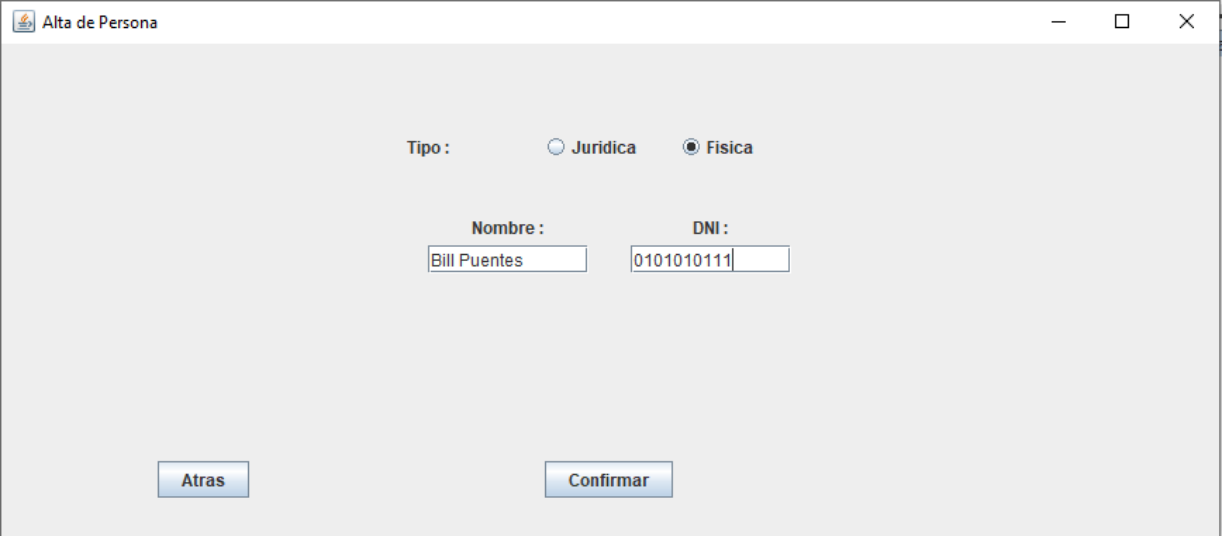
El controlador de esta ventana posee una referencia a la empresa, a la misma ventana principal y una referencia a un recurso compartido utilizado para generar bloqueo entre realizar altas y la visita de la AFIP.

Mediante el método Action Performed de la interfaz Action Listener controla la pulsación de los botones de la ventana, realizando dependiendo de cada uno la operación correspondiente mediante el uso del modelo, también se controla las excepciones que el mismo modelo lanza.

Esta ventana se encarga de lanzar las demás ventanas posibles del sistema (Altas, modificaciones y AFIP), por su parte las altas y AFIP deben bloquearse entre sí, para iniciarlas se utiliza el recurso compartido que posee métodos sincronizados que evitan que AFIP y Dar Alta se ejecuten a la vez.

Al cerrarse la ventana persiste el sistema automáticamente antes de la finalización de la ejecución.

Alta de Persona:



The screenshot shows a window titled "Alta de Persona" with a standard Windows title bar (minimize, maximize, close buttons). The window contains a form with the following elements:

- Tipo :** Two radio buttons are present: "Juridica" (unselected) and "Fisica" (selected).
- Nombre :** A text input field containing the text "Bill Puentes".
- DNI :** A text input field containing the text "0101010111".
- Buttons:** At the bottom of the window, there are two buttons: "Atras" (Back) on the left and "Confirmar" (Confirm) on the right.

Se encarga de llenar con datos un formulario para dar de alta una nueva persona en la empresa.

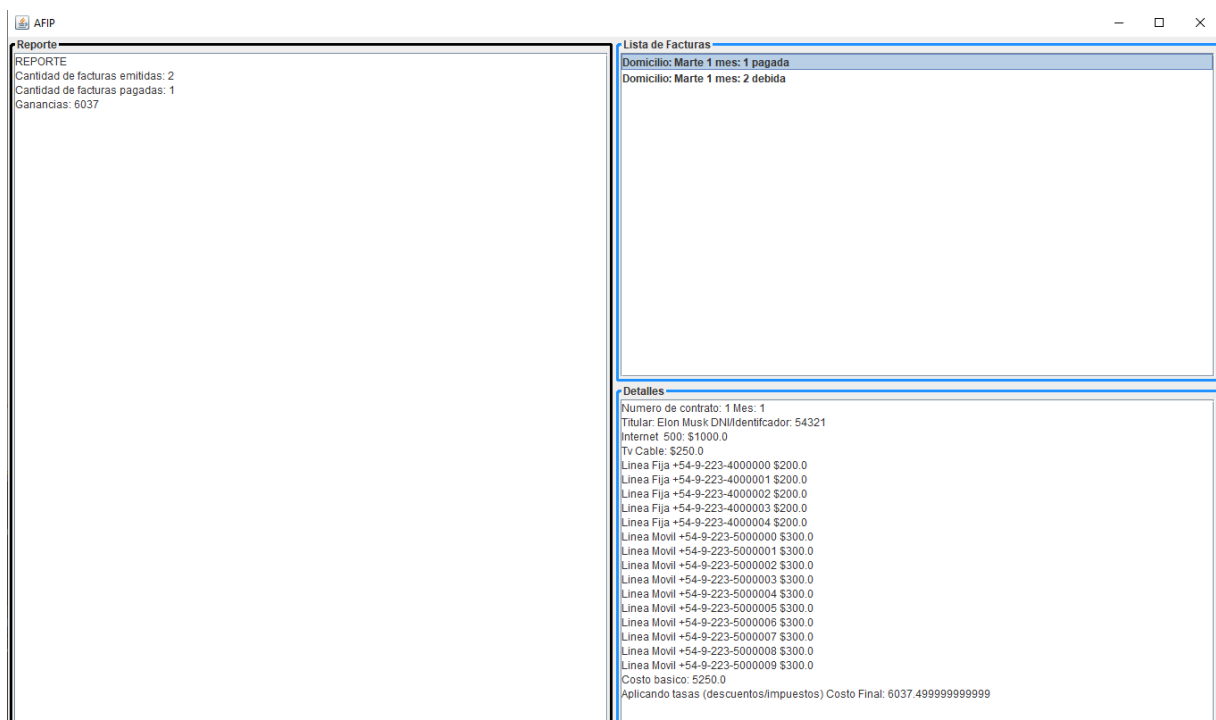
El controlador de Alta de Persona es instanciado a través del recurso compartido como dijimos anteriormente para generar un bloqueo con la AFIP.

