**UTN - Facultad Regional Santa Fe**

**MÉTODOS ÁGILES**

**PARA EL DESARROLLO DE SOFTWARE**

Trabajo Práctico Programación Extrema

**Alumnos:**

Berti, Fernando bertilxi@gmail.com

Campodonico, Daniel d.campodonico@hotmail.com

Gioria, Emiliano emigioria@hotmail.com

Moretti, Lucas moretti.lucas@hotmail.com

Rebechi, Esteban estebanrebechi\_5@hotmail.com

Rico, Andrés andres.rico94@gmail.com

**Docentes:**

* Ing. Bracalenti, Claudio.
* Ing. Ledesma, Rodrigo.

**Año 2016**

Índice

[Descripción del sistema 5](#_Toc469632629)

[Equipo de desarrollo 5](#_Toc469632630)

[Planificación del Release 5](#_Toc469632631)

[Estimación de Esfuerzo, Riesgo y Prioridad 13](#_Toc469632632)

[Spikes 14](#_Toc469632633)

[Historia 1: ABM Vendedor 14](#_Toc469632634)

[Historia 2: ABM propietario 16](#_Toc469632635)

[Historia 3: ABM inmueble, Historia 4: Consulta inmueble 17](#_Toc469632636)

[Historia 5: Catálogo de Inmuebles 19](#_Toc469632637)

[Historia 6: ABM cliente 20](#_Toc469632638)

[Historia 7: Generar Reserva 22](#_Toc469632639)

[Historia 8: Venta 23](#_Toc469632640)

[TaskCard 4: Vista principal para la administración 25](#_Toc469632641)

[TaskCard 1: Vista y lógica del Login 26](#_Toc469632642)

[Definición de Velocidad y Alcance 27](#_Toc469632643)

[Metáfora 27](#_Toc469632644)

[Fecha de Inicio del desarrollo 27](#_Toc469632645)

[Librerías, herramientas y tecnologías utilizadas 28](#_Toc469632646)

[Planificación Iteración 1 29](#_Toc469632647)

[Tareas 29](#_Toc469632648)

[ABM Vendedor 29](#_Toc469632649)

[Vista y lógica del Login 29](#_Toc469632650)

[Entidad vendedor 29](#_Toc469632651)

[Vista alta, modificar y baja Vendedor 30](#_Toc469632652)

[Vista principal para la administración 31](#_Toc469632653)

[Lógica alta, modificación y baja vendedor 31](#_Toc469632654)

[Persistidor vendedor 32](#_Toc469632655)

[ABM Propietario 33](#_Toc469632656)

[Entidad propietario 33](#_Toc469632657)

[Vista alta, modificar y baja Propietario 33](#_Toc469632658)

[Lógica alta, modificación y baja propietario 34](#_Toc469632659)

[Persistidor propietario 35](#_Toc469632660)

[ABM Inmueble 36](#_Toc469632661)

[Entidad inmueble 36](#_Toc469632662)

[Clases de datos 37](#_Toc469632663)

[Vista alta, modificar y baja inmueble 38](#_Toc469632664)

[Lógica alta, modificación y baja inmueble 39](#_Toc469632665)

[Persistidor inmueble 40](#_Toc469632666)

[ABM Cliente 41](#_Toc469632667)

[Entidad cliente 41](#_Toc469632668)

[Vista alta, modificar y baja cliente 42](#_Toc469632669)

[Lógica alta, modificación y baja cliente 43](#_Toc469632670)

[Persistidor cliente 43](#_Toc469632671)

[Estimación de Esfuerzo 44](#_Toc469632672)

[Conclusiones parciales de lo estimado 44](#_Toc469632673)

[Planificación Iteración 2 45](#_Toc469632674)

[Tareas 45](#_Toc469632675)

[Consulta inmueble 45](#_Toc469632676)

[Vista de consulta inmueble 45](#_Toc469632677)

[Lógica y persistidor de consulta inmueble 46](#_Toc469632678)

[Catálogo de inmuebles 47](#_Toc469632679)

[Vista Alta Catálogo Inmueble 47](#_Toc469632680)

[Lógica alta catálogo de inmueble y generar PDF 47](#_Toc469632681)

[Generar reserva 48](#_Toc469632682)

[Vista alta, baja y listar reserva 48](#_Toc469632683)

[Lógica alta, baja reserva y generar PDF 49](#_Toc469632684)

[Lógica envío de mail 49](#_Toc469632685)

[Cambios ABM Cliente 50](#_Toc469632686)

[Persistidor y entidad reserva 50](#_Toc469632687)

[Ventas 51](#_Toc469632688)

[Vista alta y listar venta 51](#_Toc469632689)

[Lógica alta venta y generar PDF 52](#_Toc469632690)

[Lógica imprimir venta 53](#_Toc469632691)

[Persistidor y entidad venta 53](#_Toc469632692)

[Agregar EstadoInmueble a inmueble 54](#_Toc469632693)

[Estimación de Esfuerzo 55](#_Toc469632694)

[Conclusiones parciales de lo estimado 55](#_Toc469632695)

[Código Fuente 56](#_Toc469632696)

[Pruebas de Unidad 56](#_Toc469632697)

[Refactorización 56](#_Toc469632698)

[Estándares de programación 56](#_Toc469632699)

[Conclusiones 56](#_Toc469632700)

[Código Fuente y Pruebas de Unidad 59](#_Toc469632701)

[Iteración 1 59](#_Toc469632702)

[Taskcard 1 Vista y lógica del Login 59](#_Toc469632703)

[Taskcard 2 Entidad vendedor 62](#_Toc469632704)

[Taskcard 3 Vista alta, modificar y baja vendedor 63](#_Toc469632705)

[Taskcard 4 Vista principal para la administración 78](#_Toc469632706)

[Taskcard 5 Lógica alta, modificación y baja vendedor 79](#_Toc469632707)

[Taskcard 6 Persistidor vendedor 85](#_Toc469632708)

[Taskcard 7 Entidad Propietario 87](#_Toc469632709)

[Taskcard 8 Vista alta, modificar y baja propietario 87](#_Toc469632710)

[Taskcard 9 Lógica alta, modificar y baja propietario 101](#_Toc469632711)

[Taskcard 10 Persistidor Propietario 110](#_Toc469632712)

[Taskcard 11 Entidad inmueble 111](#_Toc469632713)

[Taskcard 12 Clases de datos 115](#_Toc469632714)

[Taskcard 13 Vista alta, modificar y baja inmueble 117](#_Toc469632715)

[Taskcard 14 Lógica alta, modificación y baja inmueble 137](#_Toc469632716)

[Taskcard 15 Persistidor inmueble 148](#_Toc469632717)

[Taskcard 16 Entidad cliente 149](#_Toc469632718)

[Taskcard 17 Vista alta, modificar y baja cliente 151](#_Toc469632719)

[Taskcard 18 Lógica alta, modificar y baja cliente 170](#_Toc469632720)

[Taskcard 19 Persistidor cliente 178](#_Toc469632721)

[Iteración 2: 179](#_Toc469632722)

[Taskcard 20 Vista de consulta inmueble 179](#_Toc469632723)

[Taskcard 21 Lógica y persistidor de consulta inmuebles 181](#_Toc469632724)

[Taskcard 22 Vista alta catálogo de inmueble 184](#_Toc469632725)

[Taskcard 23 Lógica alta catálogo de inmueble y generar PDF 186](#_Toc469632726)

[Taskcard 24 Vista alta, baja y listar reserva 188](#_Toc469632727)

[Taskcard 25 Lógica alta, baja reserva y generar PDF 198](#_Toc469632728)

[Taskcard 26 Lógica envío de mail 209](#_Toc469632729)

[Taskcard 27 Cambios ABM Cliente 210](#_Toc469632730)

[Taskcard 28 Persistidor y entidad reserva 212](#_Toc469632731)

[Taskcard 29 Vista alta y listar venta 213](#_Toc469632732)

[Taskcard 30 Lógica alta venta y generar PDF 224](#_Toc469632733)

[Taskcard 31 Lógica imprimir venta 231](#_Toc469632734)

[Taskcard 32 Persistidor y entidad venta 232](#_Toc469632735)

[Taskcard 33 Agregar EstadoInmueble a inmueble 233](#_Toc469632736)

# Descripción del sistema

La inmobiliaria FRSF nos ha solicitado desarrollar un sistema de información que permita gestionar operaciones de compra-venta de inmuebles, desde que un propietario se presenta con un inmueble para la venta hasta que se concreta la misma.

# Equipo de desarrollo

* Fernando Berti: Programador.
* Daniel Campodonico: Programador.
* Emiliano Gioria: Programador.
* Lucas Moretti: Programador.
* Esteban Rebechi: Programador.
* Andrés Rico: Programador.

Para la realización de este trabajo práctico con la metodología ágil XP, el cliente del sistema es el Ing. Rodrigo Ledesma.

La programación de a pares se realizó para las dos iteraciones de la siguiente manera:

* Pareja 1: Berti/Rebechi.
* Pareja 2: Campodonico/Moretti.
* Pareja 3: Gioria/Rico.

# Planificación del Release

Historias de usuario

1. ***ABM Vendedor - Baja***

El vendedor es el usuario del sistema que realiza todas las operaciones sobre el mismo. Para su acceso se requiere los datos personales y una clave de acceso, la cual se debe solicitar antes de acceder a las funcionalidades del sistema.

1. ***ABM propietario - Alta***

Se debe permitir cargar, modificar y eliminar propietarios. Sus datos son nombre y apellido, tipo documento, número documento, calle, número, localidad, provincia, teléfono, email.

1. ***ABM inmueble - Alta***

El sistema debe permitir cargar y modificar inmuebles asociados a un propietario (se debe elegir el propietario previamente cargado). Cuando se carga un inmueble, el sistema debe otorgar código de inmueble, guardar la fecha de carga y asignar estado alta al inmueble. El código de inmueble es un valor secuencial independiente para cada inmueble.

Los datos a cargar para un inmueble son:

* provincia: por defecto Santa Fe;
* localidad: para la provincia de Santa Fe, se deben incluir las siguientes localidades en un listado: Santa Fe, Santo Tomé, Sauce Viejo, Rincón, Colastiné Norte, Colastiné Sur. En caso de que la localidad no sea ninguna de las anteriores, se debe poder ingresar una;
* calle/número: si la localidad elegida tiene asociada calles, se debe elegir la misma de un listado. Si no tiene asociada calles, se permite ingresar la calle/número o una ubicación;
* piso/departamento;
* barrio;
* tipo de inmueble: se debe poder elegir de un listado. Los tipos de inmuebles son: L/local-oficina, C/casa, D/departamento, T/terreno, Q/quinta, G/galpón;
* precio de venta (obligatorio) ;
* orientación (norte, sur, este, oeste, noreste, noroeste, sureste, suroeste), frente (metros), fondo (metros), superficie (m2). Datos del edificio: propiedad horizontal (si/no), superficie (m2), antigüedad, dormitorios, baños, garaje/cochera, patio, piscina, agua corriente, cloacas, gas natural, agua caliente, teléfono, lavadero, pavimento;
* Foto y observaciones;

1. ***Consulta Inmueble - Alta***

El sistema debe permitir consultar inmueble por provincia, localidad, barrio, tipo, cantidad de dormitorios y precio (rango).

1. ***Catálogo de Inmuebles - Media***

El sistema debe permitir generar un catálogo con inmuebles para un cliente (con posibilidad de elegir los inmuebles). Debe incluir para cada inmueble una foto, código inmueble, tipo de inmueble, localidad, dirección, barrio, cantidad de dormitorios, baños, garaje, patio, superficie de terreno, superficie edificada, precio. En cada página debe figurar la fecha de emisión.

1. ***ABM cliente - Media***

El sistema debe permitir cargar y modificar datos de clientes: nombre, apellido, teléfono y los datos del inmueble buscado: tipo de inmueble, localidad, barrios, características, monto disponible.

1. ***Generar Reserva - Media***

El cliente puede reservar un inmueble elegido generándose un documento reserva del inmueble a partir de los datos del cliente y del inmueble. Se debe ingresar el importe de la reserva y el tiempo de vigencia de la misma. Se debe marcar que el inmueble está reservado y por quién en caso de que otro cliente desee reservarlo. El documento debe quedar asociado al Cliente y debe enviarse a su casilla de correo.

1. ***Venta - Media***

Al generarse la venta se genera un documento similar al de reserva indicando los datos del inmueble vendido y el cliente. Este documento debe imprimirse para la firma por parte del cliente. Así también debe figurar el monto de la venta y se debe marcar como vendido a dicho inmueble. El documento de venta debe asociarse al cliente y al inmueble.

1. ***Publicar - Baja***

La inmobiliaria posee un sitio web donde publica los inmuebles de colocando los datos de los inmuebles más relevantes, las fotos y un video de tour del inmueble dependiendo de la calidad del mismo. En dicho sitio deben figurar los datos de contacto de la inmobiliaria.

Historias Refinadas

|  |  |
| --- | --- |
| **Historia de Usuario** | |
|
| Numero: 1 | Nombre: ABM Vendedor |
| Usuario: Fernando Berti | |
| Modificación Historia: | Iteración Asignada: 1 |
| Prioridad en Negocio: Baja | Puntos Estimados: 2 |
|
| Riesgo en Desarrollo: Baja |
|
| Descripción: | |
| Se deberán ingresar los siguientes datos personales para realizar el alta: (todos los datos son obligatorios)   * + Nombre (String:30),   + Apellido (String:30),   + Tipo de Documento (DNI, LC, LE, Pasaporte, Cédula Extranjera),   + Número de documento (String:30) y   + Contraseña (String:100).   La contraseña debe estar compuesta por caracteres alfanuméricos. Se podrán modificar todos los datos. El DNI no podrá repetirse.  Cada vendedor tendrá una clave de acceso al sistema, la cual, luego de ingresar al sistema, solo permitirá acceder a las siguientes funcionalidades:   * ABM Inmueble. * ABM Propietario. * Consulta inmueble. * Catálogo de inmueble. * ABM Cliente. * Generar Reserva. * Venta. * Publicar.   Además, se debe crear un vendedor con el rol de administrador el cual tendrá acceso a la funcionalidad de administrar otros vendedores. | |
|
|
|
| Observaciones:  La contraseña se deberá hashear con un salt distinto para cada usuario, el cual se deberá almacenar en la entidad correspondiente a Vendedor.  El vendedor debe tener un atributo Estado para manejar la baja lógica. | |
|
| **Historia de Usuario** | |
|
| Numero: 2 | Nombre: ABM propietario |
| Usuario: Lucas Moretti | |
| Modificación Historia: | Iteración Asignada: 1 |
| Prioridad en Negocio: Alta | Puntos Estimados: 2 |
|
| Riesgo en Desarrollo: Baja |
|
| Descripción: | |
| Se deberán ingresar los siguientes datos personales para realizar el alta de un propietario:   * Nombre (String:30), * Apellido (String:30), * Tipo de Documento (DNI, LC, LE, Pasaporte, Cédula Extranjera), * Número de documento (String:30), * Dirección (País, Provincia, Localidad, Barrio, Calle, Número, Piso, Departamento, Otros), * Teléfono (String:30) y * Email (String:30).   Los siguientes datos son obligatorios: nombre, apellido, tipo, número de documento, calle, número, país, provincia, localidad, barrio, teléfono.  Todos los datos pueden ser modificados.  Todos los datos pueden repetirse, excepto el tipo y número de documento.  No se llevará historial de cambios del mismo.  El propietario no podrá ser dado de baja en caso de tener alguna propiedad asociada.  La baja se realiza de manera lógica. | |
|
|
|
| Observaciones:  Una dirección está compuesta por:   * Localidad (String:50), * Barrio (String:50), * Calle (String:50), * Número (String:30), * Piso (String:30), * Departamento (String:30), * Otros (String:100).   Una Localidad tiene una Provincia (String:50).  Una Provincia tiene un País (String:50). | |
|
| **Historia de Usuario** | |
|
| Numero: 3 | Nombre: ABM inmueble |
| Usuario: Emiliano Gioria | |
| Modificación Historia: | Iteración Asignada: 1 |
| Prioridad en Negocio: Alta | Puntos Estimados: 2 |
|
| Riesgo en Desarrollo: Media |
|
| Descripción: | |
| Se deberán ingresar los siguientes datos para el alta de un inmueble:   * Código del inmueble (código generado por el sistema), * Fecha de carga (Date, fecha actual, no se puede modificar), * Propietario (previamente cargado, obligatorio), * País (String:50), * Provincia (String:50): por defecto Santa Fe (obligatorio), * Localidad (String:50): para la provincia de Santa Fe, se deben incluir las siguientes localidades en un listado: Santa Fe, Santo Tomé, Sauce Viejo, Rincón, Colastiné Norte, Colastiné Sur. En caso de que la localidad no sea ninguna de las anteriores, se debe poder ingresar una (obligatorio), * Calle (String:50) * Número (String:30, numérico): si la localidad elegida tiene asociada calles, se debe elegir la misma de un listado. Si no tiene asociada calles, se permite ingresar la calle y número o una ubicación mediante el campo Otros (obligatorio), * Piso (String:30), * Departamento (String:30), * Barrio (String:50, obligatorio), * Tipo de inmueble (listado: L/local-oficina, C/casa, D/departamento, T/terreno, Q/quinta, G/galpón) (obligatorio), * Precio de venta (Double, obligatorio), * Orientación (norte, sur, este, oeste, noreste, noroeste, sureste, suroeste); * Medidas (Double):   + Frente (en metros),   + Fondo (en metros),   + Superficie (en m2). * Datos del edificio:   + Propiedad horizontal (Boolean),   + Superficie (Double, en m2),   + Antigüedad (Integer, en años),   + Dormitorios (Integer),   + Baños (Integer),   + Garaje/cochera (Boolean),   + Patio (Boolean),   + Piscina (Boolean),   + Agua corriente (Boolean),   + Cloacas (Boolean),   + Gas natural (Boolean),   + Agua caliente (Boolean),   + Teléfono (Boolean),   + Lavadero (Boolean),   + Pavimento (Boolean). * Fotos (sin restricción de tamaño), * Observaciones (String:300).   Cada inmueble debe tener un estado (alta, baja). Luego del alta el estado pasa a ser “Alta”.  Todos los datos se pueden repetir, excepto el código.  Se pueden modificar todos los cambios.  La baja se realiza de forma lógica seteando el estado en “Baja”.  Se debe llevar un historial de todos los cambios. | |
|
|
|
| Observaciones: | |
|
| **Historia de Usuario** | |
|
| Numero: 4 | Nombre: Consulta Inmueble |
| Usuario: Andrés Rico | |
| Modificación Historia: | Iteración Asignada: 2 |
| Prioridad en Negocio: Alta | Puntos Estimados: 1 |
|
| Riesgo en Desarrollo: Baja |
|
| Descripción: | |
| El sistema debe permitir consultar por inmuebles que cumplan determinados criterios de búsqueda que los ingresa el usuario:   * País (String:50), * Provincia (String:50), * Localidad (String:50), * Barrio (String:50), * Tipo de inmueble (TipoInmueble), * Cantidad de dormitorios (Integer), * Precio (seleccionando un rango) y * Estado (vendido, disponible).   Se listarán los inmuebles de la base de datos que cumplan con los criterios de búsqueda y se los lista en una tabla mostrando el tipo, la ubicación y el propietario del mismo.  Se permitirá seleccionar un inmueble y luego elegir alguna de las siguientes funciones:   * Baja Inmueble. * Modificación Inmueble. * Catálogo de inmueble (Genera un catálogo con los inmuebles seleccionados). * Generar Reserva. * Venta.   Si se selecciona más de un inmueble se deben deshabilitar todos los botones que permitan las acciones anteriores excepto el de generar catálogo. | |
|
|
|
| Observaciones:  El criterio de búsqueda para los campos que son del tipo String debe ser “contiene”. | |
|
| **Historia de Usuario** | |
|
| Numero: 5 | Nombre: Catálogo de Inmuebles |
| Usuario: Esteban Rebechi | |
| Modificación Historia: | Iteración Asignada: 2 |
| Prioridad en Negocio: Media | Puntos Estimados: 2 |
|
| Riesgo en Desarrollo: Alta |
|
| Descripción: | |
| Se debe permitir generar un catálogo de inmuebles para un cliente. Para cada inmueble dentro del catálogo se debe incluir una foto, código inmueble, tipo de inmueble, localidad, dirección, barrio, cantidad de dormitorios, baños, garaje, patio, superficie terreno, superficie edificada, precio.  En cada página debe figurar la fecha de emisión del catálogo.  La foto que irá en el catálogo se podrá elegir entre las fotos que ya están cargadas en el sistema para cada inmueble.  El diseño del catálogo deberá contemplar un encabezado donde figure la fecha de emisión del mismo, el logo del sistema y la identificación de la inmobiliaria. Un pie de página en donde figure la página actual y la cantidad de páginas en total.  Las imágenes de los inmuebles en el catálogo deben tener un tamaño considerable; 3 inmuebles por página.  Luego de generar el catálogo se debe poder mostrar al usuario y permitirle imprimirlo y guardarlo. | |
|
|
|
| Observaciones:  En el caso de los departamentos la superficie del terreno y la cubierta es la misma.  El catálogo es generado como un archivo PDF. | |
|
| **Historia de Usuario** | |
|
| Numero: 6 | Nombre: ABM cliente |
| Usuario: Daniel Campodonico | |
| Modificación Historia: | Iteración Asignada: 1 |
| Prioridad en Negocio: Media | Puntos Estimados: 2 |
|
| Riesgo en Desarrollo: Baja |
|
| Descripción: | |
| Para realizar el alta de un nuevo cliente se deberá permitir ingresar los siguientes datos:   * Nombre (String:30), * Apellido (String:30), * Número de documento (String:30), * Teléfono (String:30) y * Los datos del inmueble buscado:   + Tipo de inmueble [L/local-oficina, C/casa, D/departamento, T/terreno, Q/quinta, G/galpón] (obligatorio),   + Localidad (String:50),   + Barrios (Lista de String:50),   + Características:     - Propiedad horizontal (Booleano),     - Superficie en m2 (Decimal),     - Antigüedad en años (Entero),     - Dormitorios mínimos (Entero),     - Baños mínimos (Entero),     - Garaje/cochera (Booleano),     - Patio (Booleano),     - Piscina (Booleano),     - Agua corriente (Booleano),     - Cloacas (Booleano),     - Gas natural (Booleano),     - Agua caliente (Booleano),     - Teléfono (Booleano),     - Lavadero (Booleano),     - Pavimento (Booleano).   Son obligatorios nombre, apellido, tipo y número de documento.  Se puede modificar y repetir todos los datos menos el tipo y número de documento.  No se llevará el historial de cambios del cliente. | |
|
|
|
| Observaciones: | |
|
| **Historia de Usuario** | |
|
| Numero: 7 | Nombre: Generar Reserva |
| Usuario: Daniel Campodonico | |
| Modificación Historia: | Iteración Asignada: 2 |
| Prioridad en Negocio: Media | Puntos Estimados: 1 |
|
| Riesgo en Desarrollo: Baja |
|
| Descripción: | |
| El cliente debe poder generar una reserva para un inmueble elegido; la reserva debe tener los siguientes datos:   * Cliente (Cliente), * Inmueble (Inmueble), * Fecha Actual (Date), * Fecha de vigencia (Date), * Importe (Double).   Para generar la reserva el usuario (vendedor) debe ingresar el importe de la reserva y el tiempo de vigencia de la misma.  El inmueble se debe marcar como reservado.  Se debe generar un documento asociado a la reserva con los datos del cliente, el inmueble, el propietario y de la reserva (fecha de vigencia e importe).  El documento se debe asociar al cliente y ser enviado a su dirección de correo.  A su vez, se guardará en la base de datos y se le preguntará al usuario si desea imprimirlo.  También se debe poder listar las reservas seleccionando un cliente o un inmueble.  Se debe permitir dar de baja una reserva. | |
|
|
|
| Observaciones:  La baja de una reserva se debe implementar de manera lógica. | |
|
| **Historia de Usuario** | |
|
| Numero: 8 | Nombre: Venta |
| Usuario: Fernando Berti | |
| Modificación Historia: | Iteración Asignada: 2 |
| Prioridad en Negocio: Media | Puntos Estimados: 2 |
|
| Riesgo en Desarrollo: Baja |
|
| Descripción: | |
| El usuario debe poder generar una venta seleccionando un inmueble e ingresando los siguientes datos:   * Cliente (ComboBox<Cliente>) (comprador), * Importe (Double), * Medio de pago (String).   Una vez confirmada la venta, se genera un documento de venta donde figuran los datos del cliente (comprador), del inmueble, del propietario (vendedor del inmueble) y datos de la venta (importe, medio de pago, fecha).  El documento se debe poder imprimir y también almacenarse digitalmente.  El inmueble debe poder ser marcado como vendido.  El documento generado se debe asociar al cliente y al inmueble.  Una venta no puede ser modificada ni eliminada. | |
|
|
|
| Observaciones: | |
|
| **Historia de Usuario** | |
|
| Numero: 9 | Nombre: Publicar |
| Usuario: Fernando Berti | |
| Modificación Historia: | Iteración Asignada: - |
| Prioridad en Negocio: Baja | Puntos Estimados: 3 |
|
| Riesgo en Desarrollo: Alta |
|
| Descripción: | |
| Se debe poder publicar un inmueble en el sitio web de la empresa con los siguientes datos:   * código inmueble (Integer), * tipo de inmueble (TipoInmueble), * Localidad (String:50), * Dirección, * Barrio (String:50), * Cantidad de dormitorios (Integer), * Baños (Integer), * Garaje (Integer), * Patio (Boolean), * Superficie terreno (Double), * Superficie edificada (Double), * Precio (Double), * Fotos del inmueble y * Video del tour dentro del inmueble.   Se mostrarán los datos disponibles del inmueble.  Nunca se mostrará el propietario del inmueble.  Se debe mostrar los siguientes datos de contacto de la inmobiliaria:   * Nombre, * Dirección, * Teléfono, * Fax, * Correo electrónico. | |
|
|
|
| Observaciones:  Al crear una publicación se debe eliminar la anterior. | |
|

