Sistema de Contratación de Servicios de Seguridad

Programación III - 2023

Leonardo Motylicki - Agustin Perea - Tomas Larsen - Dario Funes

30/04/2023

Facultad de Ingeniería - UNMDP

INTRODUCCIÓN

El siguiente proyecto tiene como finalidad, modelar y programar un sistema de contratación de servicios de seguridad con las siguientes características:

1. Los servicios ofrecidos son: Monitoreo de alarmas en viviendas o Comercios.
2. Se cuenta con servicios adicionales: Cámaras de seguridad, botones antipánico y móvil de acompañamiento. Estos mismos se añaden al servicio base, por una diferencia en la suma de dinero
3. Cada abonado puede contratar varios servicios, uno para cada uno de los domicilios de los que sea titular.
4. La empresa puede aplicar dos tipos de promociones a sus servicios: Dorada o Platino
5. Los abonados pueden ser personas fisicas o juridicas
6. El pago de los servicios puede ser en: Efectivo, cheque o tarjeta.

El sistema debe poder realizar las siguientes acciones:

- Gestión de todos los datos del mundo del problema.
- El sistema debe poder generar un reporte de todos la facturas emitidas con el detalle correspondiente de cada una. O sea, para cada factura se debe listar cada una de sus contrataciones, con su descripción y valor.
- Contener una interfaz gráfica para poder gestionar el sistema. Deberá además tener incorporada dentro de dicha interfaz gráfica una zona de respuesta a la acción solicitada. Es decir, el resultado o cumplimiento de la acción.
- Se podrán registrar las acciones de pagar factura, contratar nuevos servicios y dar de baja servicios contratados.
- Poseer un botón que actualice el mes en curso para realizar la facturación.
- Contar con un gestor de facturación que realice la facturación correspondiente al mes actual de todos los abonados y agregue cada nueva factura a la lista de facturas de cada Abonado.
- Tener un apartado para dar de alta a técnicos de la empresa, puede haber varios técnicos. Los usuarios podrán solicitar la visita de un técnico. En la interfaz gráfica se deberá representar mediante texto la dinámica de la situación.
- Al finalizar la jornada se debe persistir.
- Al iniciar la jornada se debe despersistir.
- Se debe poder acceder a un histórico de las facturas de cada abonado.

IMPLEMENTACIÓN

Paquete datos

Con el objeto de lograr la persistencia del sistema se crea el paquete datos. Siguiendo la arquitectura de capas, esta corresponde a la capa homónima.

En este caso para aplicar el concepto de persistencia y utilizar el patrón DTO se crearon diferentes clases que permitirán guardar la información en formato XML. El archivo creado contará con cierta legibilidad a la hora de interpretar la información allí contenida.

Como los datos que deberían conservarse mediante persistencia son aquellos de sistema, el cual está en una clase sujeta a patrón Singleton que no cumplimenta con los requerimientos necesarios (constructor público vacío) y además con el fin de utilizar el concepto de serialización es que se procedió a la creación de las clases SistemaSeguridadDTO y UtilSerializacionSistema. La primera cuenta con lo requerido -un constructor vacío y todos los getters y setters- para aplicar el patrón DTO. La segunda convierte los atributos correspondientes a la lista de abonados y servicio técnico de la clase SistemaSeguridad para su posterior persistencia y despersistencia conforme a lo requerido.

La clase PersistenciaXML

Contiene como atributos las clases XMLEncoder y XMLDecoder que son utilizadas para escribir y leer documentos XML codificar y adquirir datos almacenados en flujo de bytes con sus respectivas clases FileOutputStream y FileInputStream.

En esta clase se desglosan los métodos correspondientes para lograr la apertura y cierre de los archivos para escritura o lectura.

Paquete negocio

Como su nombre lo indica, el paquete negocio corresponde a la capa de negocios o aplicación. En ella se encontrarán las clases necesarias para la gestión del sistema, con todos los métodos y relaciones que las consignas lo han requerido.

Las clases de servicios adicionales modeladas por herencia

Dentro de las clases están las que representan los servicios y otras los servicios adicionales que ofrece la empresa a sus abonados. A estas últimas pertenecen

BotonAntiPanico, Camara y MovilDeAcompaniamiento.

Todas son clases hija de ServicioAdicional y heredan atributos y métodos básicos que serán sobreescritos conforme a las características individuales de cada servicio. Por ejemplo, La clase MovilDeAcompaniamiento tiene la particularidad de que cuenta con un horario de entrada y otro de salida, en los que el móvil acompaña al cliente.

Todas estas clases tienen en sus respectivos métodos asertos que mostrarán por pantalla cuando se haya ingresado un dato no válido.

