Informe

Sistema de contratación de servicios – programa Java

Facultad de Ingeniería, U.N.M.d.P

Programación III – Año 2020

INTEGRANTES

- Andrade, Tobías Ezequiel
- Della Rocca, Federico
- Rabe, Gastón

Sistema de contratación de servicios – programa Java

Introducción

En el siguiente informe se detallará el funcionamiento del programa Java que resuelve el siguiente enunciado:

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

El sistema debe poder contemplar los siguientes servicios ofrecidos para cada contratación:

Internet100

Puede agregarse también la contratación de: Celular y/o Teléfono y/o TV_Cable

Internet500

Puede agregarse también la contratación de: Celular y/o Teléfono y/o TV_Cable

Al contratar cada servicio debe quedar en claro su descripción y su valor:

1. *Si el servicio es Internet100 tendrá un valor base de $850*
2. *Si el servicio es Internet500 tendrá un valor base de $1000*
3. *Por el agregado de cada celular se incrementa el valor en $300*
4. *Por el agregado de cada teléfono fijo se incrementa el valor en $200*
5. *Por el agregado de cada línea de TV-Cable se incrementa el valor en $250*

Cada abonado puede contratar varios servicios iguales o diferentes, uno para cada uno de los domicilios que es titular. Suponemos que un mismo cliente puede tener varios domicilios. Cada una de estas contrataciones se diferenciará de las otras por su número de identificación y Domicilio. Tener en cuenta que no puede haber dos o más contrataciones con el mismo número de identificación ni tampoco el mismo domicilio. Esto es independiente a que pertenezcan a la misma lista de contrataciones de una factura o no. Del abonado nos interesa conocer su nombre y DNI.

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

De acuerdo al medio de pago y al tipo de abonado se realizarán los siguientes descuentos o incrementos de la tarifa resultante de cada contratación:

1. *Si el abonado es persona física y paga en efectivo recibe un descuento del 20%*
2. *Si el abonado es persona física y paga con cheque recibe un incremento del 10%*
3. *Si el abonado es persona física y paga con tarjeta el valor no se altera.*
4. *Si el abonado es persona jurídica y paga en efectivo recibe un descuento del 10%*
5. *Si el abonado es persona jurídica y paga con cheque recibe un incremento del 15%*
6. *Si el abonado es persona jurídica y paga con tarjeta recibe un incremento del 20%.*

Para realizar la facturación se debe tener en cuenta el abonado y una lista de contrataciones, o sea, cada factura debe contener esa información, así como el cálculo del total con y sin descuento por medio de pago.

Funcionalidades requeridas

Sistema de contratación de servicios – programa Java

El sistema debe poder realizar la gestión de todos los datos del mundo del problema. El sistema debe poder generar un reporte de todas las 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.

El sistema debe permitir solicitar un duplicado de cualquier objeto de tipo factura (clone()) y en caso de no ser posible emitir un mensaje indicando la imposibilidad.

Nuevas funcionalidades:

- *Se deberá incorporar una interfaz gráfica (unificada o varias ventanas) para poder gestionar el sistema.*
- *Se podrán 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 agregue cada nueva factura a la lista de facturas de cada Abonado.*
- *Incorporar la gestión a través de la interfaz gráfica de las nuevas acciones del abonado (pagar, contratar, dar de baja).*
- *Incorporar dentro de la interfaz gráfica una zona de respuesta a la acción solicitada, por ejemplo si un abonado físico sin contrataciones (ver más abajo) pretende pagar una factura, deberá mostrarse el cartel "no puede pagar aún".*
- *Emulación de "visita de AFIP". Se debe generar un thread, que emule la visita de la AFIP a la empresa y obtenga un clone de todas las facturas y su estado. Se debe visualizar un reporte en otra ventana y hasta que dicha ventana no sea cerrada se debe bloquear (mediante un recurso compartido) la posibilidad de dar de alta otro abonado. Si la visita de afip pretende irrumpir mientras se está dando de alta un nuevo abonado, se debe informar de esto con una ventana desplegable. Al terminar de dar el alta, se libera el recurso y AFIP podrá irrumpir.*
- *Se debe persistir al sistema con su lista de abonados y así en cascada con todo.*
    - *Al finalizar la jornada se debe persistir.*
    - *Al iniciar la jornada se debe despersistir.*

Especificaciones del diseño

Considere que el abonado de tipo persona FÍSICA puede pertenecer a alguno de los siguientes estados:

1. *sin contratación*
2. *con contrataciones*
3. *moroso*

Al reclutar un nuevo abonado, comienza en el estado "sin contratación". Al hacer la primera contratación pasa al estado "con contrataciones".