# Estimación de Esfuerzo, Riesgo y Prioridad

Para el equipo de desarrollo, un Story Point es equivalente a una semana, de 5 días, de 3 horas de trabajo cada día.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Historia** | | **Story Points** | **Prioridad en Negocio** | **Riesgo en Desarrollo** |
| **Numero** | **Nombre** |
| 1 | ABM Vendedor | 2 | Baja | Baja |
| 2 | ABM propietario | 2 | Alta | Baja |
| 3 | ABM inmueble | 2 | Alta | Media |
| 4 | Consulta Inmueble | 1 | Alta | Baja |
| 5 | Catálogo de Inmuebles | 2 | Media | Alta |
| 6 | ABM cliente | 2 | Media | Baja |
| 7 | Generar Reserva | 1 | Media | Baja |
| 8 | Venta | 2 | Media | Baja |
| 9 | Publicar | 3 | Baja | Alta |

|  |  |  |  |
| --- | --- | --- | --- |
| **Riesgo** | **Prioridad** | | |
| **Alta** | **Media** | **Baja** |
| **Alto** |  | 5 | 9 |
|
|
| **Medio** | 3 |  |  |
|
|
| **Bajo** | 2, 4 | 6, 7, 8 | 1 |
|
|

En las primeras reuniones con el cliente, se realizaron consultas y sugerencias para las historias de usuario que fueron proporcionadas en previas reuniones. Se refinaron las historias hasta que las ambigüedades desaparecieron y se obtuvo una aceptación por parte del cliente.

En otra reunión se definió el alcance que tendría que tener el primer release. No existieron negociaciones ya que el cliente decidió no incluir la historia de usuario “Publicar” y el equipo tenía la misma idea para el alcance, ya que la velocidad de desarrollo definida permitía cumplirlo dentro del tiempo aceptable para la entrega del primer release.

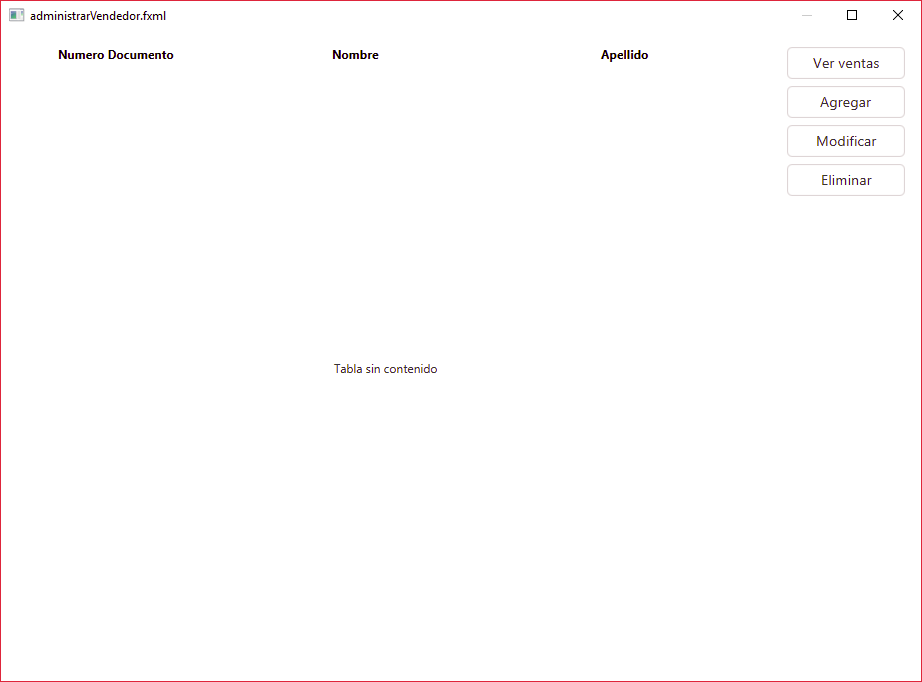
En las reuniones posteriores se expusieron los spikes para las pantallas que el sistema tendría que mostrar y se aceptaron por parte del cliente.

Durante las iteraciones, cuando surgían cuestiones que generaban interpretaciones ambiguas dentro del equipo de desarrollo, se resolvían rápidamente con una consulta al cliente vía email o presencial. El tiempo de respuesta nunca paralizó el avance del proyecto.

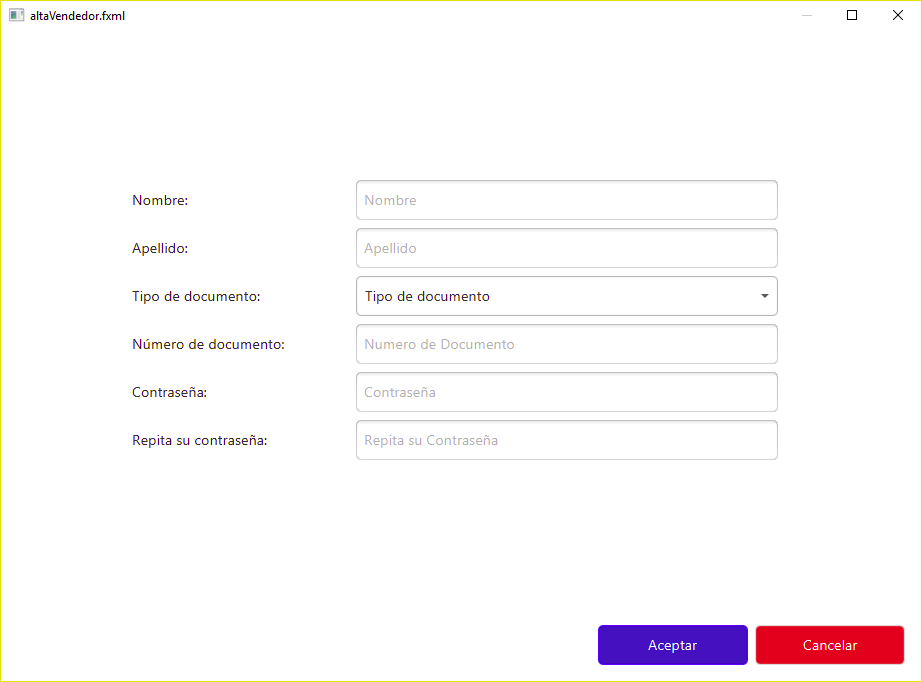
# Spikes

### Historia 1: ABM Vendedor

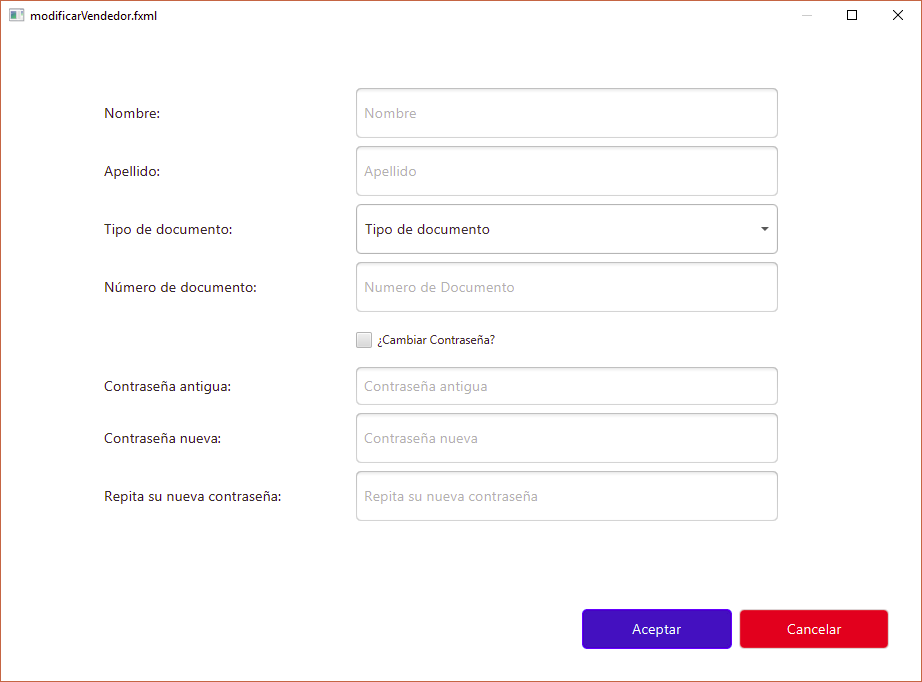
Vista donde se muestran todos los vendedores, y desde donde se los puede eliminar:



###### Vista alta vendedor

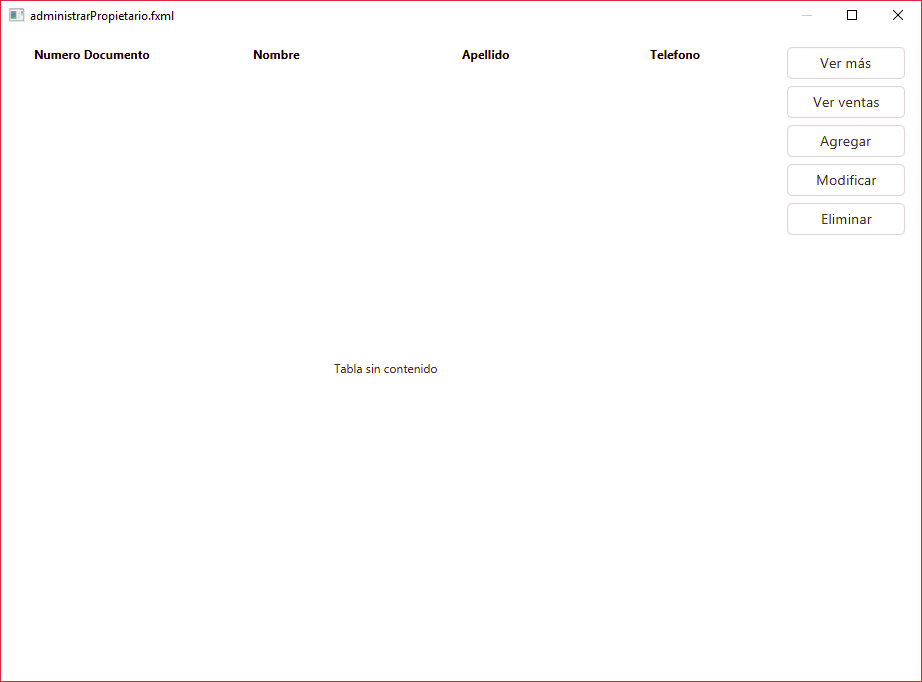


###### Vista modificar vendedor

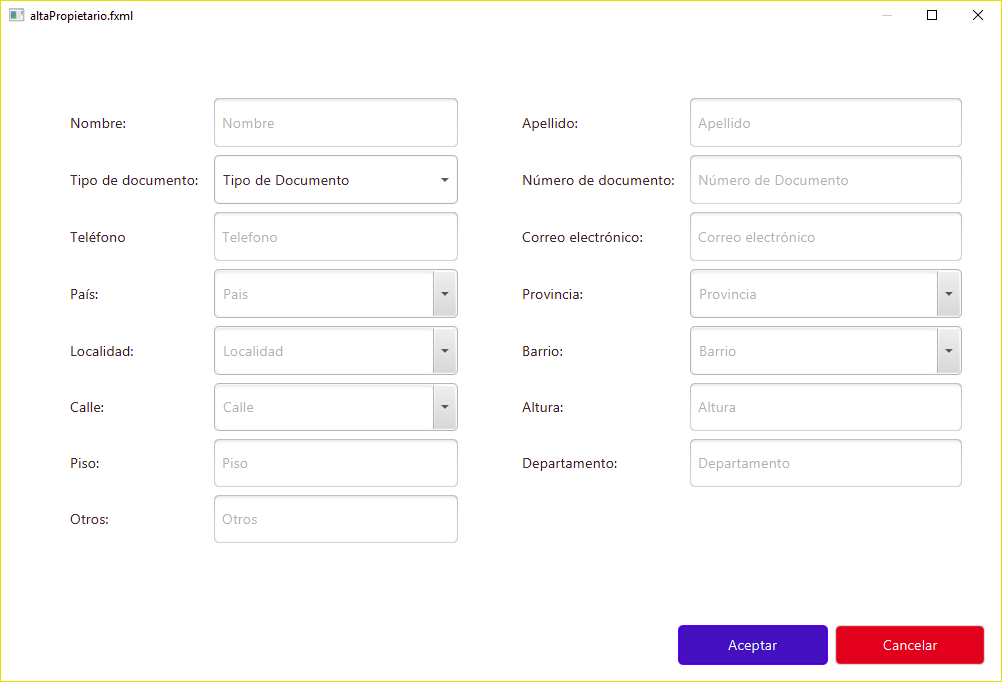


### Historia 2: ABM propietario

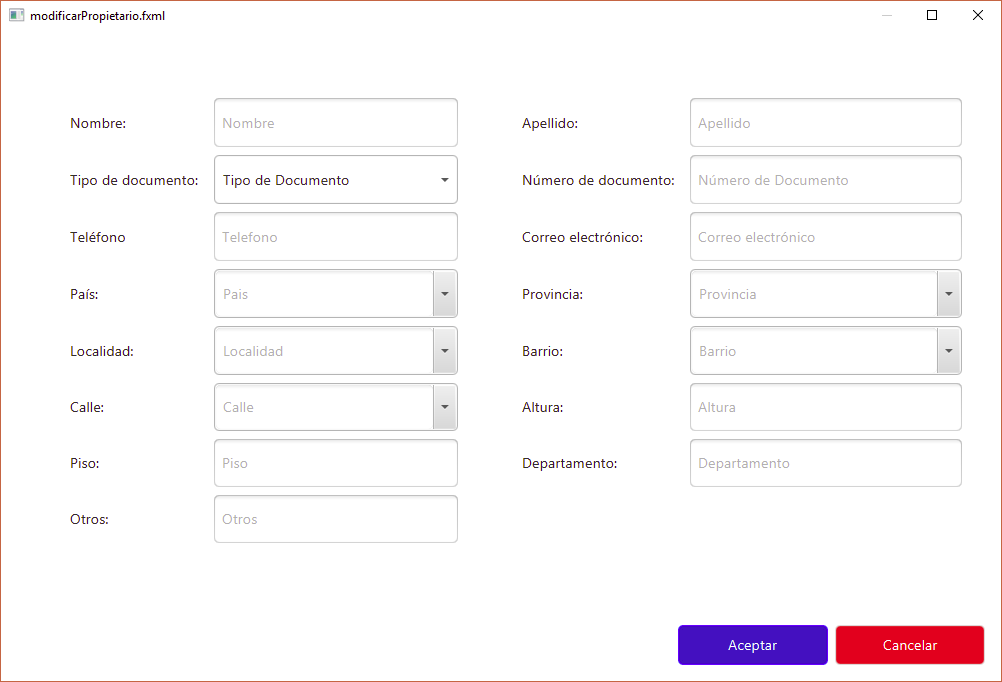
Vista donde se ven todos los propietarios, y desde donde se los puede eliminar:



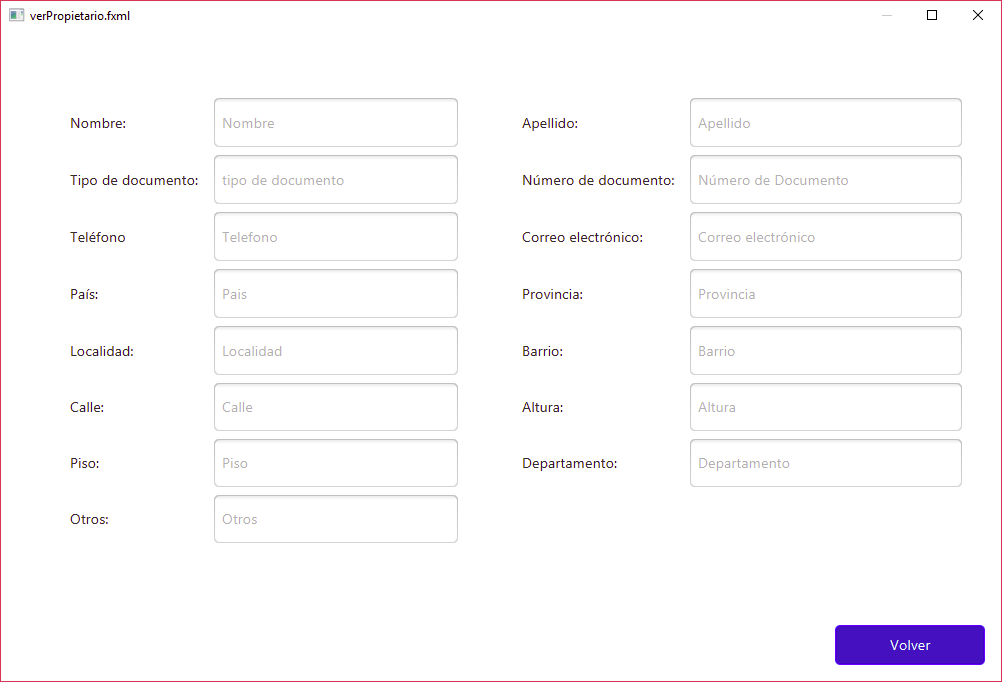
###### Vista alta propietario



###### Vista modificar propietario

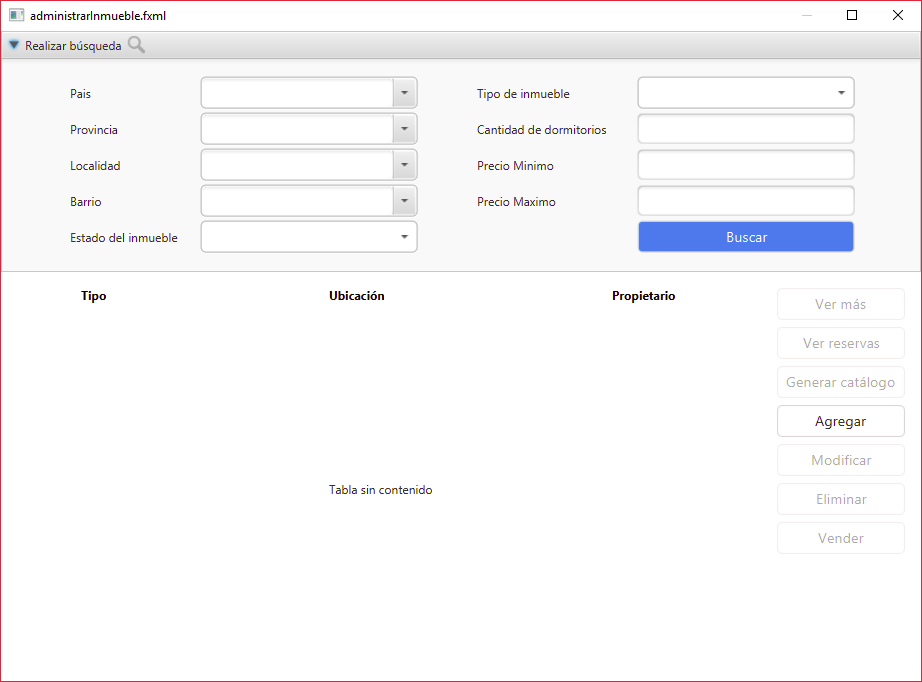


Vista donde se puede ver un propietario completo:

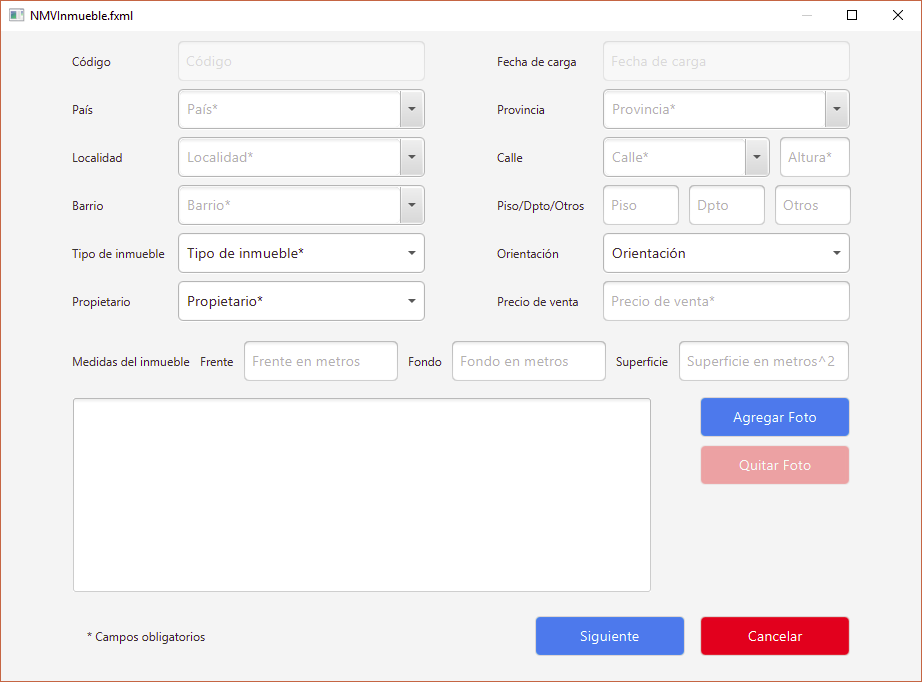


### Historia 3: ABM inmueble, Historia 4: Consulta inmueble

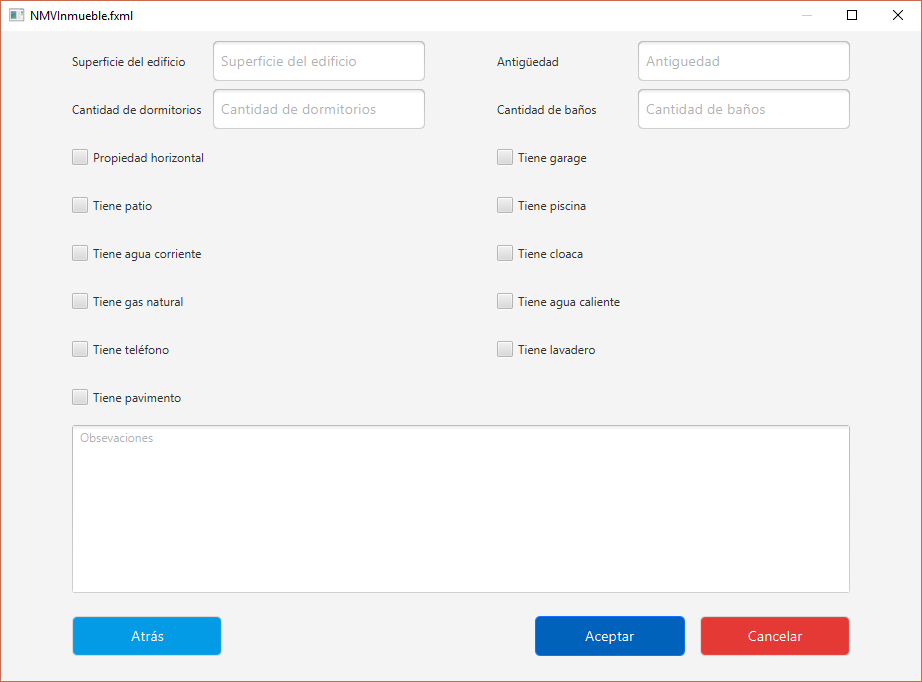
Vista donde se ven todos los inmuebles, o un subconjunto filtrado de ellos, y desde donde se los puede eliminar. También permite acceder a datos más detallados del inmueble con el botón “Ver más”, acceder a una vista que permite modificar los datos de un inmueble, acceder a una vista para cargar un inmueble nuevo, acceder a la funcionalidad de venta para vender uno de los inmuebles, acceder a las reservas de un inmueble, y acceder a la creación un catálogo con los inmuebles seleccionados.



Primera parte de la vista donde se puede crear, modificar, o ver un inmueble - (dependiendo de para qué se la llame, los campos serán editables o no):

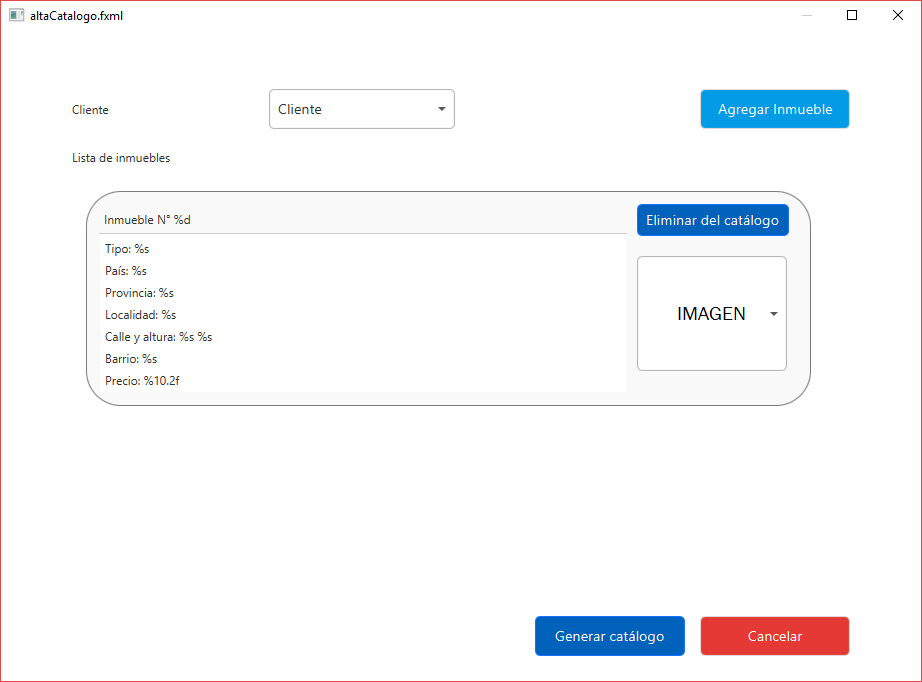


Segunda parte de la vista donde se puede crear, modificar, o ver un inmueble.



### Historia 5: Catálogo de Inmuebles

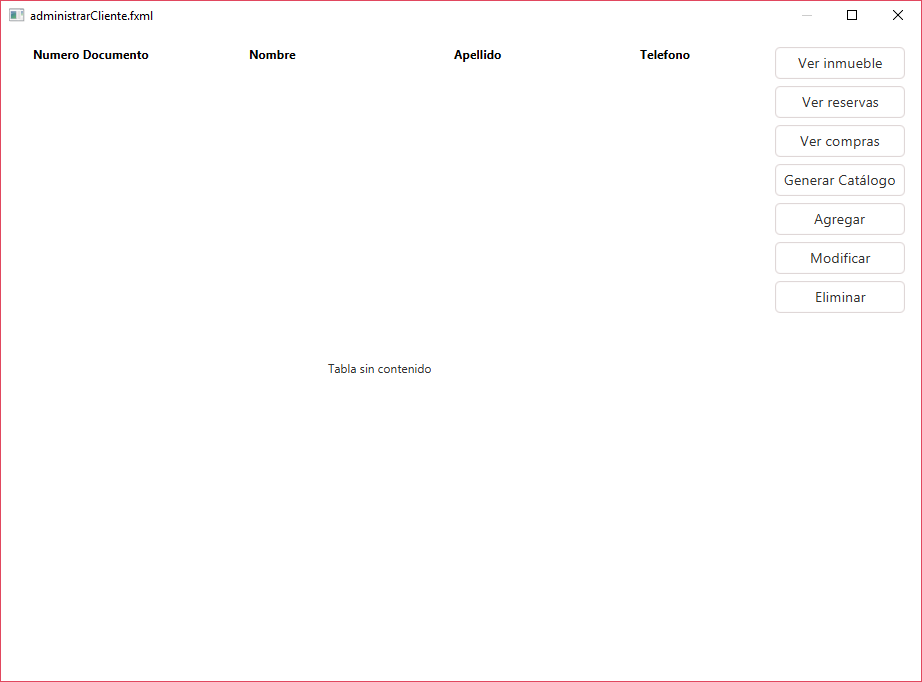
Vista que permite generar un catálogo, eligiendo el cliente para quien se genera, y una lista de inmuebles (en el spike, la lista tiene un solo elemento) pudiendo elegir la imagen de cada inmueble.



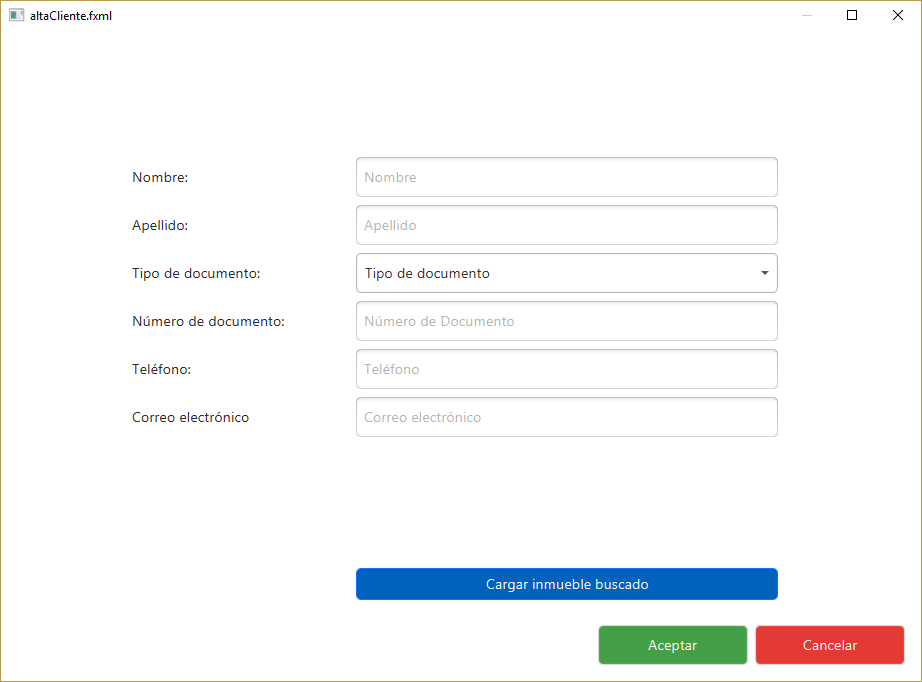
### Historia 6: ABM cliente

Vista donde se muestran todos los clientes, y da acceso a la creación de uno nuevo. Para cada cliente permite:

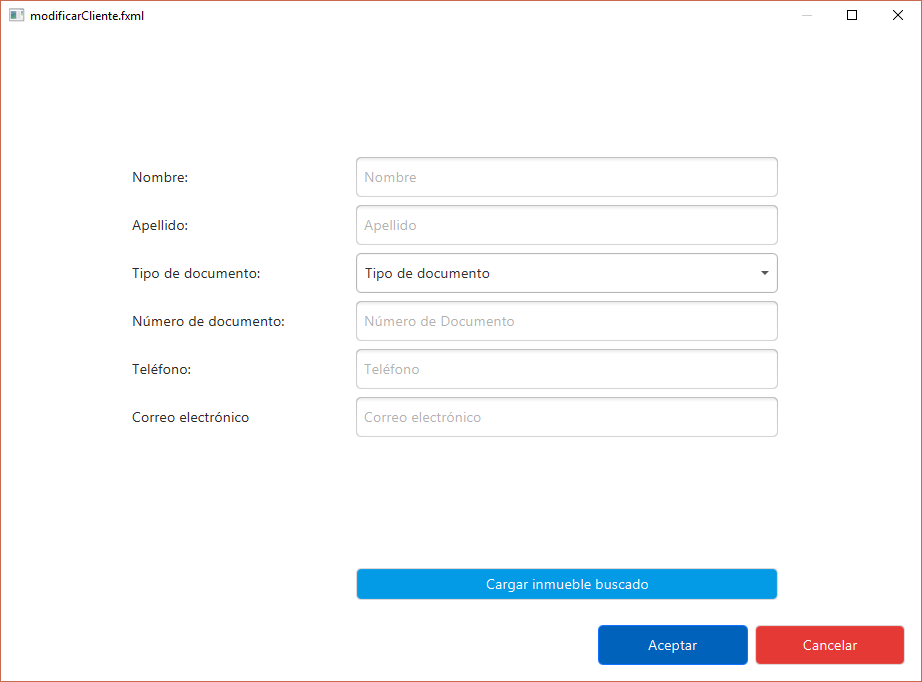
* acceder a la vista donde se ve su inmueble deseado,
* acceder a sus reservas,
* acceder a sus compras (ventas en las que el cliente fue el comprador),
* acceder a la vista para crear un catálogo personalizado,
* acceder a la vista que permite modificarlo,
* eliminarlo.