Las clases de formas de pago, patrón decorator y patrón factory

Hay clases que representan las formas de pago del cliente que ha de registrarse en el sistema al calcular la factura y registrar su debido pago. Mantienen una relación de herencia con MedioDePagoDecorator y también cuentan con asertos para advertir cuando no se cumple con la precondición de que la factura no sea null.

En el sistema de gestión programado, la facturación se realiza mediante métodos declarados en la interfaz Factura y desarrollados en las clases que la implementan.

Las facturas son clonables excepto cuando se trate de una persona jurídica. En estos casos se lanza una excepción que será tratada en la clase PersonaJuridicaCloneException.

Las clases relacionadas con este apartado son:

Interfaz IFactura:

Interfaz encargada de modelar el comportamiento de la clase Factura y su respectivo decorador.

Clase Factura:

Se extiende de IFactura. Establece, además, una relación de composición con Persona, siendo que este objeto en sus atributos tiene un arraylist de facturas. Es una clase clonable por lo que también implementa Cloneable.

Como su nombre lo indica, representa una factura correspondiente a un cierto abonado en función de los servicios contratados.

En cuanto a métodos, además de su constructor y getters, contiene el método para crear y asignar un pago a sí misma, siendo estas unas de sus responsabilidades. Además permite agregar o eliminar contrataciones. La creación de la factura dependerá de la

forma de pago, por lo cual se utiliza el decorator.

Clase abstracta Contratacion:

Modela mediante herencia el comportamiento de una Contratación. Es una clase clonable, esto se da como consecuencia de la clonación en cascada de Factura.

Es de tipo autoincremental, controlado por el campo lastId. Además cuenta con un campo Domicilio, que también es único para cada objeto de tipo Contratación.

Cuenta con métodos para calcular el valor de sus servicios, inclusive cuando tiene servicios adicionales y cuando se le aplica una promoción a la misma.

Clase Efectivo:

Clase hija de la clase abstracta MedioDePagoDecorator. En su constructor, establece el atributo descuento en 0,2 que es el coeficiente a multiplicar cuando, al pagar factura, se elija esta forma de pago.

Clase Cheque:

Clase hija de la clase abstracta MedioDePagoDecorator. En su constructor, establece el atributo descuento -0,1. Esto significa que no es un descuento, sino un recargo que se le hace al importe al elegir esta forma de pago.

Clase Tarjeta:

Clase hija de la clase abstracta MedioDePagoDecorator. En su constructor, establece el atributo descuento -0,05. Esto significa que no es un descuento, sino un recargo que se le hace al importe al elegir esta forma de pago.

Clase abstracta MedioDePagoDecorator:

Clase abstracta para poder implementar métodos del patrón Decorator que modelará de manera dinámica la factura en función de su forma de pago. Esto es porque, según la consigna, si el pago es en efectivo se realiza un descuento del 20%; si el pago es con cheque se realiza un incremento del 10%; y si el pago es con tarjeta de crédito se incrementa un 5%.

Clase MedioDePagoFactory:

Clase que implementa el Patrón Factory para un Medio de Pago.

El método getMedioDePago, recibe como parámetro un string que describe el nombre de algún medio de pago, y retorna un objeto de esa Clase.

Las clases de promociones y double dispatch

Con el objeto de ordenar de manera dinámica el sistema de gestión, la manera en que se calculan las promociones llamadas “dorada” y “platino” que se comportan de manera diferente según se trate de una contratación de tipo comercio o vivienda, se ha implementado un patrón de double dispatch. Para ello se contó con una interfase llamada Promocion. La otra parte del double dispatch se encuentra en los métodos de las clases concretas MonitoreoComercio y MonitoreoVivienda siendo estas hijas de la clase abstracta Contratación, cuyo análisis se realizará en otras líneas de este informe, más no en este apartado.

Las promociones no se pueden aplicar si en el double dispatch se recibe un parámetro null. Es por ello que se utilizaron asertos para advertir si ocurre esta situación.

Todas estas clases son de tipo clonable, ya que guardan una relación de composición con la clase Factura que puede clonarse excepto cuando corresponda a una persona de tipo jurídica.

Clase MonitoreoComercio:

Clase hija de Contratacion, para el caso particular en que la contratación es para un comercio.

Al llamar al método calculaPromo, por Double Dispatch, le está pidiendo a la subclase de Promoción (Dorada o Platino) correspondiente que utilice su método PromoComercio para calcular el valor del descuento correspondiente para la promo en particular en el monitoreo de comercios.