Cada fin de mes, se emiten las facturas para cada abonado.

Si un abonado no ha pagado dos facturas consecutivas, pasará a estado "moroso".

Sistema de contratación de servicios – programa Java

En cada uno de los estados, el abonado tiene ciertas restricciones en cuanto a las siguientes acciones:

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

Estado \ acción	*pagar factura*	*contratar nuevo servicio*	*baja de un servicio*
sin contratación	*no puede*	*sí puede*	*no puede*
con contratación	*sí puede*	*sí puede*	*sí puede*
moroso	*recargo 30%*	*no puede*	*no puede*

Se le podrá solicitar al abonado Físico que realice todas las acciones, de acuerdo al estado en que esté será la acción que realice y la descripción de la misma.

Al activar el EPT avanza el tiempo en un mes, eso hace que se deba revisar el estado de cada abonado Físico para ver si debe cambiar de estado. Se activa el "actualizador de estado". Un criterio análogo se debe aplicar en todas las acciones del tiempo o provocadas por el abonado que determinan un cambio de estado.

Al activar el EPT también se debe activar el proceso de generación de facturas del mes en curso, se activa el "generador de facturas".

Tanto el "actualizador de estados" como el "generador de facturas" son Observers del EPT, del Abonado Físico.

Utilice el patrón de diseño State, Observer/Observable"

Desarrollo

El proyecto posee ocho paquetes, cada uno con sus respectivas clases y/o interfaces:

- controlador
  - ✓ Controlador
- estados
  - ✓ ConContratacionState
  - ✓ IState
  - ✓ MorosoState
  - ✓ SinContratacionState
  - ✓ State
- excepciones
  - ✓ ContratacionException
  - ✓ DomicilioEIdentificacionException
- modelo
  - ✓ Abonado
  - ✓ AbonadoFactory
  - ✓ AFIP

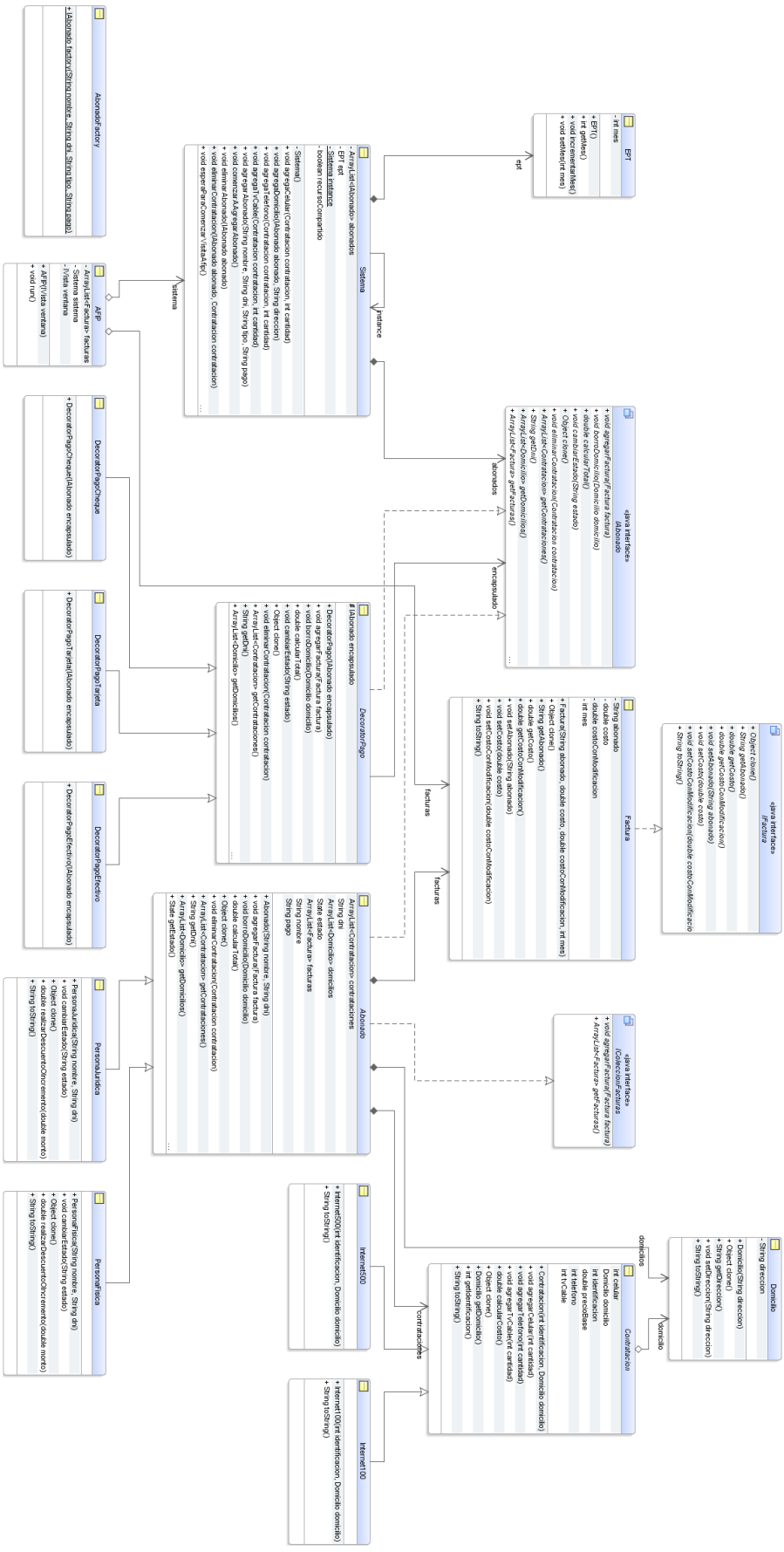
- ✓ Contratacion
- ✓ DecoratorPago
- ✓ DecoratorPagoCheque
- ✓ DecoratorPagoEfectivo
- ✓ DecoratorPagoTarjeta
- ✓ Domicilio
- ✓ EPT
- ✓ Factura
- ✓ IAbonado
- ✓ IColeccionFacturas
- ✓ IFactura
- ✓ Internet100
- ✓ Internet500
- ✓ PersonaFisica
- ✓ PersonaJuridica
- ✓ Sistema

