TP FINAL PROGRAMACIÓN III

PRIMERA PARTE – GRUPO 9

Carmenes, Santiago

García, Federico

Orte, Bautista

Zamora, Wendy

Patrones utilizados

Patrón Double Dispatch:

El patrón Double Dispatch resulta de gran utilidad en aquellas ocasiones en la que hay interacciones entre dos objetos, resuelve la situación en la que la respuesta del método va a depender de la clase pasada como parámetro. En vez de utilizar un único método que se encarga de analizar los distintos casos con if anidados, se crearán nuevos métodos con el mismo nombre para todas las clases candidatas a ser pasadas como parámetro.
Se define un método original que será encargado de llamar al método secundario del objeto pasado como parámetro. El objeto receptor de este mensaje secundario debe saber que hacer.
En nuestro programa el patrón Double dispatch es utilizado para aplicarle las promos dorada y platino. Le doy el comercio y la promo y la propia promo es la que me dice que descuento le tengo que hacer.

Patrón Factory:

En la utilización de este patrón, partiremos de una clase Factory que será la encargada de crear objetos. Este diseño nos otorga una ventaja significativa y es la no especificación de la clase exacta al momento de su creación proveyendo flexibilidad al programa.
En nuestro programa el patrón Factory utilizado para crear: abonados (persona física y persona jurídica), facturas (tarjeta, cheque y efectivo) y monitoreos (comercio y vivienda)

Patrón Singleton:

Este patrón de diseño es utilizado en aquellas ocasiones donde la instancia de una clase ocurrirá por única vez, otorgando la ventaja de que no va a ser necesario pasar a esta clase como parámetro, se podrá tener un acceso global a esta única instancia.
En nuestro programa el patrón Singleton es utilizado para crear una única instancia de Empresa, de promo platino y de promo dorada ya que en los tres casos son únicas.

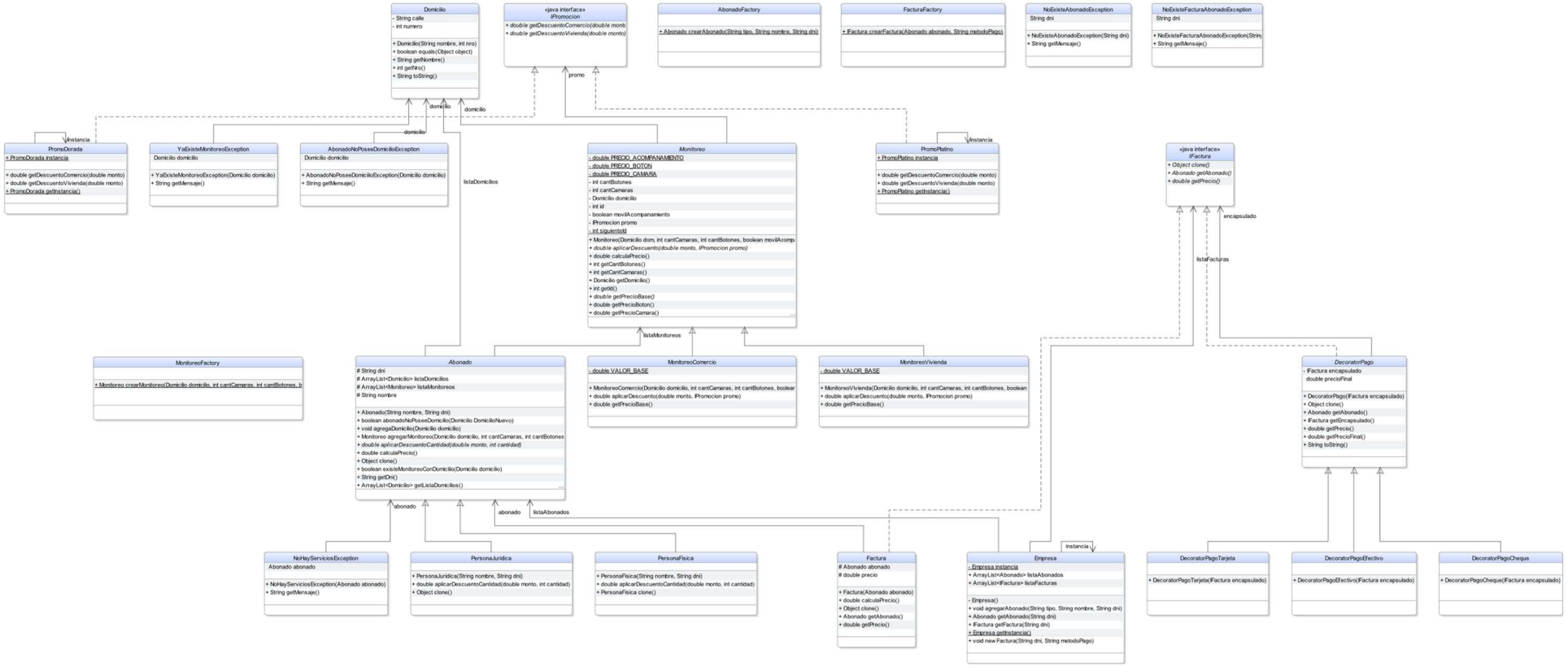
Patrón Decorator:

El patrón Decorator es una técnica de programación que nos permite agregar o quitar funcionalidad de un objeto existente de manera dinámica, sin tener que modificar el código original. En nuestro programa utilizamos el patrón Decorator para “decorar” (aplique el descuento o recargo pertinente) al encapsulado dependiendo de que tipo de pago utilice: efectivo, cheque o tarjeta.

Funcionamiento del programa

Primero creamos la instancia empresa y los dos tipos de promociones con el patrón Singleton. Declaramos tres domicilios diferentes. Creamos dos abonados utilizando el patrón Factory que lo invoca la clase empresa. En abonado recuperamos la instancia de uno de los abonados y lo almacenamos en una variable de tipo abonado (cabe recalcar que si el DNI ingresado no pertenece a ningún abonado, esto lanzará una excepción). Al abonado que tenemos en la variable le asignamos lo tres domicilio declarados anteriormente. Al mismo abonado le hacemos crear 3 monitoreos los cuales a la vez lo estamos devolviendo en variables para luego aplicarles promoción. A través de un patrón Factory generamos la factura enviando como parámetros el documento del abonado al cual le queremos generar la factura y el tipo de pago. Muestra la factura del abonado del cual le pasamos el documento como parámetro. Intenta clonar una factura, de igual manera mandado el documento del abonado como parámetro. Muestra la factura clonada anteriormente, de haber sido clonada. Intenta crear el abonado con el que estamos trabajando. Muestra el clon del abonado, de haber sido clonado. Y finalmente hace los catch de todas aquellas excepciones lanzadas y muestra los mensajes pertinentes.

A continuación adjuntamos el diagrama UML, para una mejor resolución se encuentra subido en el repositorio de git.


