TRABAJO PRACTICO PROGRAMACIÓN III

BAILÓN GUIDO, BARRIGA NAHUEL, HERNANDEZ JULIETA, FIGURAS CAGGIANO GREGORIO

Introducción

Como indica la consigna, desarrollamos un sistema con la capacidad de gestionar contrataciones de servicios de seguridad y monitoreo, así como también la gestión de abonados y facturación.

Los servicios ofrecidos serán el monitoreo de alarmas en viviendas y monitoreo de alarmas en comercios, pudiendo agregar a cada uno el monitoreo de cámaras y/o botón antipánico y/o móvil de acompañamiento. Cada servicio extra tiene un precio especifico que se le sumara al precio base. La empresa cuenta con promociones a sus servicios, afectando el valor del servicio dependiendo de su tipo.

El abonado puede ser una persona física o persona jurídica. Puede contratar varios servicios iguales o diferentes para cada uno de los domicilios que es titular. Los medios de pago son efectivo, tarjeta o cheque.

Descripción de clases

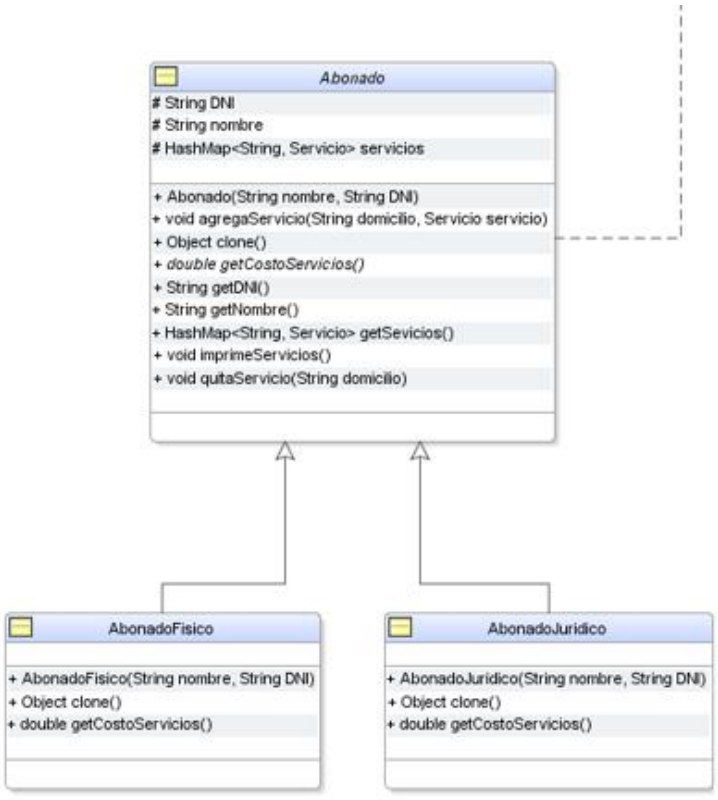
Clase AbonadoFactory:

Para poder ingresar un nuevo abonado al sistema, se creó la clase 'AbonadoFactory'. Esta clase retorna una instacia de la clase 'Abonado' dependiendo de los datos ingresados en la clase 'Prueba'.

Clases Abonado, AbonadoJuridico y AbonadoFisico:

La clase 'Abonado' tiene como hijas a las clases 'AbonadoJuridico' y 'AbonadoFisico'. La clase padre implementa la interfaz 'IAbonado' y se definen los atributos comunes para las clases hijas siendo 'nombre', 'DNI' y un haspMap de servicios ofrecidos a cada domicilio. Respecto a los métodos, se encuentran los getters de cada atributo, un método para agregar un servicio o quitarlo y un método abstracto para obtener el costo del servicio requerido. Además, en esta clase se implementa una función 'object clone' para realizar el duplicado de una factura si se requiere. La misma puede devolver un "CloneNotSupportedException" en el caso que se intente clonar una persona jurídica.

En cuanto a las clases hijas, estas se extienden de la clase 'Abonado'. No definen más métodos de los que ya heredan. Solo implementan el método 'getCostoServicio', pues este depende del tipo de abonado que sea, y el método "Object clone()" el cual en el caso de la clase "AbonadoFisico" retorna clonado, y en el caso de "AbonadoJuridico" retorna la excepción previamente mencionada con el cartel "Imposible clonar una persona jurídica".