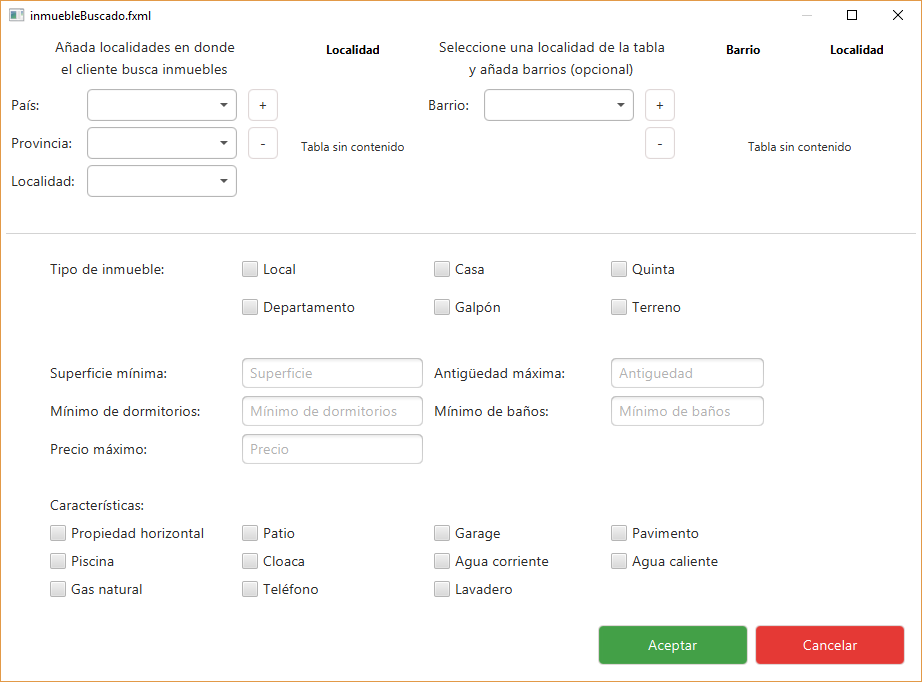
Vista que permite agregar un cliente



Vista que permite modificar los datos de un cliente existente:

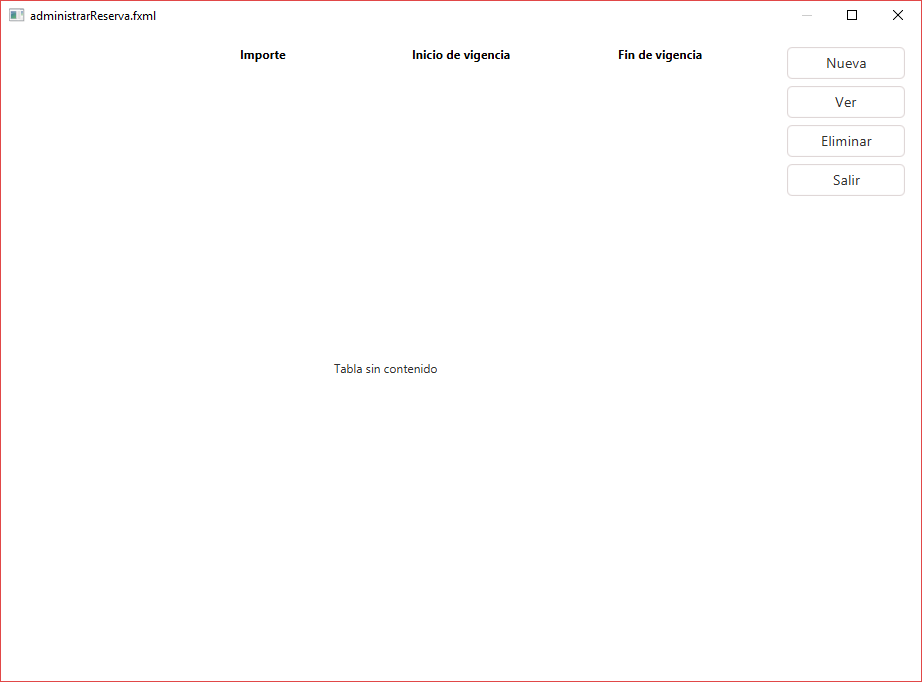


Vista que muestra las características del inmueble deseado por un cliente:

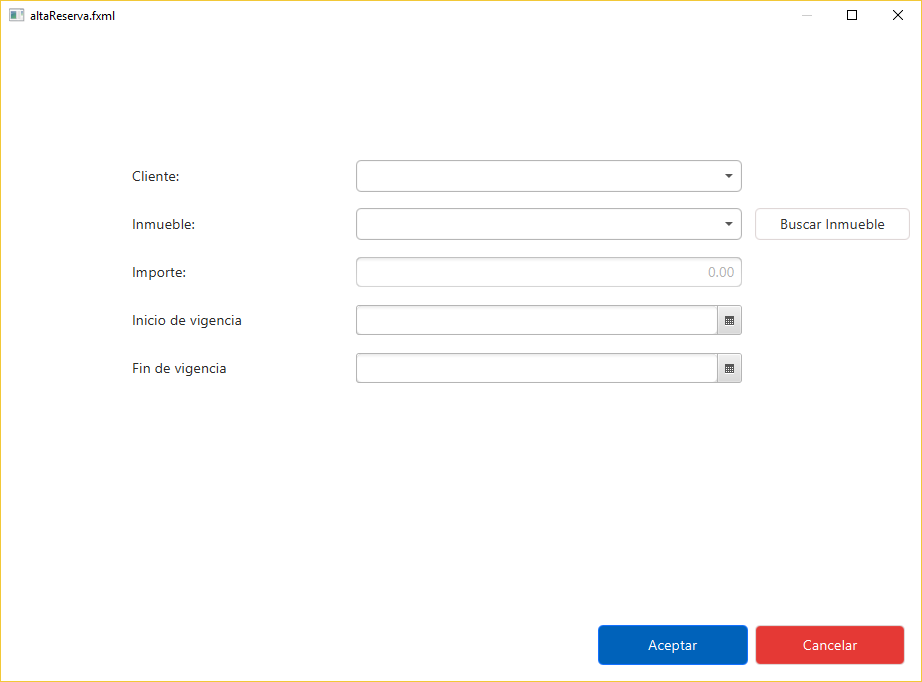


### Historia 7: Generar Reserva

Vista donde se ven todas las reservas, y desde donde se las puede eliminar. También brinda acceso a la creación de una nueva reserva.



Vista que permite crear una reserva:

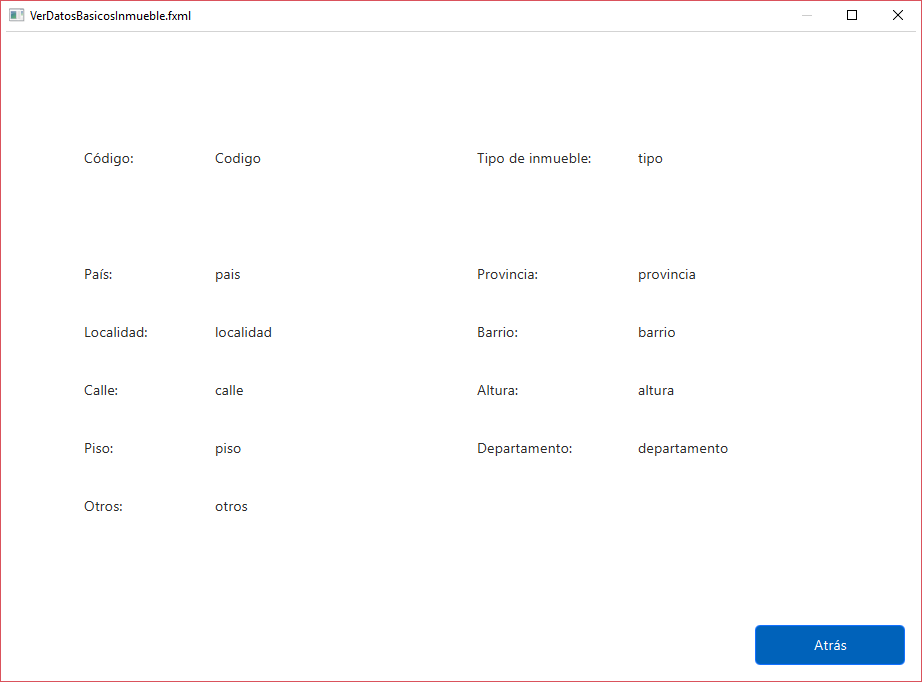


### Historia 8: Venta

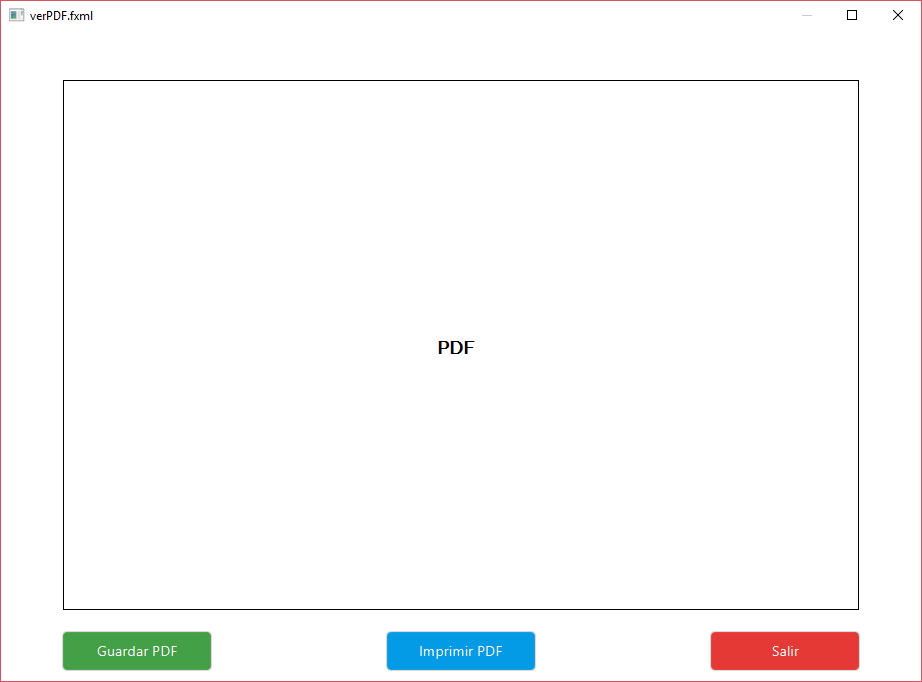
Vista que muestra las ventas asociadas a un cliente, a un propietario, o a un vendedor (dependiendo de cuál de los tres sea, su correspondiente columna no será visible). Permite acceder a ver los datos completos del inmueble asociado a una venta, y el PDF de la venta.



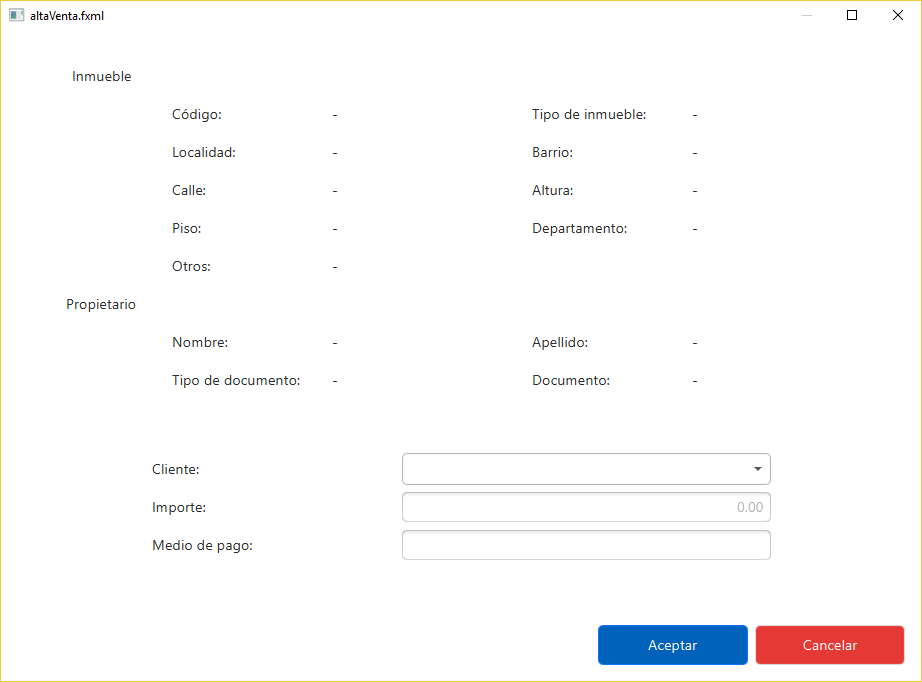
Vista donde se muestran los datos básicos del inmueble asociado a una venta:



Vista que permite visualizar un documento:

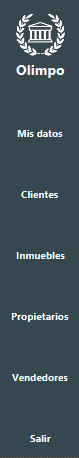


Esta vista permite crear una venta.

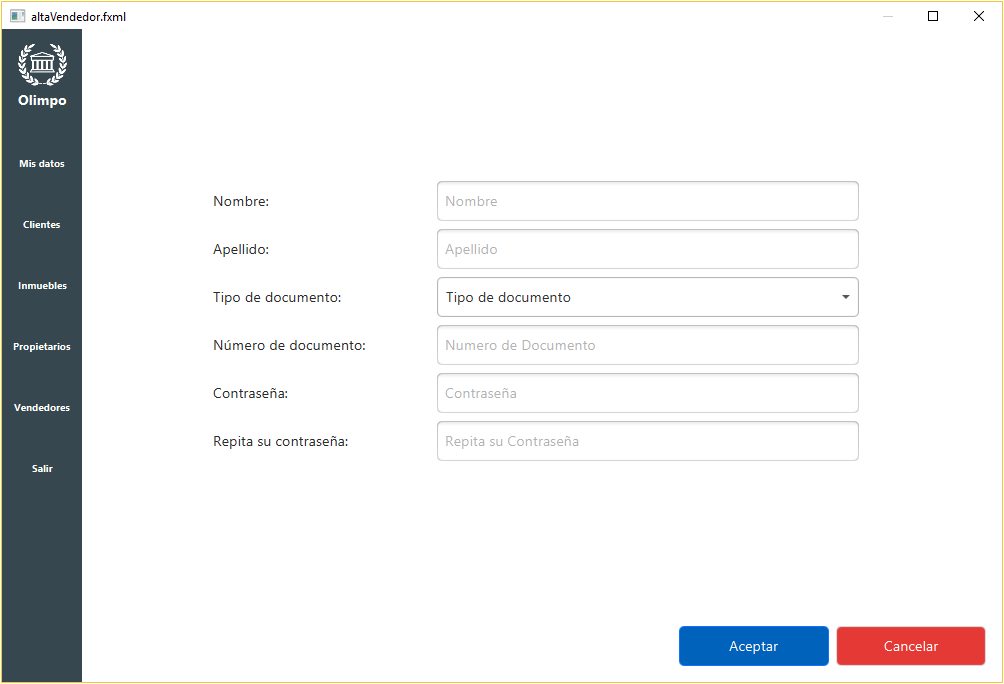


### TaskCard 4: Vista principal para la administración

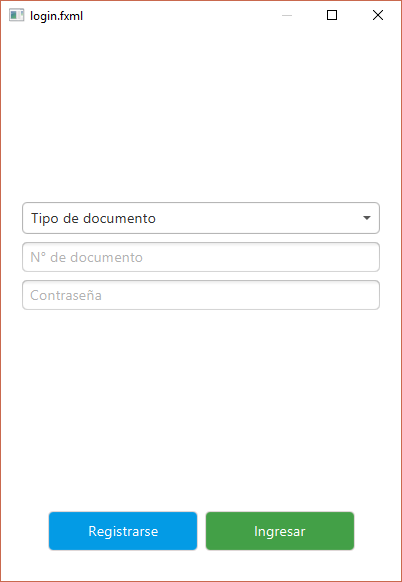
Así será el menú que permitirá acceder a las funciones del programa:



Aquí un ejemplo de cómo se vería con la función Alta Vendedor abierta:



### TaskCard 1: Vista y lógica del Login



# Definición de Velocidad y Alcance

* Total de historias de usuario = 17 Story Point.

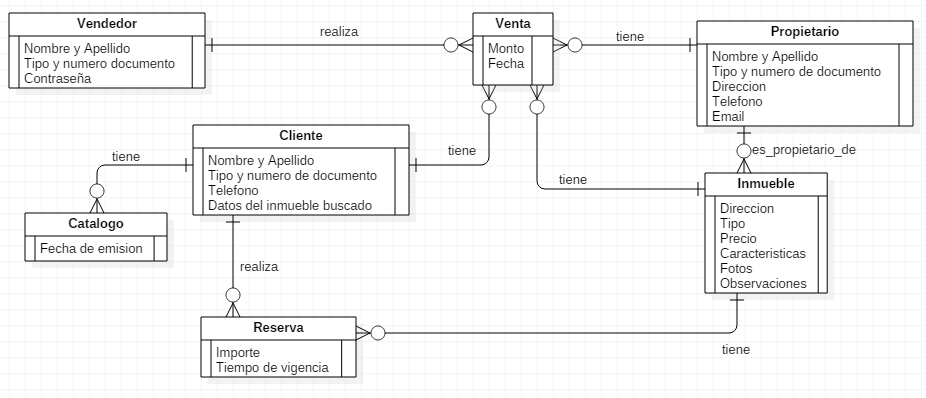
Para el primer release del sistema definimos con el cliente que no se va a implementar la historia de usuario 9.

* Alcance: 1, 2, 3, 4, 5, 6, 7 y 8.
* Total de historias de usuario para este release = 14 Story Point.
* Tiempo de iteración promedio = 3 semanas.

Estimamos que en 3 semanas una pareja de desarrollo puede realizar 3 Story Points. Esto nos da una velocidad de 9 Story Points por iteración, pero la definimos en 8 para tener un margen de maniobra en el caso de existir problemas en el desarrollo.

* Velocidad de desarrollo = 8 Story Point/iteración (para una iteración de 3 semanas).
* Cantidad de iteraciones = 14 [story point] / 8 [story point/iteración] = 2 iteraciones.

# Metáfora



Como metáfora elegimos un modelo Entidad-Relación, en el cual se modelaron las entidades principales del sistema, junto con sus atributos y relaciones con otras entidades.

Elegimos este modelo porque es relativamente fácil de entender por el cliente, y efectivo a la hora de comunicar al equipo de desarrollo la estructura de datos que tendrá el sistema.

# Fecha de Inicio del desarrollo

Fecha de inicio: martes 18 de octubre de 2016.

# Librerías, herramientas y tecnologías utilizadas

Utilizamos las siguientes herramientas:

* Java como lenguaje de programación.
* Javadoc para la creación de documentación.
* JavaFX el manejo de interfaces, es decir, la capa de presentación.
* SceneBuilder para la creación de interfaces.
* Eclipse o IntelliJ IDEA como IDEs.
* Git para el control de versiones.
* GitHub como servidor git remoto.
* JSpIRIT para detectar Code Smells para realizar el refactoring.
* PostgreSQL como base de datos.
* Maven para el manejo de librerías externas.
* StarUML para la creación de diagramas.
* Microsoft Word para la creación de documentos.

Utilizamos las siguientes librerías:

* Spring Framework para la inyección de dependencias y el manejo de transacciones a la base de datos.
* Hibernate como mapeador objeto-relacional, es decir, la capa de acceso a datos.
* PostgreSQL Driver para conectarse a la base de datos.
* JUnit para la realización de pruebas unitarias.
* Mockito para realizar los mocks de las pruebas.
* JUnitParams para la realización de varios casos de prueba en una sola prueba sólo cambiando los parámetros.
* Apache Commons Validator para realizar validaciones de datos.
* Itext para generar PDFs.
* PDFBox para imprimir PDFs.
* GoogleAPI, GoogleOAuthClient y javax.mail para el envío de correos electrónicos.
* JxBrowser para mostrar PDFs.

# Planificación Iteración 1

Fecha de inicio: martes 18 de octubre de 2016.

Fecha de finalización: martes 8 de noviembre de 2016.