Clase MonitoreoVivienda:

Clase hija de Contratacion, para el caso particular en que la contratación es para una vivienda.

Al llamar al método calculaPromo, por Double Dispatch, le está pidiendo a la subclase de Promoción (Dorada o Platino) correspondiente que utilice su método PromoVivienda para calcular el valor del descuento correspondiente para la promo en

particular en el monitoreo de comercios.

Interface Promocion:

Interfaz encargada de modelar el comportamiento que debe tener una promoción. Los métodos que define se usan para implementar el Double Dispatch.

Clase Dorada:

Implementa la interfaz Promocion. Dependiendo del tipo de contratación, la promoción devuelve el importe, al cual se le aplica un descuento.

Clase Platino:

Implementa la interfaz Promocion. Dependiendo del tipo de contratación, la promoción devuelve el importe, al cual se le aplica un descuento.

Concurrencia y observer

La empresa de monitoreo ofrece un servicio técnico que deberá acudir al domicilio una vez solicitado. Para esta parte del programa se utilizaron threads ya que el técnico sale del arraylist de Tecnico.

Clase Tecnico:

Representa al objeto encargado de acudir a un domicilio y hacer la reparación en un tiempo determinado y cuyo avance será reflejado por pantalla.

Clase ServicioTecnico:

Cuenta con un arraylist de Tecnico y los métodos sinchronized para agregar un técnico y solicitar el servicio. Para esto último se utiliza un método que, si el arraylist está vacío (no hay ningún técnico disponible) espera. Cuando haya técnicos disponible devuelve un objeto Tecnico que hará las tareas correspondientes para los que fue llamado.

Clase ServicioTecnicoRunnable:

Implementa la interfase Runnable. Requiere de dos parámetros para su construcción: un ServicioTecnico que será instanciado desde sistema y un Observer que

será instanciado desde el MainControlador (perteneciente al paquete de la IU).

En el método run se solicita un ServicioTecnico (manda un mensaje a la clase ServicioTecnico para que utilice su método de solicitar servicio) y cuando esté disponible va comunicando mediante la interfaz gráfica los pasos que está realizando el servicio técnico con una espera impuesta en el sleep de 5000. Al finalizar vuelve a agregar al objeto Tecnico al arraylist y comunica el fin de la operación.

El patrón States

Conforme a los requerimientos del programa se utilizó este patrón para indicar los diferentes estados en los que puede caer un abonado siempre y cuando sea persona física y conforme a si tiene o no contrataciones adquiridas y si está o no al día. Para ello se utilizó una interfase llamada States que cuenta con los métodos pagarFactura, agregarContrato, darBajaServicio y actualizarEstado que serán implementados por las clases ConContratacionState, SinContratacionState y Moroso.

Los métodos harán lo que sus nombres sugieren desde la IU. Esta tendrá un botón para actualizar el estado si ya transcurrió el mes, lo cual en el caso de que la persona tiene servicios contratados y no abonó la factura correspondiente adquirir el estado de Moroso. Los métodos de dar de baja y agregar contrato permitirán al abonado tener el estado de con o sin contrataciones. Asimismo el método de pagar factura será el que evitará que caigan en estado moroso, pero también es el que permitirá poder salir de él, abonando la factura correspondiente con el debido recargo del 30%.

Todos estos métodos se comunican con la clase PersonaFisica ya que tiene un atributo State en el que irá algún objeto de tipo ConContratacionState, SinContratacionState o Moroso.

Estos estados determinarán si la persona puede contratar nuevos servicios, pagar factura y dar de baja servicios contratados conforme lo requerido en la consigna.

La clase del sistema general y sus componentes

La clase general del sistema se llama SistemaSeguridad. Tendrá como atributos objetos con los que guarda una relación de composición. Estos objetos son las clases Usuario, Persona y ServicioTecnico. Estos a su vez guardarán diferentes relaciones tal y como se observará en el diagrama UML que se encuentra en la última página de este informe.

Clase SistemaSeguridad:

Clase con una única instancia (patrón Singleton). Es la clase más importante y la que constituye el modelo del patrón MVC. Se encarga de:

- Agregar y eliminar clientes y usuarios de sus arraylist.
- Generar un reporte de todas las facturas emitidas de un cliente.
- Clonar facturas
- Pagar facturas (manda mensaje a un objeto Factura para que mediante sus métodos se calcule importes y descuentos.
- Dar de alta a técnicos y gestionar la solicitud de técnicos.
- Actualizar el mes para cambiar el estado de las personas y generar facturas.

Clase abstracta Persona:

Modela mediante herencia el comportamiento que tendrán los tipos de persona. Es de tipo clonable ya que por cascada, al clonar una factura se deberá también clonar diferentes clases más, entre ellas, Persona.