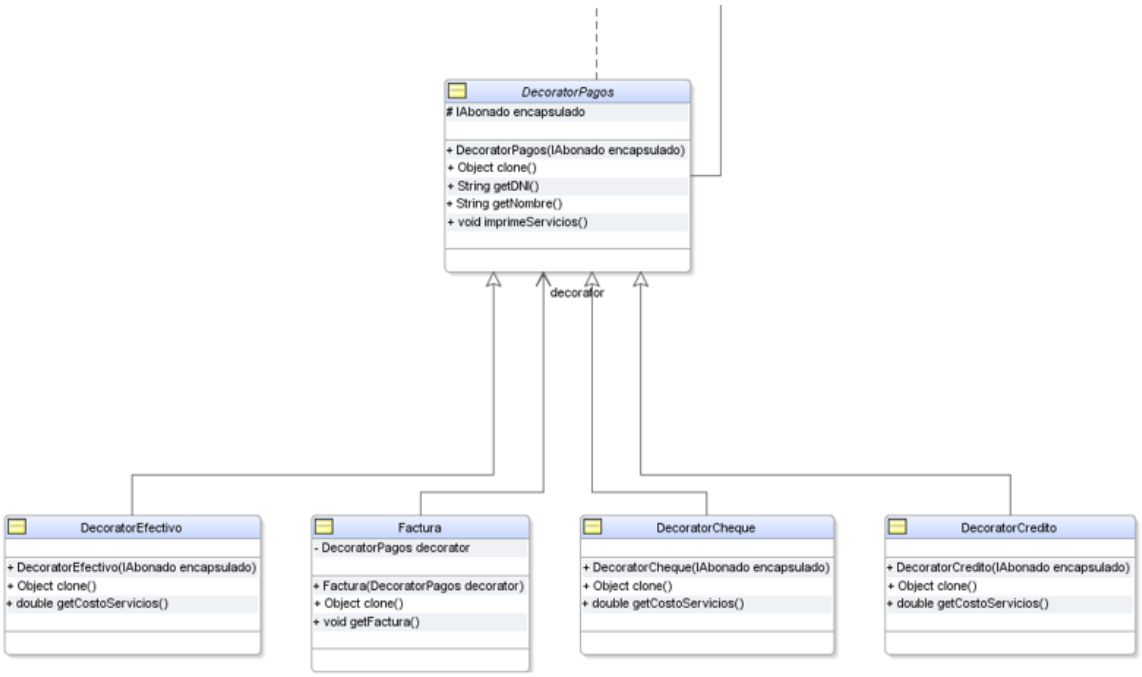


Clases DecoratorPagos, DecoratorEfectivo, DecoratorCredito, DecoratorCheque e IAbonados:

Siendo que dependiendo del tipo de pago corresponde un descuento o incremento especifico, se decidió utilizar el patron decorator para realizar la facturación del servicio para evitar el uso de clases excesivas.

La clase 'DecoratorPagos' hereda, al igual que los objetos que tiene que decorar ('DecoratorEfectivo', 'DecoratorCredito' y 'DecoratorCheque'), de la interfaz 'IAbonado'. En dicha interfaz se implementan las funciones "getCostoServicio", "getNombre", "getDNI", "imprimeServicios" y el 'Objetc clone()' las cuales son ejecutadas en un encapsulado proveniente de la clase "Abonado".

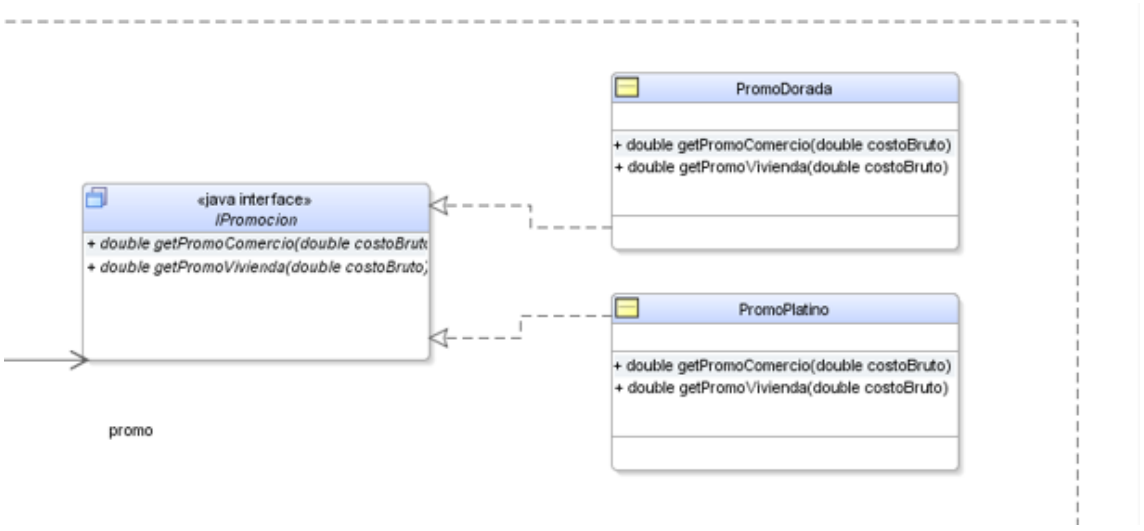
En las subclases 'DecoratorEfectivo', 'DecoratorCredito' y 'DecoratorCheque' se ejecuta el método 'Object clone()' el cual clona los datos necesarios para realizar la facturación del servicio en la clase 'Factura'. Y aplica el método double getCostoServicios que devuelve el costo final de los servicios, aplicándole promoción (si posee) y haciendo un recargo o un descuento dependiendo del tipo de método con el que el abonado abona los servicios.