## Tareas

### ABM Vendedor

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista y lógica del Login | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 08/10/2016 |  |  | Task Points | 2 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 1 |  | Número de TaskCard | | 1 |
|  |  |  |  |  |  |  |
| Descripción:  Se presenta una vista con las opciones para el inicio de sesión o para registrarse como un nuevo usuario del sistema, es decir, un vendedor.  La opción Registrar inicia la actividad de Alta Vendedor.  Para ingresar al sistema se requieren los siguientes datos:   * tipoDocumento (TipoDocumento) * numeroDocumento (String:30) * Contraseña (String)   Se deben realizar todas las validaciones antes de permitir el ingreso. De existir un error se presenta un cartel indicando lo sucedido.  La UI debe ser similar al spike realizado en la fase de exploración de la planificación | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 19/10/2016 | Definición de pruebas vista | | Terminar vista  Definición de pruebas lógica  Terminar lógica autenticar | | Definido cómo probar capa de interfaz gráfica | |
| 24/10/2016 | Terminada función ingresar de la vista | | Terminar función entrar a registrar de la vista  Definición de pruebas lógica autenticar  Terminar lógica autenticar | | Terminar función entrar a registrar de la vista:  Se debe terminar la ventana de registrar vendedor primero. | |
| 26/10/2016 | Definición de pruebas lógica autenticar | | Terminar función entrar a registrar de la vista  Terminar lógica autenticar | | Deberán ser refinadas al terminar de implementar la lógica autenticar. | |
| 03/11/2016 | Terminar función entrar a registrar de la vista. Terminar lógica autenticar | | Nada | |  | |

* Estimación 2 Task Points.
* Tiempo real 4 Task Points.
* Desvío 2 Task Points de subestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Entidad vendedor | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 1 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 1 |  | Número de TaskCard | | 2 |
|  |  |  |  |  |  |  |
| Descripción:  Implementar la clase Vendedor. La clase debe tener los siguientes atributos:   * nombre(string:30), * apellido(string:30), * tipoDocumento(TipoDocumento), * numeroDocumento(string:30), * id(Integer), * password(string:100), * salt(String), * Root(Boolean) y * estado(Estado).   Para cada atributo deben hacerse los métodos set y get.  Completar la entidad con las anotaciones necesarias para la persistencia con Hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas:  El atributo Root es true para el vendedor con privilegios de administrador en el sistema.  El atributo Estado debe representar el estado de la entidad (alta, baja). | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 18/10/2016 | Definición completa de la entidad y sus anotaciones | | Nada | |  | |

* Estimación 1 Task Points.
* Tiempo real 1 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista alta, modificar y baja Vendedor | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 09/10/2016 |  |  | Task Points | 3 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 1 |  | Número de TaskCard | | 3 |
|  |  |  |  |  |  |  |
| Descripción:  Se deberá presentar una vista donde se visualice todos los vendedores cargados en el sistema y se permita dar de alta, modificar o eliminar un vendedor.  Para dar de alta un vendedor, los campos a ingresar por el usuario son los siguientes:   * Nombre (string:30), * Apellido (string:30), * Tipo de documento (TipoDocumento), * Numero de documento (String:30), * Contraseña (String) y * Repetir contraseña (String).   Todos los datos son obligatorios. Si ocurre un error en la validación de uno o más campos, la interfaz deberá mostrar el error y explicar brevemente que ha sucedido.  La interfaz para modificar un vendedor es la misma que la interfaz de alta, y además se visualiza un checkBox para permitir al usuario cambiar o no la contraseña. Se pueden modificar todos los campos. Se deben mostrar los datos del vendedor en los campos correspondientes.  La UI para la baja de un vendedor es un cartel avisando que el vendedor va a ser eliminado, y presentará las opciones para aceptar o no la eliminación.  Las UI deben ser similares a los spikes realizados en la fase de exploración de la planificación. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 28/10/2016 | Vista alta vendedor | | Vista modificar, baja y administración | |  | |
| 31/10/2016 | Vista modificar y baja vendedor | | Vista administración y transiciones entre vistas | |  | |
| 02/11/2016 | Vista administración y transiciones entre vistas | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 3 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista principal para la administración | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 08/10/2016 |  |  | Task Points | 2 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 1 |  | Número de TaskCard | | 4 |
|  |  |  |  |  |  |  |
| Descripción:  Se deben mostrar las acciones que tiene permitido un vendedor después de ingresar al sistema:   * Ver mis datos. * Inmueble: alta, modificar, eliminar, consulta, generar reserva, vender. * Propietario: alta, modificar, eliminar. * Cliente: Alta, modificar, eliminar, ver catálogo de inmuebles * Vendedor: alta, modificar, eliminar. (sólo si es administrador)   Si una acción no está permitida, se debe ocultar el botón.  La UI debe ser similar al spike realizado en la fase de exploración de la planificación. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 02/11/2016 | Definida estructura de transición entre pantallas | | Vista y sus transiciones | |  | |
| 04/11/2016 | Vista y sus transiciones | | Nada | | No tiene pruebas porque solo muestra otras pantallas | |

* Estimación 2 Task Points.
* Tiempo real 2 Task Points.
* Desvío 0 Task Points de sobreestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica alta, modificación y baja vendedor | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 4 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 1 |  | Número de TaskCard | | 5 |
|  |  |  |  |  |  |  |
| Descripción:  Se recibe de la vista un objeto vendedor.  Para el alta se deberá validar que el tipo y número de documento no se repita con un vendedor ya registrado y el número de documento corresponda con el tipo de documento. El nombre y apellido se deberá validar que sean solo letras y tamaño máximo 30 caracteres.  Si todo es correcto deberá dar de alta el vendedor en la base de datos. Si algo no es correcto se lanza una excepción.  Para la modificación se permitirá cambiar cualquier campo y se deberán validar los campos de la misma forma. Si todo es correcto se deberá modificar el vendedor en la base de datos.  Para la baja, no se deben tener consideraciones especiales y se debe realizar la baja lógica del vendedor. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 19/10/2016 | Test y lógica de alta vendedor | | Test y lógica de modificar y de baja | |  | |
| 20/10/2016 | Test y lógica de modificar | | Lógica de baja | |  | |
| 21/10/2016 | Agregadas verificaciones de datos y casos de prueba | | Lógica de baja | |  | |
| 22/10/2016 | Lógica de baja | | Nada | |  | |

* Estimación 4 Task Points.
* Tiempo real 4 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Persistidor vendedor | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 3 |  |
|  |  |  |  |  |  |  |
| Número de Historia 1 | |  |  | Número de TaskCard | | 6 |
|  |  |  |  |  |  |  |
| Descripción:  Se debe hacer una interfaz con métodos que permitan:   * Guardar un vendedor (si resulta en error lanza excepción SaveUpdateException). * Modificar un vendedor (si resulta en error lanza excepción SaveUpdateException). * Obtener un vendedor por medio de su tipo y número de documento (si resulta en error lanza excepción SaveUpdateException). * Obtener todos los vendedores (si resulta en error lanza excepción SaveUpdateException).   Todas las excepciones mencionadas extienden de PersistenceException.  Se debe hacer una clase que implemente dicha interfaz mediante hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas  El método para obtener todos los vendedores debe buscar aquellos vendedores con el estado “Alta”. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 23/10/2016 | Definido persistidor | | Manejo de excepciones | |  | |
| 24/10/2016 | Manejo de excepciones | | Nada | | Completado en menos tiempo de lo estimado por baja complejidad | |

* Estimación 3 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Points de sobreestimación.

### ABM Propietario

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Entidad propietario | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 1 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 2 |  | Número de TaskCard | | 7 |
|  |  |  |  |  |  |  |
| Descripción  Implementar la clase Propietario. La clase debe tener los siguientes atributos:   * id(Integer), * nombre(string:30), * apellido(string:30), * tipoDocumento(TipoDocumento), * numeroDocumento(string:30), * dirección(Direccion), * teléfono(string:30), * email(string:30), * estado(Estado) e * inmuebles(ArrayList<Inmueble>).   Para cada atributo deben hacerse los métodos set y get.  Completar la entidad con las anotaciones necesarias para la persistencia con Hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas:  El atributo Estado debe representar el estado de la entidad (alta, baja). | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 20/10/2016 | atributos, setters y getters, equals, anotaciones JPA y NamedQueries | | Nada | |  | |

* Estimación 1 Task Points.
* Tiempo real 1 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista alta, modificar y baja Propietario | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 3 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 2 |  | Número de TaskCard | | 8 |
|  |  |  |  |  |  |  |
| Descripción:  Los campos a ingresar por el usuario son los siguientes:   * Nombre (string:30), * Apellido (string:30), * Tipo de documento (ComboBox<TipoDocumento>), * Número de documento (String:30), * Dirección:   + Calle (string:50),   + Número (String:30),   + Piso (String:30),   + Departamento (string:30),   + Otros (String:100)   + País (string:50),   + Provincia (string:50) y   + Localidad (string:50). * Teléfono (String:30) y * Email (string:30).   Los siguientes datos son obligatorios: nombre, apellido, tipo, número de documento, calle, número, país, provincia, localidad, barrio, teléfono.  Si ocurre un error en la validación de uno o más campos, la interfaz deberá mostrar el error y explicar brevemente que ha sucedido.  La interfaz para modificar un propietario es la misma que la interfaz de alta y se pueden modificar todos los campos. Se deben mostrar los datos del propietario en los campos correspondientes.  La UI para la baja de un propietario es un cartel avisando que el propietario va a ser eliminado y presentará las opciones para aceptar o no la eliminación.  Las UI deben ser similares a los spikes realizados en la fase de exploración de la planificación. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 25/10/2016 | Definido fxml y controlador alta | | Fxml y Controlador modificar y baja | |  | |
| 26/10/2016 | Definido fxml y Controlador modificar | | Fml y controlador de administrar propietarios | |  | |
| 27/10/2016 | Terminados fxml y controlador administrar con funcionalidad baja y transiciones a pantallas alta y modificar | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 3 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica alta, modificación y baja propietario | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 09/10/2016 |  |  | Task Points | 3 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 2 |  | Número de TaskCard | | 9 |
|  |  |  |  |  |  |  |
| Descripción:  Para el alta se recibe de la vista un objeto propietario y se debe validar que:   * Nombre sea un string de letras con un máximo 30 caracteres. * Apellido sea un string de letras con un máximo 30 caracteres. * Tipo de documento sea uno de los tipos definidos. * Número de documento sea numérico y corresponda con el tipo de documento. * Dirección:   + Calle sea un string de como máximo 30 caracteres alfanuméricos.   + Número sea String de como máximo 10 caracteres.   + Piso sea String de como máximo 10 caracteres.   + Departamento sea un string de como máximo 10 caracteres.   + País sea un string de como máximo 30 caracteres.   + Provincia sea un string de como máximo 30 caracteres.   + Localidad sea un string de como máximo 30 caracteres.   + Barrio sea un string de como máximo 50 caracteres. * Teléfono sea un String de como máximo 30 caracteres. * Email sea un string de hasta 30 caracteres con formato de correo.   Los siguientes datos son obligatorios: nombre, apellido, tipo, número de documento, calle, número, localidad, provincia, teléfono, barrio.  Si todo es correcto se da de alta al propietario en la base de datos, si no se devuelve una excepción  Para la modificación se permitirá cambiar cualquier campo, se deberán realizar las mismas validaciones y si todo es correcto se deberá modificar el propietario en la base de datos. Si algo no es correcto se deberá devolver una excepción.  Para la baja, no se deben tener consideraciones especiales y se debe realizar una baja lógica del propietario.  Se debe implementar un método que permita obtener todos los propietarios de la base de datos.  Debe implementarse métodos para la validación de los campos, pero éstos deben ser privados. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 21/10/2016 | Logica de Alta y modificacion | | Baja de un propietario, validar datos | | Se realizaron los casos de prueba antes de codificar | |
| 22/10/2016 | Validar datos | | Baja de un propietario | |  | |
| 02/11/2016 | Baja de un propietario | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 3 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Persistidor propietario | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 09/10/2016 |  |  | Task Points | 3 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 2 |  | Número de TaskCard | | 10 |
|  |  |  |  |  |  |  |
| Descripción  Se debe hacer una interfaz con métodos que permitan:   * Guardar un propietario (si resulta en error lanza excepción SaveUpdateException). * Modificar un propietario (si resulta en error lanza excepción SaveUpdateException). * Obtener un propietario (si resulta en error lanza excepción ConsultaException). * Obtener todos los propietarios (si resulta en error lanza excepción ConsultaException)   Todas las excepciones mencionadas extienden de PersistenceException.  Se debe hacer una clase que implemente dicha interfaz mediante hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas:  El método para obtener todos los propietarios debe buscar aquellos propietarios con el estado “Alta”. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 31/10/2016 | Guardar y modificar un propietario | | Obtener un propietario y listar todos los propietarios | |  | |
| 02/11/2016 | Obtener un propietario y listar todos los propietarios | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Points de sobreestimación.

### ABM Inmueble

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Entidad inmueble | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 1 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 3 |  | Número de TaskCard | | 11 |
|  |  |  |  |  |  |  |
| Descripción  Implementar la clase Inmueble. La clase debe tener los siguientes atributos:   * id(Integer), * fechaCarga(Date), * Propietario(Propietario), * tipo(TipoInmueble), * precio(Double), * orientacion(Orientacion), * frente(Double), * fondo(Double), * superficie(Double), * datosEdificio(DatosEdificio), * fotos(ArrayList<Imagen>), * Observaciones (String:300), * direccion(Direccion), * estado(Estado).   Implementar la clase HistorialInmueble. La clase debe tener los siguientes atributos:   * id(long), * fechaYHoraCambio(Date), * fechaCarga(Date), * propietario(Propietario), * tipo(TipoInmueble), * precio(Double), * orientacion(Orientacion), * frente(Double), * fondo(Double), * superficie(Double), * datosEdificio(DatosEdificio), * fotos(ArrayList<Imagen>), * Observaciones (String:300), * direccion(Direccion), * estado(Estado).   Implementar la clase DatosEdificio. La clase debe tener los siguientes atributos:   * id(Integer), * propiedadHorizontal(Boolean), * superficie(double), * antigüedad(Integer), * dormitorios(Integer), * baños(Integer), * garaje(Boolean), * patio(Boolean), * piscina(Boolean), * aguaCorriente(Boolean), * cloacas(Boolean), * gasNatural(Boolean), * aguaCaliente(Boolean), * teléfono(Boolean), * lavadero(Boolean), * pavimento(Boolean).   Para cada atributo deben hacerse los métodos set y get.  Completar la entidad con las anotaciones necesarias para la persistencia con Hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas  El atributo Estado es un enumerable con cuatro estados posibles, alta, baja, vendido, no vendido. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 20/10/2016 | Terminada Entidad Inmueble y DatosEdificio | | Terminar Entidad HistorialInmueble | | No tiene pruebas porque son solo getters y setters. | |
| 22/10/2016 | Terminada Entidad HistorialInmueble | | Nada | | No tiene pruebas porque son solo getters y setters. | |

* Estimación 1 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Point de subestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Clases de datos | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 1 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 3 |  | Número de TaskCard | | 12 |
|  |  |  |  |  |  |  |
| Descripción:  Implementar las siguientes clases de datos:  La clase Dirección que tiene los atributos:   * calle(Calle), * número(string:30), * barrio(Barrio), * piso(string:30), * departamento(string:30), * localidad(Localidad) y * otros(string:100).   La clase Barrio que tiene los atributos:   * id(Integer), * nombre(string:50) y * localidad(Localidad).   La clase Calle que tiene los atributos:   * id(Integer), * nombre(string:50) y * localidad(Localidad).   La clase Localidad que tiene los atributos:   * id(Integer), * nombre(string:50) y * provincia(Provincia).   La clase Provincia con los atributos:   * id(Integer), * nombre(string:50) y * pais(Pais).   La clase País con atributos:   * id(Integer) y * nombre(string:50).   La clase TipoDocumento tiene los atributos:   * id(Integer) y * tipo(enumerable: DNI, LC, LE, CÉDULA\_EXTRANJERA, PASAPORTE).   La clase TipoInmueble tiene los atributos:   * id(Integer) y * nombre(enumerable: L, C, D, T, Q, G).   La clase Orientacion tiene los atributos:   * id(Integer) y * orientación (enumerable: NORTE, SUR, ESTE, OESTE, NORESTE, NOROESTE, SURESTE, SUROESTE).   Para cada atributo deben hacerse los métodos set y get.  Completar las entidades con las anotaciones necesarias para la persistencia con Hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas:  Estas no son TODAS las clases necesarias. Pueden y se van a tener que hacer más. Indicar las clases agregadas en los comentarios y que no están listadas en esta Task Card. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 20/10/2016 | Atributos, getters y getters, equals | | Anotaciones | | Agregadas clase: Imagen, Archivo, Estado. | |
| 21/10/2016 | Anotaciones | | Nada | |  | |

* Estimación 1 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Points de subestimación.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista alta, modificar y baja inmueble | | | |  | | | Programador | | 3 | | |
|  |  | | |  |  | | |  | |  | |  |
| Fecha | 07/10/2016 | | |  |  | | | Task Points | | 3 | |  |
|  |  | | |  |  | | |  | |  | |  |
| Número de Historia | | | | 3 |  | | | Número de TaskCard | | | | 13 |
|  |  | | |  |  | | |  | |  | |  |
| Descripción:  Los campos a ingresar por el usuario son los siguientes:   * Propietario (Obligatorio), * País: por defecto Argentina (String:50) (obligatorio), * Provincia: por defecto Santa Fe (String:50) (obligatorio), * Localidad (String:50) (obligatorio), * Barrio (String:50) (obligatorio), * Calle (String:50), * número (String:50, numérico) (obligatorio), * Piso (String:30), * Departamento (String:30), * otros (String:100), * Tipo de inmueble (L/local-oficina, C/casa, D/departamento, T/terreno, Q/quinta, G/galpón) (obligatorio), * Precio de venta (Double, 2 cifras significativas) (obligatorio), * Orientación (norte, sur, este, oeste, noreste, noroeste, sureste, suroeste), * Medidas:   + Frente en metros (Double),   + Fondo en metros (Double),   + Superficie en m2 (Double). * Datos del edificio:   + Propiedad horizontal (Booleano),   + Superficie en m2 (Double),   + Antigüedad en años (Integer),   + Dormitorios (Integer),   + Baños (Integer),   + Garaje/cochera (Boolean),   + Patio (Boolean),   + Piscina (Boolean),   + Agua corriente (Boolean),   + Cloacas (Boolean),   + Gas natural (Boolean),   + Agua caliente (Boolean),   + Teléfono (Boolean),   + Lavadero (Boolean),   + Pavimento (Boolean). * Fotos (sin restricción de tamaño), * Observaciones (String:300).   Si ocurre un error en la validación de uno o más campos, la interfaz deberá mostrar el error y explicar brevemente que ha sucedido.  La interfaz para modificar un inmueble es la misma que la interfaz de alta, se pueden modificar todos los campos, excepto el código. Se deben mostrar los datos del inmueble en los campos correspondientes.  La UI para la baja de un inmueble es un cartel avisando que el inmueble va a ser eliminado, y presentará las opciones para aceptar o no la eliminación. | | | | | | | | | | | | |
|  | |  |  | | |  |  | |  | |  | |
| Notas:  La baja se realiza de manera lógica seteando el estado en Baja. | | | | | | | | | | | | |
|
|
| Fecha | | Realizado | | | | A realizar | | | Comentarios | | | |
| 05/11/2016 | | Empezados test de los controladores | | | | Terminar test de los controladores  Programar controladores | | |  | | | |
| 06/11/2016 | | Terminados test de los controladores.  Programados controladores. | | | | Nada | | | Fue rápido porque pudimos reutilizar código de otros controladores. | | | |

* Estimación 3 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Points de sobreestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica alta, modificación y baja inmueble | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 09/10/2016 |  |  | Task Points | 4 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 3 |  | Número de TaskCard | | 14 |
|  |  |  |  |  |  |  |
| Descripción:  Se recibe de la vista un objeto inmueble.  Para el alta se debe validar que:   * código del inmueble no se repita con otro inmueble (independientemente del estado) * fecha de carga sea una fecha bien formada. * propietario sea un propietario previamente cargado (obligatorio). * País: sea un string de como máximo 50 caracteres (obligatorio). * Provincia: sea un string de como máximo 50 caracteres (obligatorio). * Localidad: sea un string de como máximo 50 caracteres (obligatorio). * Calle sea un string de como máximo 50 caracteres alfanuméricos. * Número sea numérico. * Piso sea numérico. * Departamento sea un string de como máximo 30 caracteres. * Barrio sea un string de como máximo 50 caracteres. * Tipo de inmueble: sea del tipo: L/local-oficina, C/casa, D/departamento, T/terreno, Q/quinta, G/galpón (obligatorio); * Precio de venta sea numérico (obligatorio); * Orientación sea un string (norte, sur, este, oeste, noreste, noroeste, sureste, suroeste); * Medidas: Frente (numérico), fondo (numérico), superficie (numérico); * Datos del edificio: propiedad horizontal (booleano), superficie (numérico), antigüedad (numérico), dormitorios (numérico), baños (numérico), garaje/cochera (booleano), patio (booleano), piscina (booleano), agua corriente (booleano), cloacas (booleano), gas natural (booleano), agua caliente (booleano), teléfono (numérico), lavadero (booleano), pavimento (booleano); * Observaciones, string de 300 caracteres, como máximo;   Cada inmueble debe tener un estado. Luego del alta el estado pasa a ser “Alta”.  Si todo es correcto se da de alta al inmueble en la base de datos y se le asigna el estado “Alta”. Si ocurrieron errores de validación se lanza una excepción.  Para la modificación se pueden modificar todos los campos (excepto el código) y se deberá validar de la misma manera. Si todo es correcto se guarda un historial de cambio y se modifica el inmueble en la base de datos. De lo contrario, se lanza una excepción.  Para la baja no se deben tener consideraciones especiales y se debe realizar la baja lógica del inmueble seteando el estado en “Baja” | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 03/11/2016 | Pruebas ABMC lógica inmueble. | | Implementación ABMC inmueble. | |  | |
| 04/11/2016 | Implementación ABMC inmueble. | | Nada | |  | |

* Estimación 4 Task Points.
* Tiempo real 2 Task Points.
* Desvío 2 Task Points de sobreestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Persistidor inmueble | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 09/10/2016 |  |  | Task Points | 3 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 3 |  | Número de TaskCard | | 15 |
|  |  |  |  |  |  |  |
| Descripción:  Se debe hacer una interfaz con métodos que permitan:   * Guardar un inmueble (si resulta en error lanza excepción SaveUpdateException). * Modificar un inmueble (si resulta en error lanza excepción SaveUpdateException). * Obtener un inmueble por id (si resulta en error lanza excepción ConsultaException). * Obtener todos los inmuebles (si resulta en error lanza excepción ConsultaException)   Todas las excepciones mencionadas extienden de PersistenceException.  Se debe hacer una clase que implemente dicha interfaz mediante hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas:  El método para obtener todos los inmuebles debe buscar aquellos inmuebles con el estado “Alta”. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 01/11/2016 | Guardar, modificar, obtener un inmueble y obtener inmuebles | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 1 Task Points.
* Desvío 2 Task Points de sobreestimación.