Asimismo tiene una relación de composición con Sistema siendo que Sistema tiene como atributo un ArrayList de esta clase. También tiene una composición con Domicilio, siendo que el Domicilio no existiría sin una Persona. Eliminar la Persona, elimina el Domicilio.

Persona representa a un abonado del sistema. Es una clase padre de los diferentes tipos de persona que pueden ser clientes del sistema (física o jurídica) ya que mantiene una relación de herencia con esas clases, modelando su comportamiento.

Los métodos que tiene son abstractos y serán sobreescritos en sus clases hija. En particular tiene un método que recibe un ArrayList de doubles llamado recibeDescuento. Este método recibe como parámetro otro ArrayList, pero de contrataciones. Su función es la de crear un Array que guarde en cada componente i un coeficiente. Este coeficiente representa cuánto se ha de cobrar por la contratación i según se trate o no de una contratación de persona jurídica n° 4 o más. Es una forma de resolver cómo aplicar el descuento que reciben las personas jurídicas a partir de su tercer contratación. Este Array tiene tantas componentes como contrataciones tiene la persona.

Clase PersonaJurídica:

Clase hija de Persona. Hereda los atributos de ella y sobreescribe sus métodos. No

es clonable, lo que representa una situación para la clonación de factura siento que PersonaJuridica es hija de Persona y Persona tiene una relación con Factura, una clase que es clonable.

El método para construir el ArrayList de recibeDescuento en este caso lo que hace es completar el array con 1s en los primeros 3 componentes y luego 0.5 del 3ero en adelante. Lo que significa que a partir del 4to servicio, el mismo saldrá solo la mitad de su valor. Al calcular el monto de la factura se deberá multiplicar el valor de cada contratación por el coeficiente guardado en este array.

Clase PersonaFísica:

Clase hija de Persona. Hereda los atributos de ella y sobreescribe sus métodos. Esta clase sí es clonable.

En este caso el método para construir el ArrayList de recibeDescuento, como no va a haber nunca un descuento para las personas físicas, no importa cuántas contrataciones tenga, el coeficiente siempre va a valer 1.

Clase Domicilio:

Clase concreta que representa el domicilio de un abonado. Mantiene una relación de composición con Persona y una agregación con Contratacion.

Sobreescribe los métodos de Object equals y hashCode ya que el sistema necesita ver si un domicilio es igual a otro porque de ser así, habría un error. Cada domicilio debe ser distinto y cada domicilio debe llevar una contratación determinada.

Clase Usuarios:

Un usuario es una persona de la empresa autorizada a utilizar el sistema de gestión. Cuenta con un nombre y una contraseña de ingreso que se ingresarán por teclado todo gestionado por la UI. Esta clase cuenta con su constructor, getters y setters.

Paquete presentación

Conforme a las especificaciones requeridas se crea la capa de presentación en el paquete del mismo nombre.

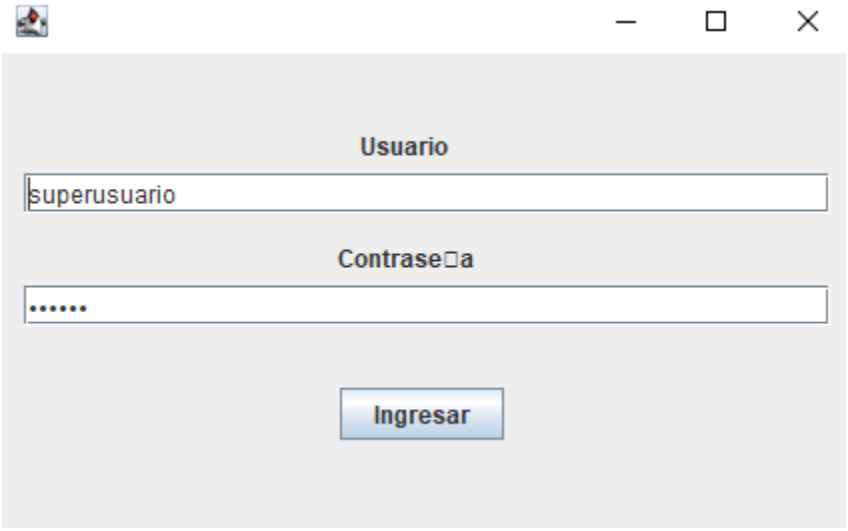
Los JFrame

Se utilizaron para la vista dos JFrame distintos. Se trata de las clases VistaLogin y

VistaSistema.