Clases Ipromocion, PromocionDorada y PromocionPlatino:

Respecto a las promociones que ofrece la empresa, se creó una interfaz 'Ipromocion' en la cual de definen los métodos 'getPromoVivienda' y 'getPromoComercio'. Esta interfaz encapsula las clases 'PromoDorada' y 'PromoPlatino' quienes implementan dichas funciones que retornan el costo neto.

Debido a que dependiendo del tipo de servicio y al tipo de promocion seleccionado se aplica un descuento diferente, se utilizó Double Dispatch para devolver los 4 posibles valores de descuento.

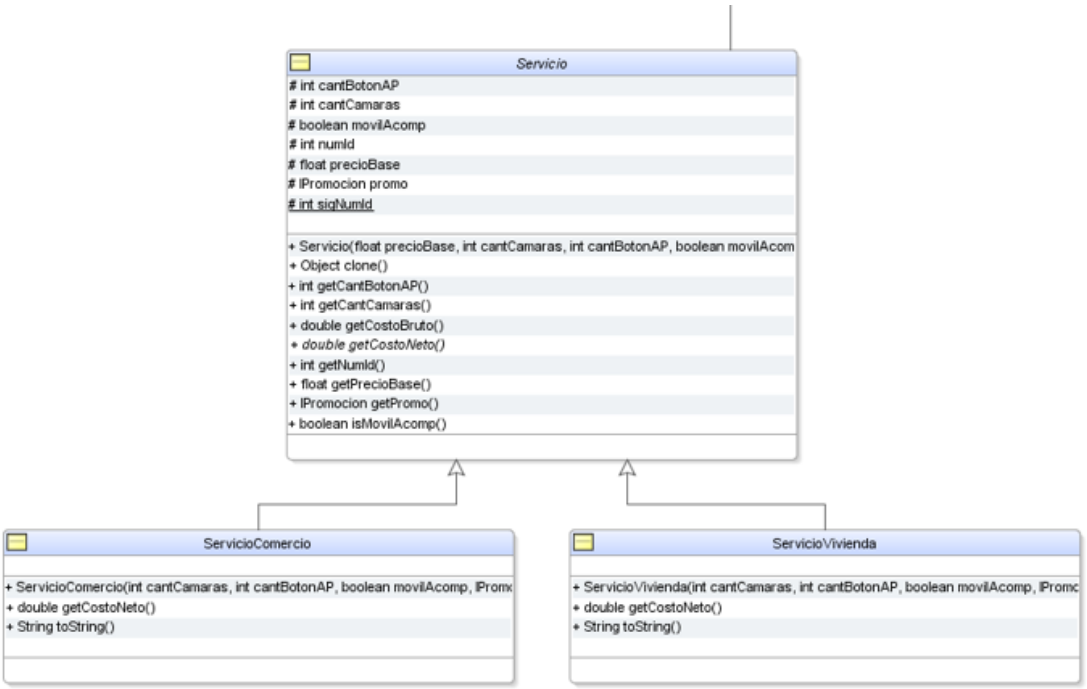


Clases Servicio, ServicioComercio y ServicioVivienda:

La clase 'Servicio' es de tipo abstracta debido a que será extendida por las clases 'SercivioVivienda' y 'ServicioComercio'. Esta implementa Cloneable ya que para clonar una factura es necesario hacer una clonación profunda en y clonar el hasmap el cual está cargado de servicios, por lo cual este debe ser clonado para tener una clonación correcta. En esta clase se definen todos los atributos necesarios relacionados con los diferentes servicios adquiribles por el usuario, tales como el precio, cantidad de cámaras, botón antipánico, etc. También se implementan los getters y setters de dichas variables, se calcula el costo bruto y se define una función abstracta llamada 'getCostoNeto()' la cual implementa double dispatch. Además se desarrolla el método 'objet clone()' que puede lanzar una excepción de tipo 'CloneNotSupportedException'. En esta clase también se define una variable de tipo 'Ipromocion' llamada 'promo', esta se encarga de hacer referencia al encapsulado de tipo 'PromocionDorada' o 'PromocionPlatino'.

