**PRUEBAS UNITARIAS**

En el caso de las pruebas unitarias se decidió realizar 10 entre las cuales tenemos

* assertEquals
* assertTrue
* assertSame
* assertNotNull
* assertFalse
* assertNull
* assertNotSame

Para el primer assert (TestObjetos()) comparamos dos objetos para saber si dichos objetos (controladores) eran iguales, en este caso AddClienteController y AddUsuarioController.

Para el segundo assert (TestNull()) queríamos conocer si el campo ubicación perteneciente a la clase local no contenía ningún dato.

Para el tercer assert (TestSame()) en este caso se requería conocer si dos objetos estaban referencian al mismo objeto.

Para el cuarto assert (TestNotNull()) conocer si el nombre que recibía dicho local era no nulo.

Para el quinto assert (TestTrue()) verificamos si el local poseía internet wifi o no.

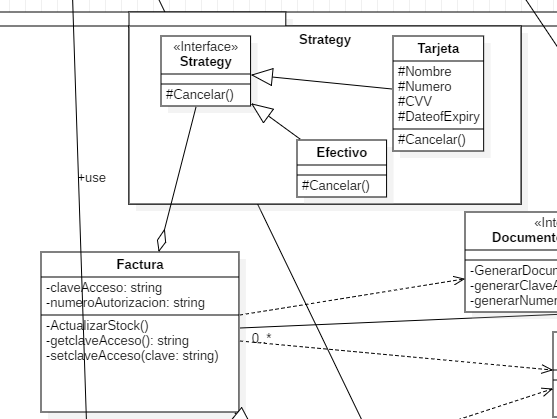
Para el sexto assert (TestFalse()) conocer si el resultado que se estaba iterando era falso es decir, no existía un resultado siguiente con relación a la base de datos.

Para el séptimo assert (TestSame2()) verificar si los objetos que en este caso son persona y usuario estaban referencian al mismo objeto.

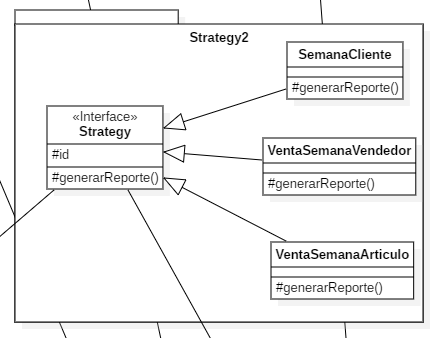
Para el octavo assert (TestNull2()) verificar si el campo asisRest era de valor null.

Para el noveno assert (TestNotNullMethods()) verificar si los campos recibidos de los objetos local y usuario no referenciaban a un miso objeto.

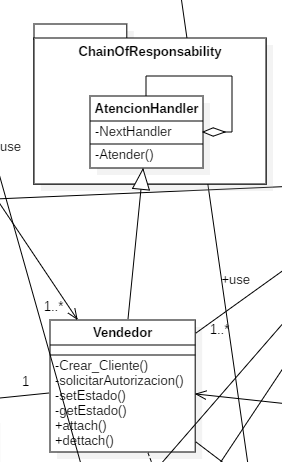
Para el decimo assert (TestNotNullConnection()) se quería conocer si la conexión a la base de datos no era null, es decir si se adquiría una base.

**Patrones de Diseño**

Strategy

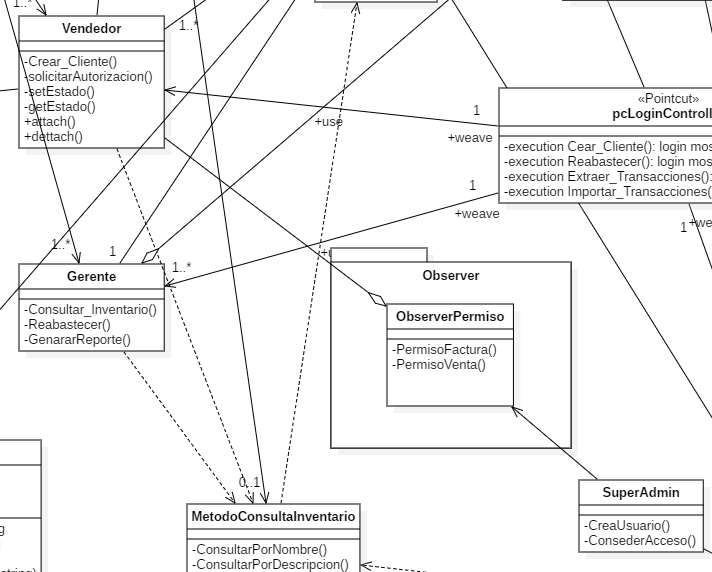
Se uso para poder representar las formas en que se iba a poder cancelar una factura ya que estas pueden variar en tiempo de ejecución por lo que nos basamos de un objeto abstracto y no de uno concreto para de esta forma cumplir con el principio de Open Close

Al poseer Distintos tipos de reportes decidimos poder trabajar con el strategy ya que nos manejamos con un grupo de objetos abstractos ya que el gerente y otra clase hacían uso de estos y de esta forma mantenemos abstracción en los objetos y dejamos a las subclases comportarse como deseen.



Chain of Responsability

Al presentarse la funcionalidad de que cuando un cliente llega a un local un vendedor que se encuentre disponible debe de tratarlo por lo que es un claro ejemplo de pasarse entre objetos esa solicitud hasta encontrar un vendedor que pueda tratarlo

Observer

La funcionalidad del SuperAdmin de otorgar permisos en tiempo de ejecución de forma dinámica alude a añadir esta funcionalidad ya que un grupo de vendedores pueden estar suscriptos a un observer dependiendo de la decisión del superadmin de otorgar a ciertos vendedores esta funcionalidad cuando se requiera y quitárselas automáticamente cuando ya no hagan uso de esta