Clase VistaLogin:

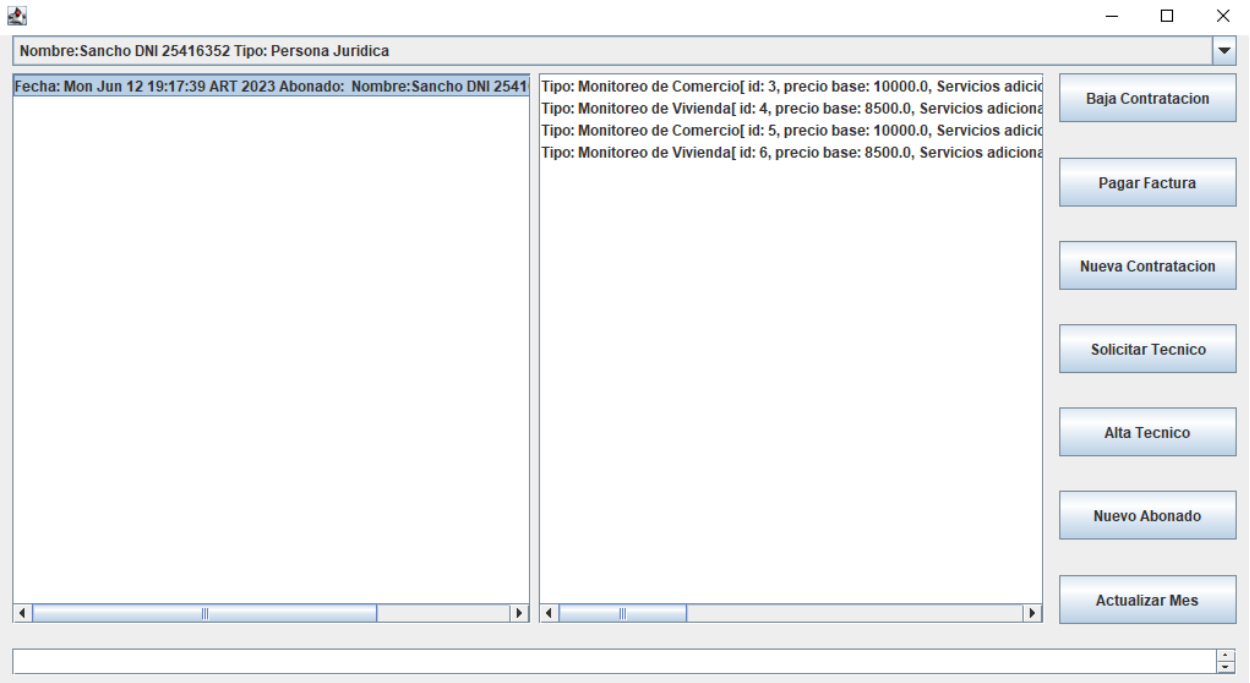
Esencialmente es el cuadro de login del usuario donde pedirá su nombre y contraseña para leerlos e ingresar al sistema. Cuenta con un campo de contraseña, uno de texto, un botón para ingresar y 3 etiquetas. Todas serán instanciadas desde esta clase.



Clase VistaSistema:

Es la ventana principal del sistema implementada con un MouseListener. Una vez ingresado el usuario y contraseña, se accede a ella con todas las funcionalidades. Cuenta con los paneles de zonaPrincipal, zonaBotones, zonaRespuesta, zonas de texto y scrolls.

Registra los eventos -clicks de mouse e inputs por teclado- y se comunica con las clases principales correspondientes de la capa de negocio.



Clase MainControlador:

Es la clase que contiene los action listeners y la que se comunica con el modelo para la realización de las diferentes acciones requeridas por el usuario. Estas se relacionan con los diferentes botones del panel: baja contratación; pagar factura; nueva contratación; solicitar técnico; alta técnico; nuevo abonado; y abonar mes.

Por supuesto que contempla las restricciones debidas y, por ejemplo, si se quiere realizar alguna acción limitada por el estado del abonado, estos botones no estarán disponibles para el usuario.

Los JDialogs

Con respecto a las ventanas secundarias JDialogs el sistema contará con 6 de ellas. Las mismas se utilizarán para las ventanas de Alta de un técnico, error, pagar factura, nueva contratación, nuevo abonado y el informe con el estado y facturas del abonado.

Paquete main

Contiene la clase Main. Es la que se ejecutará para dar inicio al programa.

La clase Main será la encargada de correr el programa, instanciando el sistema, el controlador de la interfaz visual y sus respectivas ventanas principales.

A modo de ejemplo se han instanciado abonados para corroborar la funcionalidad del sistema.

DIAGRAMA DE CLASES