- observadores
  - ✓ ActualizadorDeDatos
  - ✓ GestorFacturacion
- persistencia
  - ✓ IPersistencia
  - ✓ PersistenciaBIN
- util
  - ✓ Util
- vista
  - ✓ IVista
  - ✓ VentanaAfip
  - ✓ VentanaSistema

A continuación, se explicará brevemente la relación entre las clases y el funcionamiento del programa. Para ello, se recomienda ir leyendo y viendo el diagrama UML a la vez.

Paquete **modelo:**

Sistema de contratación de servicios – programa Java



La clase Sistema implementa el patrón Singleton ya que existe una sola instancia de dicha clase, y

su implementación evita las dobles referencias entre las clases de la aplicación. Ésta tiene como atributos

un ArrayList de IAbonados, un emulador de paso del tiempo (EPT), y una variable booleana que controla el recurso compartido. Los métodos de la clase Sistema contemplan todas las funcionalidades que debe tener el programa, esto se debe a que toda operación o movimiento que se desee realizar será a través del Sistema, y éste es el que se encarga de derivar las responsabilidades a las clases correspondientes para realizar la tarea que se requiera.

La clase Abonado es de tipo abstracta e implementa la interfaz IAbonado e IColeccionFacturas, y de ella extienden las clases PersonaFisica y PersonaJuridica. Éstas sobrescriben los métodos toString, clone, implementa el método realizarDescuentoOIncremento y tienen sus respectivos constructores. La razón por la que ambas clases extienden de Abonado es que las dos son tipos de éste, pero a cada una se le realiza descuentos o incrementos a su manera. Además, la clase Abonado tiene un atributo de tipo State que nos permite saber el estado de cada abonado, un ArrayList de facturas y un String con el método de pago.

Cada instancia de clase concreta que extienda de Abonado posee nombre, DNI, y luego un ArrayList de sus domicilios y otro ArrayList de sus contrataciones. Vale aclarar que puede haber domicilios sin contrataciones, por eso la importancia del ArrayList de domicilios, pues de no estar presente este atributo, los domicilios tendrían una relación de composición con las contrataciones, y la relación en este caso es de agregación.

Utilizamos el patrón Decorator para decorar el abonado con la forma de pago a llevar a cabo. Dicha forma, dependiendo del tipo de abonado que sea, puede obtener un descuento, un recargo o ninguna de las dos.

Para crear una instancia de la clase Abonado se utiliza el patrón factory en la clase AbonadoFatory, esto se implementó para evitar los errores a la hora de crear un abonado, ya que este tiene atributos esenciales para el funcionamiento del sistema. Por esta razón se pasan como parámetro al método static de la clase factory el nombre, el tipo y el método de pago, y el método se encarga de crear una instancia de Abonado del tipo que reciba por parámetro, setear los atributos correspondientes y devolver el objeto creado en su totalidad listo para entrar al sistema.