Domicilio
- String calle
- int numero
+ Domicilio(String nombre, int nro)
+ boolean equals(Object object)
+ String getNombre()
+ int getNro()
+ String toString()
«java interface»
IPromocion
+ double getDescuentoComercio(double monto)
+ double getDescuentoVivienda(double monto)
AbonadoFactory
+ Abonado crearAbonado(String tipo, String nombre, String dni)
FacturaFactory
+ IFactura crearFactura(Abonado abonado, String metodoPago)
NoExisteAbonadoException
String dni
+ NoExisteAbonadoException(String dni)
+ String getMensaje()
NoExisteFacturaAbonadoException
String dni
+ NoExisteFacturaAbonadoException(String
+ String getMensaje()
promo
domicilio
domicilio
domicilio
instancia
PromoDorada
+ PromoDorada instancia
+ double getDescuentoComercio(double monto)
+ double getDescuentoVivienda(double monto)
+ PromoDorada getInstancia()
YaExisteMonitoreoException
Domicilio domicilio
+ YaExisteMonitoreoException(Domicilio domicilio)
+ String getMensaje()
AbonadoNoPoseeDomicilioException
Domicilio domicilio
+ AbonadoNoPoseeDomicilioException(Domicilio domicilio)
+ String getMensaje()
listaDomicilios
Monitoreo
- double PRECIO_ACOMPANAMIENTO
- double PRECIO_BOTON
- double PRECIO_CAMARA
- int cantBotones
- int cantCamaras
- Domicilio domicilio
- int id
- boolean movilAcompanamiento
- IPromocion promo
- int siguienteId
+ Monitoreo(Domicilio dom, int cantCamaras, int cantBotones, boolean movilAcomp
+ double aplicarDescuento(double monto, IPromocion promo)
+ double calculaPrecio()
+ int getCantBotones()
+ int getCantCamaras()
+ Domicilio getDomicilio()
+ int getId()
+ double getPrecioBase()
+ double getPrecioBoton()
+ double getPrecioCamara()
instancia
PromoPlatino
+ PromoPlatino instancia
+ double getDescuentoComercio(double monto)
+ double getDescuentoVivienda(double monto)
+ PromoPlatino getInstancia()
«java interface»
IFactura
+ Object clone()
+ Abonado getAbonado()
+ double getPrecio()
encapsulado
listaFacturas
MonitoreoFactory
+ Monitoreo crearMonitoreo(Domicilio domicilio, int cantCamaras, int cantBotones, b
listaMonitoreos
Abonado
String dni
ArrayList<Domicilio> listaDomicilios
ArrayList<Monitoreo> listaMonitoreos
String nombre
+ Abonado(String nombre, String dni)
+ boolean abonadoNoPoseeDomicilio(Domicilio DomicilioNuevo)
+ void agregaDomicilio(Domicilio domicilio)
+ Monitoreo agregarMonitoreo(Domicilio domicilio, int cantCamaras, int cantBotones
+ double aplicarDescuentoCantidad(double monto, int cantidad)
+ double calculaPrecio()
+ Object clone()
+ boolean existeMonitoreoConDomicilio(Domicilio domicilio)
+ String getDni()
+ ArrayList<Domicilio> getListaDomicilios()
MonitoreoComercio
- double VALOR_BASE
+ MonitoreoComercio(Domicilio domicilio, int cantCamaras, int cantBotones, boolear
+ double aplicarDescuento(double monto, IPromocion promo)
+ double getPrecioBase()
MonitoreoVivienda
- double VALOR_BASE
+ MonitoreoVivienda(Domicilio domicilio, int cantCamaras, int cantBotones, boolean
+ double aplicarDescuento(double monto, IPromocion promo)
+ double getPrecioBase()
DecoratorPago
- IFactura encapsulado
double precioFinal
+ DecoratorPago(IFactura encapsulado)
+ Object clone()
+ Abonado getAbonado()
+ IFactura getEncapsulado()
+ double getPrecio()
+ double getPrecioFinal()
+ String toString()
abonado
abonado
listaAbonados
instancia
NoHayServiciosException
Abonado abonado
+ NoHayServiciosException(Abonado abonado)
+ String getMensaje()
PersonaJuridica
+ PersonaJuridica(String nombre, String dni)
+ double aplicarDescuentoCantidad(double monto, int cantidad)
+ Object clone()
PersonaFisica
+ PersonaFisica(String nombre, String dni)
+ double aplicarDescuentoCantidad(double monto, int cantidad)
+ PersonaFisica clone()
Factura
Abonado abonado
double precio
+ Factura(Abonado abonado)
+ double calculaPrecio()
+ Object clone()
+ Abonado getAbonado()
+ double getPrecio()
Empresa
- Empresa instancia
+ ArrayList<Abonado> listaAbonados
+ ArrayList<Factura> listaFacturas
- Empresa()
+ void agregarAbonado(String tipo, String nombre, String dni)
+ Abonado getAbonado(String dni)
+ IFactura getFactura(String dni)
+ Empresa getInstancia()
+ void newFactura(String dni, String metodoPago)
DecoratorPagoTarjeta
+ DecoratorPagoTarjeta(IFactura encapsulado)
DecoratorPagoEfectivo
+ DecoratorPagoEfectivo(IFactura encapsulado)
DecoratorPagoCheque
+ DecoratorPagoCheque(IFactura encapsulado)