### ABM Cliente

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Entidad cliente | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 1 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 6 |  | Número de TaskCard | | 16 |
|  |  |  |  |  |  |  |
| Descripción  Implementar la clase Cliente. La clase debe tener los siguientes atributos:   * Id (Integer), * nombre (string:30), * apellido (string:30), * tipoDocumento (TipoDocumento), * numeroDocumento (string:30), * teléfono (string:30), * estado (Estado), * inmuebleBuscado (InmuebleBuscado)   Implementar la clase InmuebleBuscado. La clase debe tener los siguientes atributos:   * cliente(Cliente), * Tipo de inmueble [L/local-oficina, C/casa, D/departamento, T/terreno, Q/quinta, G/galpón] (obligatorio) * Localidad (String:50) * Barrios (Lista de String:50) * Características:   + Propiedad horizontal (Booleano)   + Superficie en m2 (Decimal)   + Antigüedad en años (Entero)   + Dormitorios mínimos (Entero)   + Baños mínimos (Entero)   + Garaje/cochera (Booleano)   + Patio (Booleano)   + Piscina (Booleano)   + Agua corriente (Booleano)   + Cloacas (Booleano)   + Gas natural (Booleano)   + Agua caliente (Booleano)   + Teléfono (Booleano)   + Lavadero (Booleano)   + Pavimento (Booleano)   Para cada atributo deben hacerse los métodos set y get.  Completar la entidad con las anotaciones necesarias para la persistencia con Hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas:  El atributo Estado debe representar el estado de la entidad (alta, baja). | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 18/10/2016 | Terminada | | Nada | | No se hacen pruebas porque son solo setters y getters | |

* Estimación 1 Task Points.
* Tiempo real 1 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista alta, modificar y baja cliente | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 3 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 6 |  | Número de TaskCard | | 17 |
|  |  |  |  |  |  |  |
| Descripción:  Se presenta una vista con los campos a ingresar por el usuario para realizar el alta de un cliente:   * Nombre (String:30) (obligatorio) * Apellido (String:30) (obligatorio) * Tipo de documento [DNI, LC, LE, Pasaporte, Cédula Extranjera] (obligatorio) * Número de documento (String:30) (obligatorio) * Teléfono (String:30) * Datos del inmueble buscado:   + Tipo de inmueble [L/local-oficina, C/casa, D/departamento, T/terreno, Q/quinta, G/galpón] (obligatorio)   + Localidad (String:50)   + Barrios (Deben ser de la misma localidad seleccionada o estar vacío si no se seleccionó ninguna) (Lista de String:50)   + Características:     - Propiedad horizontal (Booleano)     - Superficie en m2 (Decimal)     - Antigüedad en años (Entero)     - Dormitorios mínimos (Entero)     - Baños mínimos (Entero)     - Garaje/cochera (Booleano)     - Patio (Booleano)     - Piscina (Booleano)     - Agua corriente (Booleano)     - Cloacas (Booleano)     - Gas natural (Booleano)     - Agua caliente (Booleano)     - Teléfono (Booleano)     - Lavadero (Booleano)     - Pavimento (Booleano) * Monto disponible.   Si ocurre un error en la validación de uno o más campos, la interfaz deberá mostrar el error y explicar brevemente que ha sucedido.  La interfaz para modificar un cliente es la misma que la interfaz de alta, se pueden modificar todos los campos. Se deben mostrar los datos del cliente a modificar en los campos correspondientes.  La UI para la baja de un cliente es un cartel avisando que el cliente va a ser eliminado, y presentará las opciones para aceptar o no la eliminación.  Las UI deben ser similares a los spikes realizados en la fase de exploración de la planificación. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 28/10/2016 | Definido fxml y controladores alta | | Fxml y controladores Modificar, administrar, baja, inmueble buscado | |  | |
| 29/10/2016 | Definido fxml y controlador modificar | | Fxml y controladores administrar-baja e inmueble buscado | |  | |
| 30/10/2016 | Definido fxml y controlador administrar clientes con funcionalidad baja | | Fxml y controlador de cargar inmueble buscado | |  | |
| 31/10/2016 | Terminado fxml y controlador de cargar inmueble buscado | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 4 Task Points.
* Desvío 1 Task Point de subestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica alta, modificación y baja cliente | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 07/10/2016 |  |  | Task Points | 4 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 6 |  | Número de TaskCard | | 18 |
|  |  |  |  |  |  |  |
| Descripción:  Se recibe de la vista un objeto cliente.  Se deberá validar que:   * Nombre sea un string de letras con un máximo 30 caracteres. * Apellido sea un string de letras con un máximo 30 caracteres. * Tipo de documento sea uno de los tipos definidos. * Número de documento sea String de números con un máximo de 30 caracteres y corresponda con el tipo de documento. Debe ser único. * Teléfono sea String de números con un máximo 30 caracteres. * Validaciones para los datos del inmueble buscado.   Son obligatorios nombre, apellido, tipo y número de documento.  Si todo es correcto se da de alta el cliente en la base de datos. Si ocurre un error se lanza una excepción.  Se puede modificar todos los datos y se deben validar de la misma manera. Si todo es correcto se modifica al cliente en la base de datos. Si ocurre un error se lanza una excepción.  Para la baja no se deben tener consideraciones especiales y se debe realizar la baja del propietario de manera lógica cambiando el estado a “Baja”. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 01/11/2016 | Test y lógica de alta | | Tests y lógica de modificar y de baja | |  | |
| 02/11/2016 | Terminado modificar y baja | | Nada | |  | |

* Estimación 4 Task Points.
* Tiempo real 2 Task Points.
* Desvío 2 Task Point de sobreestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Persistidor cliente | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 10/10/2016 |  |  | Task Points | 3 |  |
|  |  |  |  |  |  |  |
| Número de Historia | | 6 |  | Número de TaskCard | | 19 |
|  |  |  |  |  |  |  |
| Descripción  Se debe hacer una interfaz con métodos que permitan:   * Guardar un cliente (si resulta en error lanza excepción SaveUpdateException). * Modificar un cliente (si resulta en error lanza excepción SaveUpdateException). * Obtener un cliente por tipo y número de documento (si resulta en error lanza excepción SaveUpdateException). * Obtener todos los clientes (si resulta en error lanza excepción SaveUpdateException).   Todas las excepciones mencionadas extienden de PersistenceException.  Se debe hacer una clase que implemente dicha interfaz mediante hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas  El método para obtener todos los clientes debe buscar aquellos clientes con el estado “Alta”. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 30/11/2016 | Terminadas pruebas persistidor | | Programar persistidor | |  | |
| 01/11/2016 | Terminado de programar persistidor | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Points de sobreestimación.

## Estimación de Esfuerzo

Sumatoria de desvíos = 5 Task Points de sobreestimación (15 horas).

## Conclusiones parciales de lo estimado

Consideramos que el desvío es por demás de aceptable teniendo en cuenta que fue la primera estimación en el proyecto. Esto se debió a que la mayoría del equipo de desarrollo tenía conocimientos de las herramientas de desarrollo utilizadas, ya que habían trabajado en proyectos que utilizaban estas herramientas, haciendo que el tiempo de resolución de problemas frecuentes tienda a cero.

Gran parte del tiempo se disminuyó porque las tareas se paralelizaron correctamente. Como la arquitectura del sistema elegida fue Model-View-Controller, la mayoría de las historias se dividieron en tres task, una por cada capa; entonces, cuando una pareja del equipo tenía conocimientos frescos sobre una tecnología correspondiente a una capa del modelo y tenía una task correspondiente a esa capa, esa pareja debía terminar la task antes que el resto. Luego, el resto de las parejas que tenían una task similar, podían reutilizar o adaptar las pruebas y el código a su task (o al menos mirar cómo se implementa una cuestión en particular).

También se reutilizó código desarrollado por miembros del equipo de desarrollo para otros proyectos que realizan funcionalidades requeridas en éste; por ejemplo, clases de datos (país, provincia, localidad, dirección, estado, documento, entre otras), validadores de formato, persistidores, gestor de contraseña, entre otros.

El desarrollo de las pruebas antes que la codificación fue muy engorroso al principio. Una vez que se codificaron algunas pruebas, éstas sirvieron de guía para tasks que requerían pruebas similares.

Por lo tanto, para la siguiente iteración se mantuvo la estimación para tasks similares.

# Planificación Iteración 2

Fecha de inicio: miércoles 30 de noviembre de 2016.

Fecha de finalización: miércoles 14 de diciembre de 2016.

## Tareas

### Consulta inmueble

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista de consulta inmueble | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 27/11/2016 |  |  | Task Points |  | 2 |
|  |  |  |  |  |  |  |
| Número de Historia | | 4 |  | Número de TaskCard | | 20 |
|  |  |  |  |  |  |  |
| Descripción:  Se deberá presentar una vista donde permita consultar un inmueble ingresando:   * País ((ComboBox <País>), * provincia ((ComboBox<Provincia>), * localidad ((ComboBox<Localidad>), * barrio ((ComboBox<Barrio >), * tipo ((ComboBox<TipoInmueble>), * cantidad de dormitorios (Entero > 0), * precio máximo (Double), * precio mínimo (Double) y * estado del inmuble (ComboBox<EstadoInmueble>).   Si el rango de precios está al revés, debe invertirse y luego proseguir la consulta.  Si ocurre un error en la validación de uno o más campos, la interfaz deberá mostrar el error y explicar brevemente que ha sucedido.  Si todo es correcto se debe pasar a la lógica un objeto filtro con los campos completados por el usuario. Se debe mostrar cualquier error recibido. Si todo es correcto se deben presentar los inmuebles obtenidos como resultado de la consulta en forma de lista mostrando el tipo, ubicación (país provincia, localidad, barrio, calle y número), nombre y apellido del propietario.  Luego de la consulta se permitirá seleccionar algún resultado y elegir alguna de las siguientes funciones:   * Baja Inmueble. * Modificación Inmueble. * Catálogo de inmueble (Genera un catálogo con los inmuebles seleccionados, uno o más). * Generar Reserva. * Venta. * Ver reservas. * Ver más (muestra la pantalla de modificar inmueble con los campos correspondientes rellenados y deshabilitados).   Si se selecciona alguna de esas funciones, se debe invocar a la acción correspondiente.  Si se selecciona más de un inmueble, solo se permite la opción “Generar catálogo de inmueble”. | | | | | | |
|  |  |  |  |  |  |  |
| Notas:  Los campos de dirección con ComboBox permitirán ingresar texto y el criterio de la búsqueda será “contiene”. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 30/11/2016 | Agregada sección de búsqueda al fxml de administrarInmueble y atributos asociados en su controlador | | Lógica de control y de transiciones del controlador | | Se agregó la búsqueda a una pantalla existente | |
| 01/12/2016 | Lógica de control y de transiciones del controlador | | Nada | |  | |

* Estimación 2 Task Points.
* Tiempo real 2 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica y persistidor de consulta inmueble | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 27/11/2016 |  |  | Task Points |  | 3 |
|  |  |  |  |  |  |  |
| Número de Historia | | 4 |  | Número de TaskCard | | 21 |
|  |  |  |  |  |  |  |
| Descripción:  Se recibe de la vista un objeto filtro con los datos necesarios para realizar una consulta a la base de datos (País, Provincia, localidad, barrio, TipoInmueble, cantidad de dormitorios, rango de precios y estado).  Se realiza la consulta en la base de datos con el filtro recibido. Los campos buscados deben contener el string ingresado en el filtro. Si ocurre algún error durante la consulta se debe devolver una excepción ConsultaException que extienda de PersistenceException.  Si no ha ocurrido ningún error se debe devolver un arreglo con los inmuebles que arrojó como resultado la consulta.  La realización de la consulta en la base de datos se debe implementar mediante hibernate y se deben completar las entidades con las anotaciones necesarias. | | | | | | |
|  |  |  |  |  |  |  |
| Notas:  Los inmuebles que se obtienen de la base de datos deben tener el estado “Alta”. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 03/12/2016 | Lógica y persistidor consulta inmueble | | Filtro consulta inmuble y test querys filtro consulta inmueble. | |  | |
| 04/12/2016 | Test querys filtro consulta inmueble | | Filtro consulta inmueble | | En los test se prueba sólo que las querys sean válidas para hibernate. | |
| 05/12/2016 | Filtro consulta inmueble | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 3 Task Points.
* Desvío 0 Task Points de desvío.

### Catálogo de inmuebles

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista Alta Catálogo Inmueble | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 27/11/2016 |  |  | Task Points |  | 3 |
|  |  |  |  |  |  |  |
| Número de Historia | | 5 |  | Número de TaskCard | | 22 |
|  |  |  |  |  |  |  |
| Descripción:  La carga de un catálogo es invocada desde la administración de clientes; al seleccionar un cliente, se habilita la opción de generar un catálogo para el mismo. También puede ser invocado desde la vista administración de inmuebles, al seleccionar uno o varios inmuebles y presionar la opción de generar un catálogo para los mismos.  Se deberá presentar una vista donde permita crear un nuevo catálogo seleccionando los inmuebles que se incluirán en el mismo. Para realizar esta operación, se presenta un listado (inicialmente vacío si se llamó desde la administración de clientes) con los inmuebles que serán parte del catálogo, junto con las opciones de “agregar nuevo inmueble”, “eliminar del catálogo” y “cambiar foto”.  Cuando se presiona “Agregar nuevo inmueble”, se presenta la vista correspondiente a consultar inmuebles, sólo con la opción de “agregar”. Se permitirá agregar más de un inmueble por consulta.  En la vista se visualizarán los inmuebles a incluir en el catálogo junto con la foto que va a ser incluida en el mismo (por defecto la foto es seleccionada aleatoriamente entre las asociadas al inmueble). La foto puede ser cambiada; Cuando se selecciona esta opción, se presenta una vista que muestre las fotos asociadas al inmueble, permitiendo seleccionar solo una foto. Si no hay ninguna foto en el inmueble, el campo de foto no se mostrará.  Cuando la lista de inmuebles no está vacía, se habilita la opción de “Generar catálogo”. Esta operación debe invocar a la generación del catálogo de la lógica. Si el catálogo es generado exitosamente, se visualiza automáticamente en otra vista, con las opciones para imprimir y/o guardar.  Si ocurre un error en la generación del catálogo se debe visualizar con un cartel describiendo lo que sucedió. | | | | | | |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 11/12/2016 | Test vista alta catálogo. | | Vista alta catálogo | |  | |
| 12/12/2016 | Vista alta catálogo | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Points de subestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica alta catálogo de inmueble y generar PDF | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 26/11/2016 |  |  | Task Points |  | 3 |
|  |  |  |  |  |  |  |
| Número de Historia | | 5 |  | Número de TaskCard | | 23 |
|  |  |  |  |  |  |  |
| Descripción:  Se recibe de la vista un objeto CatalogoVista, que consta de un cliente, una lista de inmuebles y un Map de inmueble con foto.  Para el alta se deberá validar que los inmuebles no se repitan y que haya un cliente asociado al catálogo. También que los siguientes datos de los inmuebles deben existir: código inmueble, tipo de inmueble, localidad, dirección, barrio, precio.  Si todo es correcto se deberá generar un archivo PDF con los datos del catálogo: cliente, foto del inmueble, datos del inmueble (código inmueble, tipo de inmueble, país, provincia, localidad, dirección, barrio, precio, cantidad de dormitorios, baños, garaje, patio, superficie terreno, superficie edificada). Si los siguientes datos no existen, no deben mostrarse: cantidad de dormitorios, baños, garaje, patio, superficie terreno, superficie edificada.  Si algo no es correcto se devuelve un error que explique la situación. | | | | | | |
|  |  |  |  |  |  |  |
| Notas  La generación del PDF se debe hacer en una clase aparte encargada de esto. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 12/12/2016 | Lógica alta catálogo y manejo de errores | | Definir estructura del PDF y lógica para generar PDF | |  | |
| 13/12/2016 | Hecho generar PDF | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Point de sobreestimación.

### Generar reserva

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista alta, baja y listar reserva | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 26/11/2016 |  |  | Task Points |  | 3 |
|  |  |  |  |  |  |  |
| Número de Historia | | 7 |  | Número de TaskCard | | 24 |
|  |  |  |  |  |  |  |
| Descripción:  En la vista de alta de reserva, los campos a ingresar por el usuario son los siguientes:   * Cliente (ComboBox<Cliente>) * Inmueble (ComboBox<Inmueble>) * Importe (Decimal, 2 dígitos) * Tiempo de vigencia:   + Fecha inicio de la reserva (Date)   + Fecha fin de la reserva (Date)   Si el rango de fechas está al revés, debe invertirse y luego proseguir al alta.  Para seleccionar un inmueble, debe listarse en un comboBox los inmuebles de la base de datos.  Todos los campos son obligatorios.  Si ocurre un error en la validación de uno o más campos, la interfaz deberá mostrar el error y explicar brevemente que ha sucedido.  La vista de listar reserva puede accederse al seleccionar un inmueble o un cliente. Esta vista lista todas las reservas del cliente o inmueble seleccionado y de esta se puede proceder a ver el PDF de una reserva seleccionada, dar de baja la reserva seleccionada o a crear una nueva reserva para el inmueble o el cliente seleccionado. Al hacer esto último, se carga el campo correspondiente y se lo hace no editable. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 07/12/2016 | Vista de listar reservas | | Vista alta y baja.  Transiciones desde listar inmueble y listar cliente. | |  | |
| 08/12/2016 | Vista alta y baja reserva | | Transiciones desde listar inmueble y listar cliente | |  | |
| 09/12/2016 | Transiciones desde listar inmueble y listar cliente | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 3 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica alta, baja reserva y generar PDF | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 28/11/2016 |  |  | Task Points |  | 3 |
|  |  |  |  |  |  |  |
| Número de Historia | | 7 |  | Número de TaskCard | | 25 |
|  |  |  |  |  |  |  |
| Descripción:  Se recibe de la vista un objeto Reserva con un cliente, un inmueble, un importe y un rango de vigencia.  Para el alta se deberá validar que haya un cliente y un inmueble asociados al catálogo y que estos tengan los datos necesarios para generar el PDF. También que el importe sea un número decimal válido mayor que 0 y que el rango de vigencia tenga una fecha inicial mayor a hoy, y que tenga una fecha de inicio anterior a la de finalización.  Si todo es correcto se deberá generar un archivo PDF con los datos de la reserva. Luego se genera una Reserva con dicho PDF, el cliente, el inmueble y el rango de fechas pasado y se dará de alta en la base de datos. También se deberá enviar un correo al mail del cliente con el archivo PDF como adjunto en otro hilo.  Si algo no es correcto se devuelve un error que explique la situación.  Para la baja, no se deben tener consideraciones especiales y se debe realizar una baja lógica de la reserva. Si algo no es correcto se devuelve un error que explique la situación. | | | | | | |
|  |  |  |  |  |  |  |
| Notas:  La generación del PDF se debe hacer en una clase aparte encargada de esto. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 06/12/2016 | Tests de alta y baja reserva. | | Lógica alta y baja reserva y generar PDF de reserva y su test. | |  | |
| 07/12/2016 | Lógica alta y baja reserva. | | Generar PDF de reserva. | |  | |
| 08/12/2016 | Generar PDF de reserva y su test. | | Nada | |  | |

* Estimación 3 Task Points.
* Tiempo real 3 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica envío de mail | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 28/11/2016 |  |  | Task Points |  | 1 |
|  |  |  |  |  |  |  |
| Número de Historia | | 7 |  | Número de TaskCard | | 26 |
|  |  |  |  |  |  |  |
| Descripción:  Se recibe un objeto mail con un String de correo, un String de asunto, un mensaje y un Archivo para adjuntar.  La clase debe armar con estos datos un correo y lo debe enviar a través del servidor de correo y la cuenta del cliente. ([olimpoagilinmobiliaria2016@gmail.com](mailto:olimpoagilinmobiliaria2016@gmail.com)) | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 06/12/2016 | Hecha lógica envío de mail | |  | | Se ha utilizado la API de Gmail para simplificar la lógica del envío de mail.  2 horas. | |
| 07/12/2016 | Corregidos errores presentados en Windows | | Nada | | En Windows se presentaba un error que impedía el envío de mails, mientras que en Linux funcionaba correctamente.  1 hora. | |

* Estimación 1 Task Points.
* Tiempo real 1 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Cambios ABM Cliente | |  | Programador | 3 | |
|  |  |  |  |  |  |  |
| Fecha | 28/11/2016 |  |  | Task Points |  | 1 |
|  |  |  |  |  |  |  |
| Número de Historia | | 7 |  | Número de TaskCard | | 27 |
|  |  |  |  |  |  |  |
| Descripción:  Se debe agregar un campo de correo al cliente. Para esto hay que modificar tanto la vista de crear como la de modificar cliente. También hay que validar este correo en la lógica de alta y modificación de un cliente, y modificar la entidad Cliente. | | | | | | |
|  |  |  |  |  |  |  |
| Notas:  Recordar actualizar las pruebas. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 02/12/2016 | Terminada | | Nada | | Se actualizaron los casos de prueba y se verificó que no se hayan introducido errores. | |

* Estimación 1 Task Points.
* Tiempo real 1 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Persistidor y entidad reserva | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 28/11/2016 |  |  | Task Points |  | 1 |
|  |  |  |  |  |  |  |
| Número de Historia | | 7 |  | Número de TaskCard | | 28 |
|  |  |  |  |  |  |  |
| Descripción:  Se debe hacer una interfaz con métodos que permitan:   * Guardar una reserva (si resulta en error lanza excepción SaveUpdateException). * Modificar una reserva (si resulta en error lanza excepción SaveUpdateException). * Obtener todas las reservas (si resulta en error lanza excepción ConsultaException).   Todas las excepciones mencionadas extienden de PersistenceException.  Se debe hacer una clase que implemente dicha interfaz mediante hibernate.  Implementar la clase Reserva. La clase debe tener los siguientes atributos:   * id (Integer), * cliente (Cliente), * inmueble (Inmueble), * archivoPDF (PDF), * fechaInicio (Date), * fechaFin (Date), * estado (Estado).   Para cada atributo deben hacerse los métodos set y get.  Completar las entidades con las anotaciones necesarias para la persistencia con Hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas:  El atributo Estado debe representar el estado de la entidad (alta, baja).  El método para obtener todas las reservas debe buscar aquellas reservas con el estado “Alta”. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 30/11/2016 | Entidad Reserva, agregados atributos y anotaciones JPA a las clases relacionadas, métodos guardar, modificar y obtener reservas | | Nada | |  | |

* Estimación 1 Task Points.
* Tiempo real 1 Task Points.
* Desvío 0 Task Points de desvío.