La clase Contratación es de tipo abstracta, de ella extienden las clases Internet100 e Internet500. Éstas sobrescriben los métodos toString y tienen sus constructores. Los constructores llaman a super y luego setean el precio base correspondiente a cada contratación dependiendo su tipo. Esto se puede hacer ya que las subclases tienen acceso a los atributos de la superclase, por ser estos declarados como default. Cada instancia de clase concreta que extienda de Contratación posee número de identificación, número (cantidad) de celulares, teléfonos fijos y de líneas de TV-Cable, también precio base de la contratación y domicilio. Para su creación se necesita el abonado el cual quiere realizar la contratación, un domicilio que: se debe verificar que pertenezca a dicho abonado y además revisar que si otro abonado tiene el mismo domicilio (es posible en esta aplicación), éste no debe tener una contratación, pues si la tuviera, ésta no podrá ser creada. Además, se necesita el tipo de contratación que se desea realizar y un número de identificación, el cual debe revisarse que ninguna contratación del sistema tenga el mismo.

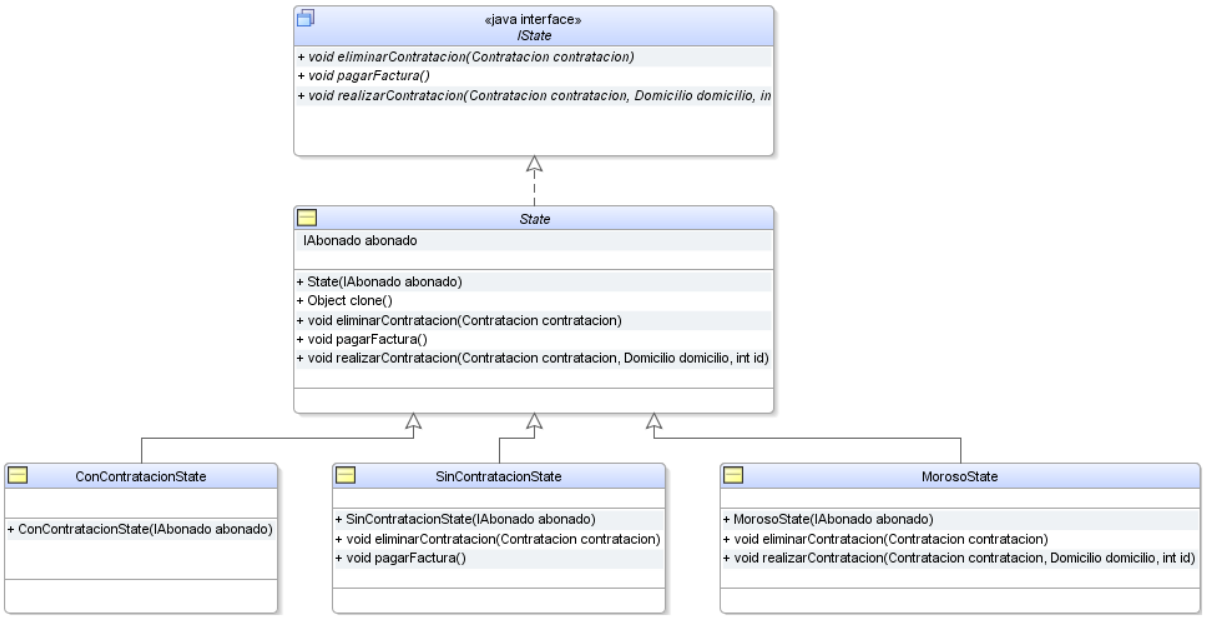
La clase Domicilio únicamente tiene el atributo Direccion. Es el Sistema quien se encarga de agregar un domicilio a un abonado, para esto se recurre al método agregaDomicilio y se le debe pasar como parámetro el nombre del abonado y la dirección del nuevo domicilio a agregar. Éste se encargará solo de crear la instancia Domicilio y de agregarla al ArrayList de domicilios en el abonado correspondiente.

La clase Factura posee como atributos el nombre del abonado al cual pertenece, el mes de su emisión, el costo inicial de la factura y por último el costo modificado. Esto depende del tipo de abonado al cual se le esté añadiendo la factura, este costo puede tener un descuento o un incremento en cuanto al costo inicial. Las facturas son emitidas por el "gestor de facturación" al terminar cada mes, es decir al activar el EPT.

Las clases de este paquete que implementan la interfaz Cloneable son Factura, Contratacion y Domicilio, ya que se debe poder pedir el duplicado de una factura.