Este controlador posee una referencia a la empresa pasada al constructor, al recurso compartido y al controlador de la ventana principal así como a su propia ventana.

El Action Performed de esta ventana controla la pulsación del botón confirmar y al presionarse ejecuta el método de la empresa para agregar una persona.

Al cerrarse la ventana ya sea por confirmar o al cerrarla naturalmente se comunica al recurso compartido que dicha alta ha terminado para liberarlo de ser necesario.

Visita AFIP:



Se encarga de mostrar el reporte de ganancias de la empresa, tiene una lista con las facturas y muestra el detalle de la seleccionada.

El controlador de la interfaz gráfica AFIP, posee una referencia a la ventana que controla, así como una referencia al controlador principal, a un objeto AFIP y al recurso compartido necesario para bloquear el alta al estarse ejecutando la ventana AFIP.

Al cambiar la selección en la lista de facturas el controlador comunica a la ventana que cambie el detalle mostrado, utilizando al modelo para obtenerlo dentro del controlador. Al cerrarse se

comunica al recurso compartido que dicha AFIP ha terminado para liberarlo de ser necesario.

Recurso Compartido:

Es una clase con métodos sincronizados que como se dijo anteriormente controla que la ejecución de Dar de Alta y la visita de la AFIP no se hagan en simultaneo, para esto posee un ArrayList con los controladores de cada ventana de Dar de Alta y un Boolean indicando si existe una visita de AFIP en ejecución (Por ser una a la vez). Para esto se implementaron dos métodos de ejecución uno de altas y otro de AFIP, además dos métodos de finalización respectivamente.

El método de ejecución de Altas carga un alta en el ArrayList si se pudo ejecutar y si no es posible pone al Thread que la llamo a esperar. El método que finaliza alta saca del ArrayList el controlador de altas que finalizo.

El método de ejecución de AFIP controla si ya existe una AFIP en ejecución (de ser así no hace nada), si no existe cambia el estado del Boolean a true para no permitir otra ejecución de AFIP en simultaneo.

A la vez si el ArrayList de altas esta vacío (no hay ninguna en ejecución) crea un controlador de AFIP que iniciara la ventana, de lo contrario pone a la misma a esperar hasta que terminen todas las altas y dicho ArrayList se vacíe.

Modifica Contrato:

Elon Musk | 54321

Lista de Contratos

ID	Domicilio
1	Marte 1

Domicilio: Marte 1

Cantidad de Cables: 1

Eliminar Celular Eliminar Telefono

Eliminar Contrato Eliminar Cable

Cantidad:

Lista de Celulares

- Celular [numero=5000000]
- Celular [numero=5000001]
- Celular [numero=5000002]
- Celular [numero=5000003]
- Celular [numero=5000004]
- Celular [numero=5000005]
- Celular [numero=5000006]
- Celular [numero=5000007]
- Celular [numero=5000008]
- Celular [numero=5000009]

Lista de Telefonos

- Telefono [numero=4000000]
- Telefono [numero=4000001]
- Telefono [numero=4000002]
- Telefono [numero=4000003]
- Telefono [numero=4000004]

Calle: Numero:

Medio de pago: ☒ Cheque ☐ Efectivo ☐ Tarjeta

Internet: ☐ 100 ☒ 500

Celulares: Cables:

Telefonos:

Atras Crear Contrato Modificar Agregar

Esta ventana se encarga de la gestión de los contratos (Crear, eliminar y modificar servicios) de una persona, muestra todos los contratos y servicios contratados de la misma, también valida los datos ingresados en ella habilitando y deshabilitando los botones de acuerdo a los campos requeridos.

El controlador de la ventana tiene una referencia a la persona de la cual gestionara los contratos, además de las referencias a la ventana controlada y al controlador principal.

El método Action Performed controla cada botón y realiza las operaciones necesarias a su función. Además dentro del mismo se controlan las excepciones que el modelo puede lanzar al gestionar contratos.

Los métodos de este controlador actualiza medio de pago y actualiza internet dado que es la única forma que encontramos de conocer el tipo de internet o el medio de pago sin romper el patrón MVC, o crear un método en el estado que devuelva de cual se trata.