### Ventas

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Vista alta y listar venta | |  | Programador | | 1 | |
|  |  |  |  |  | |  |  |
| Fecha | 29/11/2016 |  |  | Task Points | |  | 3 |
|  |  |  |  |  | |  |  |
| Número de Historia | | 8 |  | Número de TaskCard | | | 29 |
|  |  |  |  |  |  | |  |
| Descripción:  Un vendedor seleccionará, desde el listado de inmuebles, un inmueble que desea vender.  Se mostrará una pantalla con el propietario y datos relevantes del inmueble.  En la pantalla se pedirá que se ingrese el monto de venta y el nuevo propietario, que debe ser un cliente cargado en el sistema.  Se presenta un botón de cancelar venta y otro de realizar venta; en este último se pedirá la contraseña al vendedor para confirmar la venta. Al confirmar la venta, esta se pasa a la lógica para que genere un PDF y guarde la venta.  Si hay algún error debe mostrarse. Sino, debe abrirse una vista para visualizar el documento de venta en PDF generado por la lógica. Se permiten realizar dos acciones: imprimir y guardar.  Para la vista de listar ventas, se debe tener en cuenta de donde es llamada, ya que puede llamarse desde la selección de un vendedor (al listar vendedores), desde la selección de un cliente (al listar clientes), desde la selección de un propietario (al listar propietarios). Se debe mostrar el nombre y apellido de a quién pertenecen las ventas junto con su rol.  Al listar las ventas, se debe mostrar la fecha de la venta, cliente (nombre y apellido), propietario (nombre y apellido), inmueble (dirección), vendedor (nombre y apellido) – quitando la columna del rol que lo llamó. Se debe poder seleccionar una y ver el documento PDF generado al concretar la venta. | | | | | | | |
|  |  |  |  |  |  | |  |
| Notas: | | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | | |
| 02/12/2016 | Tests del controlador de alta venta, fxml de alta venta | | Controlador de alta venta, fxml y controlador de confirmar contraseña.  Test, fxml y controlador de listar venta | |  | | |
| 03/12/2016 | Controlador de alta venta, fxml y controlador de confirmar contraseña | | Test, fxml y controlador de listar venta | |  | | |
| 04/12/2016 | Test, fxml y controlador de listar venta | | Nada | |  | | |

* Estimación 3 Task Points.
* Tiempo real 3 Task Points.
* Desvío 0 Task Points de desvío.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica alta venta y generar PDF | |  | Programador | | 1 | |
|  |  |  |  |  | |  |  |
| Fecha | 28/11/2016 |  |  | Task Points | |  | 3 |
|  |  |  |  |  | |  |  |
| Número de Historia | | 8 |  | Número de TaskCard | | | 30 |
|  |  |  |  |  |  | |  |
| Descripción:  Se recibe un objeto venta con el monto de venta, un vendedor, un cliente, un propietario y un inmueble.  Se le asigna una fecha de venta.  Se valida que el inmueble se encuentre en el estado “No vendido”.  Se valida el monto ingresado y que el cliente que compra el inmueble sea distinto del propietario. También hay que validar que, si el inmueble tiene una reserva en el día de la fecha de venta, la venta no se realizará, a menos que el cliente de la reserva sea el mismo del de la compra.  Si todo es correcto se deberá generar un archivo PDF con los siguientes datos de la venta: fecha, monto, propietario anterior, propietario actual, y los datos relevantes del inmueble.  Setear el estado del inmueble a “Vendido”. | | | | | | | |
|  |  |  |  |  |  | |  |
| Notas  La generación del PDF se debe hacer en una clase aparte encargada de esto. | | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | | |
| 05/12/2016 | Test de alta venta y lógica de alta venta | | Test y lógica de generar pdf de venta | |  | | |
| 06/12/2016 | Test y lógica de generar pdf | | Nada | | Nos llevó menos tiempo por experiencia y soluciones similares realizadas anteriormente que ayudaron | | |

* Estimación 3 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Point de sobreestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Lógica imprimir venta | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 28/11/2016 |  |  | Task Points |  | 1 |
|  |  |  |  |  |  |  |
| Número de Historia | | 8 |  | Número de TaskCard | | 31 |
|  |  |  |  |  |  |  |
| Descripción:  Se recibe un objeto PDF y se realiza una impresión del mismo. | | | | | | |
|  |  |  |  |  |  |  |
| Notas: | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 07/12/2016 | Estructura (clase y método que no hace nada) para que pueda ser referenciado | | Lógica de la impresión | | No llevó un task point ya que tomó solo un par de minutos | |
| 09/12/2016 | Lógica de la impresión | | Nada | | Nos llevó más tiempo de lo estimado ya que nos encontramos con problemas para realizar este tipo de impresión | |

* Estimación 1 Task Points.
* Tiempo real 2 Task Points.
* Desvío 1 Task Points de subestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Persistidor y entidad venta | |  | Programador | 2 | |
|  |  |  |  |  |  |  |
| Fecha | 29/11/2016 |  |  | Task Points |  | 2 |
|  |  |  |  |  |  |  |
| Número de Historia | | 8 |  | Número de TaskCard | | 32 |
|  |  |  |  |  |  |  |
| Descripción  Se debe hacer una interfaz con métodos que permitan:   * Guardar una venta (si resulta en error lanza excepción SaveUpdateException). * Modificar una venta (si resulta en error lanza excepción SaveUpdateException). * Obtener todas las ventas (si resulta en error lanza excepción ConsultaException). * Obtener todas las ventas de un vendedor (si resulta en error lanza excepción ConsultaException). * Obtener todas las ventas de un propietario (si resulta en error lanza excepción ConsultaException). * Obtener todas las ventas de un cliente (si resulta en error lanza excepción ConsultaException). * Obtener la venta de un inmueble (si resulta en error lanza excepción ConsultaException)   Todas las excepciones mencionadas extienden de PersistenceException.  Se debe hacer una clase que implemente dicha interfaz mediante hibernate.  Implementar la clase Venta. La clase debe tener los siguientes atributos:   * id (Integer), * cliente (Cliente), * inmueble (Inmueble), * propietario (Propietario), * vendedor (Vendedor), * archivoPDF (PDF), * importe (Double), * medioDePago (String), * fecha (Date).   Para cada atributo deben hacerse los métodos set y get.  Completar las entidades con las anotaciones necesarias para la persistencia con Hibernate. | | | | | | |
|
|
|  |  |  |  |  |  |  |
| Notas:  El atributo Estado debe representar el estado de la entidad (alta, baja).  Los métodos para obtener todas las ventas deben buscar aquellas ventas con el estado “Alta”. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 02/12/2016 | Anotaciones JPA, clase venta, relaciones con las demás clases y método guardar venta | | Nada | | No fue necesario realizar los métodos obtener ventas ya que las ventas pueden ser obtenidas directamente de las clases vendedor, propietario e inmueble | |

* Estimación 2 Task Points.
* Tiempo real 1 Task Points.
* Desvío 1 Task Points de sobreestimación.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Task Card** | Agregar EstadoInmueble a inmueble | |  | Programador | 1 | |
|  |  |  |  |  |  |  |
| Fecha | 26/11/2016 |  |  | Task Points |  | 1 |
|  |  |  |  |  |  |  |
| Número de Historia | | 8 |  | Número de TaskCard | | 33 |
|  |  |  |  |  |  |  |
| Descripción:  Se debe agregar un campo de EstadoInmueble al Inmueble. Para esto hay que modificar la lógica de creación del inmueble y modificar la entidad Inmueble.  Se debe agregar un campo de EstadoInmueble al HistorialInmueble. Para esto hay que modificar la lógica de modificación del inmueble y modificar la entidad HistorialInmueble.  También hay que agregar una función para listar los EstadoInmueble en el gestor de datos y en persistidor de datos.  Hay que crear una entidad EstadoInmueble y un Enum EstadoInmuebleStr (VENDIDO y NO\_VENDIDO). También hay que agregar estos Strings al sql inicial que carga los datos necesarios para ejecutar el sistema. | | | | | | |
|  |  |  |  |  |  |  |
| Notas  Recordar actualizar las pruebas. | | | | | | |
|
|
| Fecha | Realizado | | A realizar | | Comentarios | |
| 07/12/2016 | Terminada | | Nada | | 1 hora. | |

* Estimación 1 Task Points.
* Tiempo real 1/3 Task Points.
* Desvío 2/3 Task Points de sobreestimación.

## Estimación de Esfuerzo

Sumatoria de desvíos = 5/3 Task Points de sobreestimación (5 horas).

## Conclusiones parciales de lo estimado

El desvío por sobre lo estimado es debido a la experiencia de la iteración anterior y el aprendizaje ya afianzado de algunas herramientas y frameworks.

Las codificaciones de las pruebas no consumieron mucho tiempo y algunas tasks se sobre estimaron con el objetivo de dedicar un poco más de tiempo al diseño de las interfaces, probar a fondo las funcionalidades y depurar el código.

Las cuestiones como el manejo de transiciones entre pantallas, manejo de excepciones, validaciones, entre otras, ya estaban más claras y aceptadas por todo el equipo de desarrollo.

# Código Fuente

El código fuente correspondiente a cada Task Card se encuentra en la sección Anexos, ordenado por historia de usuario y por task card.

# Pruebas de Unidad

El código correspondiente a las pruebas de unidad, junto con los casos de prueba, correspondiente a cada Task Card se encuentra en la sección Anexos, ordenado por historia de usuario y por task card, debajo del código fuente del método a probar.

# Refactorización

(deben especificarse a que tarea corresponden)

# Estándares de programación

El equipo de desarrollo creó y estableció un estándar de programación mediante un formateador de código que se ejecuta cada vez que se guarda un archivo, el cual todos los miembros lo tenían seteado en el IDE que utilizaban. Este formateador define cuestiones de indentación, supresión de imports no utilizados, supresión de líneas de código vacías, utilización de corchetes, entre otras (ver estilo en el código adjuntado).

También se definió:

* Nombres de variables.
* Estilo de documentación.
* Orden de código dentro de una clase (atributos, setters, getters, hash, equals, toString).

# Conclusiones

Realizando una comparación con las metodologías tradicionales usadas en la carrera, programación extrema nos resultó más cómoda y nos proporcionó una sensación de utilidad para cada producto de trabajo realizado; la metodología, así como indican os principios ágiles, busca agregar valor al cliente, si un producto de trabajo no produce este efecto, entonces no es necesario hacerlo; por lo tanto nos centramos en aquellas cosas que proporcionan valor para el cliente, o comunican información valiosa para el equipo de desarrollo.

Por otro lado, también tiene cuestiones “incómodas”. Una es la programación por parejas, que, si bien es efectiva y nos dio mucho resultado en la detección de errores en tiempo real, la dependencia y la coordinación de tiempos que se tiene que tener con la otra persona para programar es impracticable cuando los tiempos entre ambas no coinciden (trabajo, facultad, otros). Y la otra cuestión, es que depende mucho de la voluntad de los miembros del equipo de desarrollo; cuando existen cuestiones que no fueron tenidas en cuenta para agregarlas en una task card, las implementaciones necesarias que tiene que hacer una pareja para no detener el desarrollo son completamente voluntarias (implementar clases que son necesarias para modelar un atributo, implementar los mecanismos para la transición entre pantallas, escribir archivos de configuración para la base de datos, escribir los inserts necesarios para iniciar el sistema con datos, entre otros).

Estas cuestiones surgieron al principio y luego se fueron aceitando para el fin de la iteración 1.

Notamos que para el éxito de un proyecto de software utilizando esta metodología es necesario contar con un equipo de desarrollo compacto, con valores y excelencia técnica, para tomar decisiones y no paralizar el avance del proyecto con conflictos eternos.

La codificación de las pruebas antes de codificar fue lo que más costó. Teníamos la teoría de TDD, pero no la práctica. Aunque éramos conscientes de que debíamos codificar las pruebas antes de empezar a codificar el método, muchas veces ocurrió que empezamos a codificar, hasta que nos dábamos cuenta que estábamos haciendo las cosas mal. Después, pudimos comprobar la ventaja que tiene usar TDD. La codificación era a consciencia, teniendo en cuenta las cosas que podían fallar, ya que se habían definido los casos de prueba. También, las pruebas unitarias se corrían para realizar pruebas de regresión cuando se modificaba un método. Los resultados son positivos, aunque haya que sacrificar tiempo de codificación, luego se ahorra.

ANEXOS

# Código Fuente y Pruebas de Unidad

## Iteración 1

### Taskcard 1 Vista y lógica del Login

Código del archivo LoginController.java

/\*\*

\* Método que permite a un vendedor entrar al sistema. Se llama al hacer click en el botón ingresar.

\* Pertenece a la taskcard 1 de la iteración 1 y a la historia 1

\*

\* **@return** ResultadoControlador que resume lo que hizo el controlador

\*/

@FXML

**public** ResultadoControlador ingresar() {

//Inicialización de variables

Set<ErrorControlador> erroresControlador = **new** HashSet<>();

ResultadoAutenticacion resultado = **null**;

Boolean hayErrores;

DatosLogin datos;

String errores = "";

//Toma de datos de la vista

TipoDocumento tipoDocumento = cbTipoDocumento.getValue();

String dni = tfNumeroDocumento.getText().trim();

**char**[] pass = pfContra.getText().toCharArray();

//Procesamiento de errores de la vista

**if**(tipoDocumento == **null** || dni.isEmpty() || pass.length < 1){

presentador.presentarError("No se ha podido iniciar sesión", "Campos vacíos.", stage);

**return** **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***);

}

//Se cargan los datos de la vista como datos a utilizar en el login

datos = **new** DatosLogin(tipoDocumento, dni, pass);

//Inicio transacción al gestor

**try**{

//Se llama a la lógica para loguear al vendedor y se recibe el resultado de las validaciones y datos extras de ser necesarios

resultado = coordinador.autenticarVendedor(datos);

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

//Procesamiento de errores de la lógica

hayErrores = resultado.hayErrores();

**if**(hayErrores){

**for**(ErrorAutenticacion r: resultado.getErrores()){

**switch**(r) {

**case** ***Datos\_Incorrectos***:

errores += "Datos inválidos al iniciar sesión.\n";

erroresControlador.add(ErrorControlador.***Datos\_Incorrectos***);

**break**;

}

}

**if**(!errores.isEmpty()){

//Se muestran los errores

presentador.presentarError("No se ha podido iniciar sesión", errores, stage);

}

}

**else**{

//Se pasa a la pantalla de inicio

BaseController siguientePantalla = (BaseController) cambiarmeAScene(BaseController.***URLVista***, ***URLVista***, **true**);

//Se le setea el vendedor logueado

siguientePantalla.formatearConVendedor(resultado.getVendedorLogueado());

}

**return** **new** ResultadoControlador(erroresControlador.toArray(**new** ErrorControlador[0]));

}

/\*\*

\* Método que se llama al hacer click en el botón registrar.

\*/

@FXML

**public** **void** registrar() {

//Se pasa a la pantalla de registro de vendedor

cambiarmeAScene(AltaVendedorController.***URLVista***, ***URLVista***, **true**);

}

Prueba de unidad del archivo LoginControllerTest.java

//Casos de prueba

//tipoDocumento, numDoc, contra, resultadoVista, resultadoLogica, excepcion

/\* 0 \*/**new** Object[] { (**new** TipoDocumento()).setTipo(TipoDocumentoStr.***DNI***), "12345678", "pepe", **new** ResultadoControlador(), **new** ResultadoAutenticacion(*vendedorRetorno*), **null** }, //prueba ingreso correcto

/\* 1 \*/**new** Object[] { **null**, "", "pepe", **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***), **new** ResultadoAutenticacion(**null**, ErrorAutenticacion.***Datos\_Incorrectos***), **null** }, //prueba TipoDocumento vacio

/\* 2 \*/**new** Object[] { (**new** TipoDocumento()).setTipo(TipoDocumentoStr.***LC***), "", "pepe", **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***), **new** ResultadoAutenticacion(**null**, ErrorAutenticacion.***Datos\_Incorrectos***), **null** }, //prueba numero de documento vacio

/\* 3 \*/**new** Object[] { (**new** TipoDocumento()).setTipo(TipoDocumentoStr.***LE***), "12345678", "", **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***), **new** ResultadoAutenticacion(**null**, ErrorAutenticacion.***Datos\_Incorrectos***), **null** }, //prueba Contraseña vacia

/\* 4 \*/**new** Object[] { (**new** TipoDocumento()).setTipo(TipoDocumentoStr.***CEDULA\_EXTRANJERA***), "12345678", "pepe", **new** ResultadoControlador(ErrorControlador.***Datos\_Incorrectos***), **new** ResultadoAutenticacion(**null**, ErrorAutenticacion.***Datos\_Incorrectos***), **null** }, //prueba un ingreso incorrecto

/\* 5 \*/**new** Object[] { (**new** TipoDocumento()).setTipo(TipoDocumentoStr.***PASAPORTE***), "ñú", "pepe", **new** ResultadoControlador(ErrorControlador.***Datos\_Incorrectos***), **new** ResultadoAutenticacion(**null**, ErrorAutenticacion.***Datos\_Incorrectos***), **null** }, //prueba un ingreso incorrecto con caracteres UTF8

/\* 6 \*/**new** Object[] { (**new** TipoDocumento()).setTipo(TipoDocumentoStr.***LC***), "ñú", "pepe", **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***), **null**, **new** ObjNotFoundException("Error de persistencia. Test.", **new** Exception()) }, //Prueba una excepcion de persistencia

/\* 7 \*/**new** Object[] { (**new** TipoDocumento()).setTipo(TipoDocumentoStr.***DNI***), "ñú", "pepe", **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***), **null**, **new** Exception() } //Prueba una excepcion desconocida

/\*\*

\* Prueba el método ingresar(), el cual corresponde con la taskcard 1 de la iteración 1 y a la historia 1

\*

\* **@param** tipoDocumento

\* que se usará en el test

\* **@param** numDoc

\* que se usará en el test

\* **@param** contra

\* es la contraseña que se usará en el test

\* **@param** resultadoVista

\* es lo que se espera que devuelva el metodo

\* **@param** resultadoLogica

\* es lo que el mock de la lógica debe devolver en el test y que el controlador recibe

\* **@param** excepcion

\* es la excepcion que debe lanzar el mock de la lógica, si la prueba involucra procesar una excepcion de dicha lógica, debe ser nulo resultadoLogica para que se use