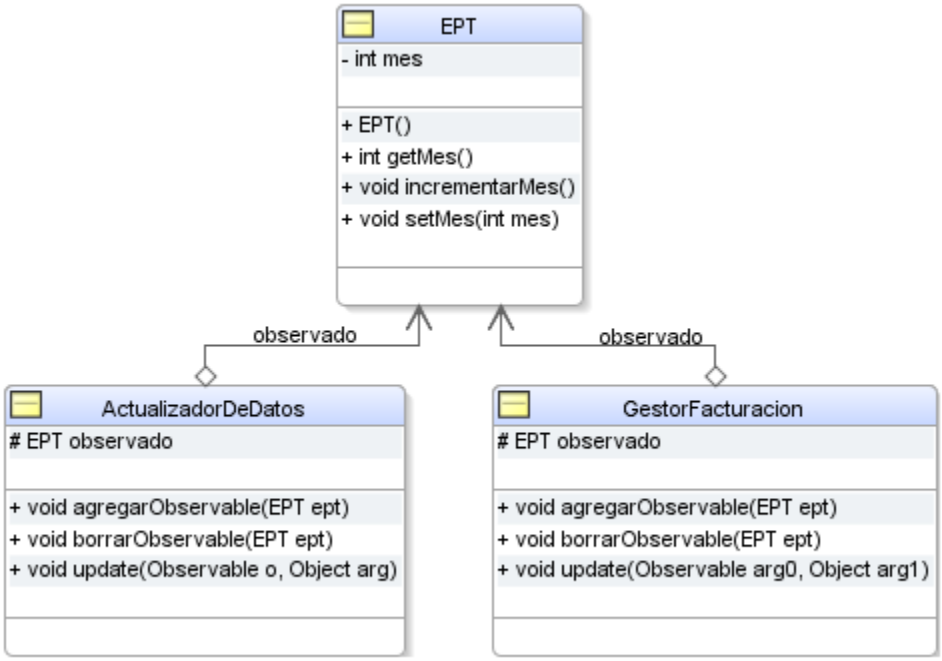
La clase AFIP se extiende de Thread y su objetivo es interrumpir el funcionamiento del sistema para mostrar las facturas no pagas. Posee un ArrayList de facturas con los “duplicados” de las facturas no pagas. Cuando entra en acción este thread, si el recurso compartido está libre lo toma y bloquea el botón de "Incorporar" para que no se pueda dar de alta un abonado mientras la visita está en curso. Si al intentar tomar protagonismo el recurso compartido está en uso (se está dando de alta un abonado), se abre una notificación de que la visita AFIP comenzara cuando se termine de dar de alta el abonado en curso.

Paquete **estados:**



En el paquete **estados** se lleva a cabo un patrón State, que se encarga de asignarle a cada abonado el estado en el que se encuentra; dichos estados pueden ser: con contratación, sin contratación y moroso. Dependiendo del estado de cada abonado, se pueden o no realizar ciertas acciones; es por ello que se utiliza este patrón.

Paquete **observadores:**



Las clases ActualizadorDeDatos y GestorFacturacion actúan como observadores (implementan Observer) y EPT, que pertenece al paquete **modelo**, se extiende de Observable. ActualizadorDeDatos actualiza los estados de cada abonado a medida que se incrementa el mes de EPT. GestorFacturacion genera las facturas del mes en curso, sólo para los abonados físicos.

Paquetes **vista y controlador:**

Podemos encontrar dos ventanas: VentanaAfip representa el funcionamiento de AFIP. Aparece cuando el thread AFIP toma el control del recurso compartido, mostrando únicamente en su ventana los datos de las facturas que faltan pagar. VentanaSistema representa el funcionamiento del sistema dando las opciones de agregar abonados o quitarlos, de agregar domicilio, agregar o quitar contrataciones, añadir servicios a dichas contrataciones, pagar facturas e incrementar mes.

Para el manejo de las ventanas aplicamos el patrón MVC, que consiste en manejar el modelo, la vista y el controlador por separado. El controlador interactúa con VentanaSistema mediante la interfaz IVista, de manera que, si se deseara modificar dicha ventana, no sería necesario modificar el controlador.

Paquete **persistencia**:

Se decidió persistir los datos mediante persistencia binaria ya que resultó más fácil implementar la interfaz Serializable que tener constructor vacío y getters y setter en cada clase a persistir (para llevar a cabo una codificación XML).

Conclusión:

Se puede concluir mediante este informe que se pudo resolver el problema planteado utilizando el lenguaje de programación orientado a objetos de Java, implementando distintos patrones y técnicas para el correcto funcionamiento del programa.