Tanto en 'SercivioVivienda' y 'ServicioComercio' se ejecuta la función 'GetCostoNeto' la cual invoca la función 'GetPromoVivienda' y 'GetPromoComercio' respectivamente. Estas, por medio de la interfaz, invocan y ejecutan dichas funciones dentro del encapsulado el cual puede ser de tipo 'PromocionDorada' o 'PromocionPlatino'.

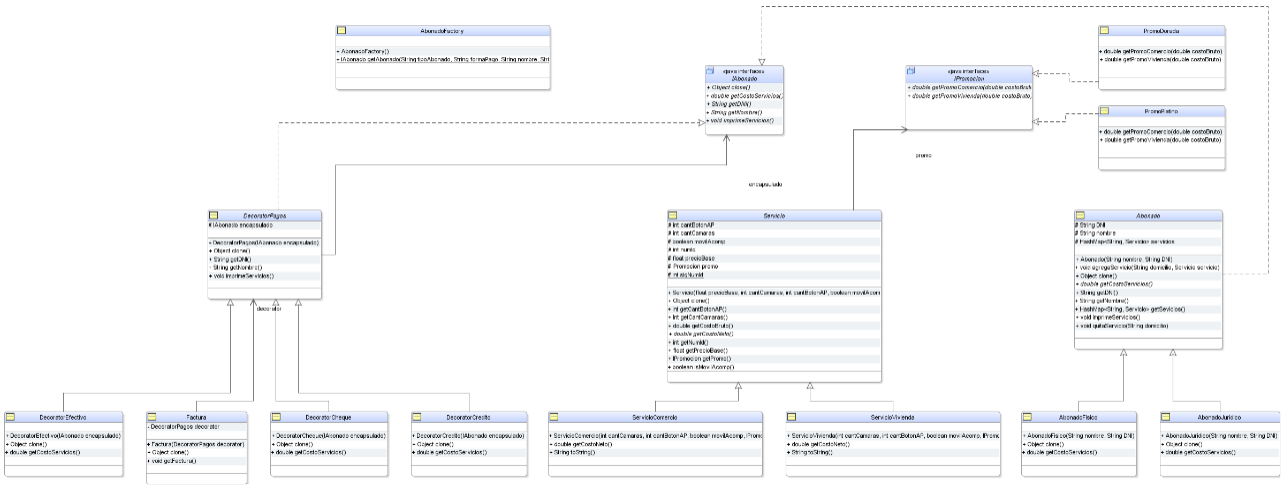
Debido a cómo se creó el Double Dispatch es posible en el futuro agregar nuevas promociones, pero no va a ser posible implementar nuevos servicios sin tener que modificar el código de la interfaz, clases y subclases.



Paquete excepciones

En este paquete, como su nombre indica, se encuentran las excepciones. 'DomicilioInvalidoExcepcion' se puede dar al querer quitar un determinado servicio de un domicilio invalido en el hashmap de servicios. 'TipoAbonadoInvalidoException' y 'TipoAbonadoInvalidoException' se pueden dar al ingresar un tipo de abonado que no sea de tipo jurídico o físico, o al ingresar un tipo de pago que no sea efectivo, crédito o tarjeta.

Diagrama de clases UML



Conclusiones

Para resumir, este proyecto nos permitió analizar una situación que podría presentarse en un contacto profesional y, a partir de ella, trabajar en equipo combinando nuestras habilidades individuales y colaborar de manera efectiva para alcanzar una solución óptima.

Es importante señalar que pudimos desarrollar diversas propuestas y, gracias al pensamiento de cada uno, la discusión y el debate constructivo, llegar a una solución conjunta que resulto ser la más adecuada para la situación que enfrentábamos.

A modo de comentario grupal nos gustaría destacar cuanto más fructífero en términos de aprendizaje consideramos la metodología de proyecto en relación a un examen donde haya que desarrollar código en un tiempo límite. Pensamos que esta forma de evaluar se acerca mucho más a una situación laborar real.