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testIngresar(TipoDocumento tipoDocumento, String numDoc, String contra, ResultadoControlador resultadoVista, ResultadoAutenticacion resultadoLogica, Throwable excepcion) **throws** Exception {

//Se crean los mocks de la prueba

CoordinadorJavaFX coordinadorMock = **new** CoordinadorJavaFX() {

@Override

**public** ResultadoAutenticacion autenticarVendedor(DatosLogin login) **throws** PersistenciaException {

**if**(resultadoLogica != **null**){

**return** resultadoLogica;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

};

PresentadorVentanas presentadorMock = **new** PresentadorVentanasMock();

//Se crea el controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos que ingresaría el usuario en la vista

LoginController loginController = **new** LoginController() {

@Override

**public** ResultadoControlador ingresar() {

tfNumeroDocumento.setText(numDoc);

pfContra.setText(contra);

cbTipoDocumento.getItems().clear();

cbTipoDocumento.getItems().add(tipoDocumento);

cbTipoDocumento.getSelectionModel().select(tipoDocumento);

**return** **super**.ingresar();

};

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

}

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**protected** OlimpoController cambiarmeAScene(String URLVistaACambiar, String URLVistaRetorno, Boolean useSceneSize) {

**return** **new** BaseController() {

@Override

**public** **void** formatearConVendedor(Vendedor vendedor) {

*assertEquals*(*vendedorRetorno*, vendedor);

}

};

}

};

loginController.setCoordinador(coordinadorMock);

loginController.setPresentador(presentadorMock);

//Se crea lo necesario para correr la prueba en el hilo de JavaFX porque los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(LoginController.***URLVista***, loginController);

loginController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

*assertEquals*(resultadoVista, loginController.ingresar());

}

};

**try**{

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

}

### Taskcard 2 Entidad vendedor

Código del archivo en Vendedor.java

@NamedQueries(value = { @NamedQuery(name = "obtenerVendedor", query = "SELECT v FROM Vendedor v WHERE numeroDocumento = :documento AND tipoDocumento.tipo = :tipoDocumento"), @NamedQuery(name = "listarVendedores", query = "SELECT v FROM Vendedor v WHERE v.estado.estado = 'ALTA'") })

@Entity

@Table(name = "vendedor", uniqueConstraints = @UniqueConstraint(name = "vendedor\_numerodocumento\_idtipo\_uk", columnNames = { "numerodocumento", "idtipo" }))

/\*\*

\* Entidad que modela a un vendedor

\* Task card 2 de la iteración 1, historia de usuario 1

\*/

**public** **class** Vendedor {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**protected** Integer id; //ID

@Column(name = "nombre", length = 30, nullable = **false**)

**private** String nombre;

@Column(name = "apellido", length = 30, nullable = **false**)

**private** String apellido;

@Column(name = "numerodocumento", length = 30, nullable = **false**)

**private** String numeroDocumento;

@Column(name = "password", length = 100, nullable = **false**)

**private** String password;

@Column(name = "salt", nullable = **false**)

**private** String salt;

@Column(name = "root", nullable = **false**)

**private** Boolean root; //representa si es un vendedor con permisos privilegiados

@ManyToOne(fetch = FetchType.***LAZY***)

@JoinColumn(name = "idtipo", referencedColumnName = "id", foreignKey = @ForeignKey(name = "vendedor\_idtipo\_fk"), nullable = **false**)

**private** TipoDocumento tipoDocumento;

//estado lógico que tiene la entidad

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idestado", referencedColumnName = "id", foreignKey = @ForeignKey(name = "vendedor\_idestado\_fk"), nullable = **false**)

**private** Estado estado;

//getters, setters, constructores y otros métodos

(…)

### Taskcard 3 Vista alta, modificar y baja vendedor

Código del archivo AdministrarVendedorController.java

/\*\*

\* Acción que se ejecuta al presionar el botón agregar

\* Se pasa a la pantalla alta vendedor

\*/

**public** **void** agregarAction(ActionEvent event) {

//Se llama a la pantalla alta vendedor

cambiarmeAScene(AltaVendedorController.***URLVista***, ***URLVista***);

}

/\*\*

\* Acción que se ejecuta al presionar el botón modificar

\* Se pasa a la pantalla modificar vendedor

\*/

**public** **void** modificarAction(ActionEvent event) {

//Se comprueba que efectivamente haya un vendedor seleccionado

Vendedor vendedor = tablaVendedores.getSelectionModel().getSelectedItem();

**if**(vendedor == **null**){

**return**;

}

//Se llama a la pantalla de modificar vendedor

ModificarVendedorController controlador = (ModificarVendedorController) cambiarmeAScene(ModificarVendedorController.***URLVista***, ***URLVista***);

controlador.setVendedor(vendedor);

}

/\*\*

\* Acción que se ejecuta al presionar el botón eliminar

\* Se muestra una ventana emergente para confirmar la operación

\*/

**public** **void** eliminarAction(ActionEvent event) {

//Se comprueba que efectivamente haya un vendedor seleccionado

Vendedor vendedor = tablaVendedores.getSelectionModel().getSelectedItem();

**if**(vendedor == **null**){

**return**;

}

//Se pregunta al usuario si quiere eliminar el vendedor

VentanaConfirmacion ventana = presentador.presentarConfirmacion("Eliminar vendedor", "Está a punto de eliminar al vendedor.\n¿Desea continuar?", **this**.stage);

**if**(ventana.acepta()){

**try**{

//Se llama a la lógica para eliminar el inmueble y se recibe el resultado de las validaciones

ResultadoEliminarVendedor resultado = coordinador.eliminarVendedor(vendedor);

**if**(resultado.hayErrores()){

StringBuilder stringErrores = **new** StringBuilder();

//Procesamiento de errores de la lógica

**for**(ErrorEliminarVendedor err: resultado.getErrores()){

**switch**(err) {

//Por el momento no hay errores que se puedan mostrar

}

}

//Se muestran los errores devueltos por la vista

presentador.presentarError("No se pudo eliminar el vendedor", stringErrores.toString(), stage);

}

**else**{

//Si no hubo errores

//Se quita el inmueble de la vista

tablaVendedores.getItems().remove(vendedor);

//Se muestra una notificación de que se eliminó correctamente el inmueble

presentador.presentarToast("Se ha eliminado al vendedor " + vendedor.getNombre() + " con éxito", stage);

}

//Se muestran las excepciones, en caso de que ocurra alguno

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e,stage);

}

}

}

Prueba de unidad del archivo AdministrarInmuebleControllerTest.java

/\*\*

\* Prueba el método agregarVendedor()

\* Se comprueba que se llame a la pantalla correspondiente cuando se presiona el boton agregar

\*

\* **@throws** Throwable

\*/

@Test

**public** **void** testAgregarVendedor() **throws** Throwable {

//Se crean los mocks necesarios

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

AltaVendedorController altaVendedorControllerMock = *mock*(AltaVendedorController.**class**);

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

//Se setea lo que debe devolver el mock cuando es invocado por la clase a probar

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(altaVendedorControllerMock);

//Controlador a probar;

AdministrarVendedorController administrarVendedorController = **new** AdministrarVendedorController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.setScenographyChanger(scenographyChangerMock);

**super**.inicializar(location, resources);

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarVendedorController.***URLVista***, administrarVendedorController);

administrarVendedorController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Método a probar

administrarVendedorController.agregarAction(**null**);

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(scenographyChangerMock, *times*(1)).cambiarScenography(AltaVendedorController.***URLVista***, **false**);

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

;

}

//Casos de prueba

//listaVendedores, llamaAModificar

/\* 0 \*/ **new** Object[] { listaVendedoresConUnVendedor, 1}, //Si hay un vendedor seleccionado se debe llamar a la pantalla modificar

/\* 1 \*/ **new** Object[] { listaVendedoresVacia, 0} //Si la lista es vacía no se debería llamar a modificar

/\*\*

\* Prueba el método Modificar Vendedor()

\* **@param** listaVendedores

\* vendedores inicialmente en la tabla de vendedores

\* **@param** llamaAModificar

\* indica si se debe llamar a la pantalla modificar vendedor

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testModificarVendedor(ArrayList<Vendedor> listaVendedores, Integer llamaAModificar) **throws** Throwable {

//Se crean los mocks necesarios

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

ModificarVendedorController modificarVendedorControllerMock = *mock*(ModificarVendedorController.**class**);

//Se setea lo que deben devolver los mocks cuando son invocados por la clase a probar

*when*(coordinadorMock.obtenerVendedores()).thenReturn(listaVendedores);

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(modificarVendedorControllerMock);

//Controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos en la tabla

AdministrarVendedorController administrarVendedorController = **new** AdministrarVendedorController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.setScenographyChanger(scenographyChangerMock);

**super**.inicializar(location, resources);

**this**.tablaVendedores.getSelectionModel().select(0);

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarVendedorController.***URLVista***, administrarVendedorController);

administrarVendedorController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Método a probar

administrarVendedorController.modificarAction(**null**);

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(scenographyChangerMock, *times*(llamaAModificar)).cambiarScenography(ModificarVendedorController.***URLVista***, **false**);

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

//Casos de prueba

//listaVendedores, llamaAPresentadorVentanasPresentarConfirmacion, aceptarVentanaConfirmacion, resultadoEliminarVendedorEsperado, llamaAPresentadorVentanasPresentarError, llamaAPresentadorVentanasPresentarExcepcion, excepcion

/\* 0 \*/ **new** Object[] { listaVendedores, 1, **true**, resultadoSinError, 0, 0, **null** }, //test correcto, se acepta la confirmación y sin errores

/\* 1 \*/ **new** Object[] { listaVendedores, 1, **true**, resultadoSinError, 0, 1, exception }, //la lógica devuelve una excepción

/\* 2 \*/ **new** Object[] { listaVendedores, 1, **false**, resultadoSinError, 0, 0, **null** }, //no se acepta la ventna de confirmación

/\* 3 \*/ **new** Object[] { listaVendedoresVacia, 0, **false**, resultadoSinError, 0, 0, **null** }, //la lista de vendedores es vacía

/\*\*

\* Prueba el método eliminar Vendedor()

\* **@param** listaVendedores

\* vendedores inicialmente en la tabla de vendedores

\* **@param** llamaAPresentadorVentanasPresentarConfirmacion

\* indica si se debe llamar a la ventana de presentar confirmación

\* **@param** aceptarVentanaConfirmacion

\* indica si el usuario acepta la ventana de confirmación

\* **@param** resultadoEliminarVendedorEsperado

\* resultado devuelto por la capa lógica

\* **@param** llamaAPresentadorVentanasPresentarError

\* indica si se debe presentar un error

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* indica si se debe presentar una excepcion

\* **@param** excepcion

\* excepcion lanzada por la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testEliminarVendedor(ArrayList<Vendedor> listaVendedores,

Integer llamaAPresentadorVentanasPresentarConfirmacion,

Boolean aceptarVentanaConfirmacion,

ResultadoEliminarVendedor resultadoEliminarVendedorEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

PersistenciaException excepcion)

**throws** Throwable {

//Se crean los mocks necesarios

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaConfirmacion ventanaConfirmacionMock = *mock*(VentanaConfirmacion.**class**);

//Se setea lo que deben devolver los mocks cuando son invocados por la clase a probar

*when*(coordinadorMock.obtenerVendedores()).thenReturn(listaVendedores);

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarConfirmacion(*any*(), *any*(), *any*())).thenReturn(ventanaConfirmacionMock);

*when*(ventanaConfirmacionMock.acepta()).thenReturn(aceptarVentanaConfirmacion);

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

*when*(coordinadorMock.eliminarVendedor(*any*(Vendedor.**class**))).thenReturn(resultadoEliminarVendedorEsperado);

**if**(excepcion != **null**){

*when*(coordinadorMock.eliminarVendedor(*any*(Vendedor.**class**))).thenThrow(excepcion);

}

//Controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos en la tabla

AdministrarVendedorController administrarVendedorController = **new** AdministrarVendedorController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.setScenographyChanger(scenographyChangerMock);

**this**.presentador = **new** PresentadorVentanas();

**this**.presentador = presentadorVentanasMock;

**super**.inicializar(location, resources);

**this**.tablaVendedores.getSelectionModel().select(0); //se selecciona el primer vendedor de la lista

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarVendedorController.***URLVista***, administrarVendedorController);

administrarVendedorController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

Integer cantidadVendedoresAntesDeEliminar = administrarVendedorController.tablaVendedores.getItems().size();

//Método a probar

administrarVendedorController.eliminarAction(**null**);

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarConfirmacion)).presentarConfirmacion(*any*(), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarConfirmacion)).presentarConfirmacion(*any*(), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

**if**(llamaAPresentadorVentanasPresentarExcepcion == 0){

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*any*(), *any*(), *any*());

}

Integer cantidadVendedoresDespuesDeEliminar = administrarVendedorController.tablaVendedores.getItems().size();

**if**(llamaAPresentadorVentanasPresentarError != 1 && llamaAPresentadorVentanasPresentarExcepcion != 1 && aceptarVentanaConfirmacion){

*assertNotEquals*(cantidadVendedoresAntesDeEliminar, cantidadVendedoresDespuesDeEliminar);

}

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

Código del archivo AltaVendedorController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga al vendedor y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, éstos se muestran al usuario.

\*/

**public** **void** acceptAction() {

//Se toman los datos ingresados por el usuario

cargarDatos();

//Se validan los datos ingresados por el usuario

StringBuilder error = **new** StringBuilder("");

validarVistaAltaVendedor(error);

//Si hay errores, se los muestra en pantalla, si no, se llama a crear un vendedor en la capa lógica

**if**(!error.toString().isEmpty()){

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

crearVendedor();

}

}

/\*\*

\* Método que encapsula el llamado de crear un vendedor a la capa lógica y maneja los errores devueltos

\*/

**private** **void** crearVendedor() {

//Se crea el vendedor a pasarle a la capa lógica

Vendedor vendedor = **new** Vendedor();

vendedor.setNombre(nombre)

.setApellido(apellido)

.setNumeroDocumento(numeroDocumento)

.setTipoDocumento(tipoDoc)

.setSalt(encriptador.generarSal())

.setPassword(encriptador.encriptar(password1.toCharArray(), vendedor.getSalt()));

ResultadoCrearVendedor resultadoCrearVendedor = **null**;

**try**{

//Se llama a crear un vendedor en la capa lógica

resultadoCrearVendedor = coordinador.crearVendedor(vendedor);

StringBuilder error = **new** StringBuilder("");

List<ErrorCrearVendedor> listaErrores = resultadoCrearVendedor.getErrores();

//Se crea un mensaje apropiado a mostrar según el error recibido

parsearErroresLogica(error, listaErrores);

//Si hay errores, se los muestra en pantalla, si no, se presenta una notificación indicando el éxito de la operación

**if**(!error.toString().isEmpty()){

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

presentador.presentarToast("Se ha creado el vendedor con éxito", stage);

salir();

}

//Se manejan las excepciones que puede devolver la lógica

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

} **catch**(EntidadExistenteConEstadoBajaException e){

//Si el vendedor ya existía pero fue dado de baja se debe mostrar una ventana preguntando al usuario si desea darlo de alta

manejarVendedorExistenteBaja(vendedor);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e, stage); //falta el stage

}

}

/\*\*

\* Método que se llama al querer dar de alta un vendedor que fue dado de baja previamente

\* Si el usuario confirma la operación se llama a la pantalla modificar vendedor

\*

\* **@param** vendedor

\*/

**private** **void** manejarVendedorExistenteBaja(Vendedor vendedor) {

VentanaConfirmacion ventana = presentador.presentarConfirmacion("El vendedor ya existe", "El vendedor está dado de baja. Si continúa podrá darlo de alta nuevamente. ¿Desea continuar?", stage);

**if**(ventana.acepta()){

//Si el usuario acepta se llama a la pantalla modificar vendedor

**try**{

vendedor = coordinador.obtenerVendedor(vendedor);

} **catch**(PersistenciaException e1){

presentador.presentarExcepcion(e1, stage);

}

ModificarVendedorController controlador = (ModificarVendedorController) cambiarmeAScene(ModificarVendedorController.***URLVista***, URLVistaRetorno);

controlador.setVendedor(vendedor);

controlador.setAltaVendedor();

}

}

/\*\*

\* Se convierten los errores devueltos por la capa lógica a errores a mostrar al usuario

\*

\* **@param** error

\* Texto a mostrar al usuario

\* **@param** listaErrores

\* Lista de errores de la capa lógica

\*/

**private** **void** parsearErroresLogica(StringBuilder error, List<ErrorCrearVendedor> listaErrores) {

**if**(listaErrores.contains(ErrorCrearVendedor.***Formato\_Nombre\_Incorrecto***)){

error.append("Nombre Incorrecto").append("\r\n");

}

**if**(listaErrores.contains(ErrorCrearVendedor.***Formato\_Apellido\_Incorrecto***)){

error.append("Apellido Incorrecto").append("\r\n");

}

**if**(listaErrores.contains(ErrorCrearVendedor.***Formato\_Documento\_Incorrecto***)){

error.append("Documento Incorrecto").append("\r\n");

}

**if**(listaErrores.contains(ErrorCrearVendedor.***Ya\_Existe\_Vendedor***)){

error.append("Ya existe un vendedor registrado con ese documento").append("\r\n");

}

}

/\*\*

\* Se valida que los campos ingresados por el usuario sean correctos

\*

\* **@param** error

\* Texto a mostrar en caso de que haya algún error

\*/

**private** **void** validarVistaAltaVendedor(StringBuilder error) {

**if**(nombre.isEmpty()){

error.append("Inserte un nombre").append("\r\n");

}

**if**(apellido.isEmpty()){

error.append("Inserte un apellido").append("\r\n");

}

**if**(tipoDoc == **null**){

error.append("Elija un tipo de documento").append("\r\n");

}

**if**(numeroDocumento.isEmpty()){

error.append("Inserte un numero de documento").append("\r\n");

}

**if**(password1.isEmpty() && password2.isEmpty()){

error.append("Inserte su contraseña").append("\r\n");

}

**if**(!password1.isEmpty() && password2.isEmpty()){

error.append("Inserte su contraseña nuevamente").append("\r\n");

}

**if**(!password1.equals(password2)){

error.append("Sus contraseñas no coinciden, Ingreselas nuevamente").append("\r\n ");

}

}

/\*\*

\* Se cargan los datos de la vista a variables para su posterior utilización

\*/

**private** **void** cargarDatos() {

nombre = textFieldNombre.getText().trim();

apellido = textFieldApellido.getText().trim();

numeroDocumento = textFieldNumeroDocumento.getText().trim();

password1 = passwordFieldContraseña.getText();

password2 = passwordFieldRepiteContraseña.getText();

tipoDoc = comboBoxTipoDocumento.getValue();

}

Prueba de unidad del archivo AltaVendedorControllerTest.java

//Casos de prueba //nombre,apellido,tipoDocumento,numeroDocumento,contraseña,contraseña2,resultadoCrearVendedorEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,llamaAPresentadorVentanasPresentarExcepcionInesperada,llamaACrearVendedor,excepcion,aceptarVentanaConfirmacion,llamaACambiarScene

/\* 0 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", ***resultadoCorrecto***, 0, 0, 0, 1, **null**, **true**, 0 }, //prueba correcta

/\* 1 \*/**new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", ***resultadoCrearNombreIncorrecto***, 1, 0, 0, 1, **null**, **true**, 0 }, //prueba nombre incorrecto

/\* 2 \*/**new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", ***resultadoCrearApellidoIncorrecto***, 1, 0, 0, 1, **null**, **true**, 0 }, //prueba apellido incorrecto

/\* 3 \*/**new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", ***resultadoCrearDocumentoIncorrecto***, 1, 0, 0, 1, **null**, **true**, 0 }, //prueba documento incorrecto

/\* 4 \*/**new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", ***resultadoCrearYaExiste***, 1, 0, 0, 1, **null**, **true**, 0 }, //prueba ya existe vendedor

/\* 5 \*/**new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", **new** ResultadoCrearVendedor(ErrorCrearVendedor.***Formato\_Nombre\_Incorrecto***, ErrorCrearVendedor.***Formato\_Apellido\_Incorrecto***), 1, 0, 0, 1, **null**, **true**, 0 }, //prueba nombre y apellido incorrectos

/\* 6 \*/**new** Object[] { "", "Perez", doc, "12345678", "abc", "abc", **null**, 1, 0, 0, 0, **null**, **true**, 0 }, //prueba nombre vacio

/\* 7 \*/**new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", ***resultadoCorrecto***, 0, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **true**, 1 }, //prueba Vendedor Existente y acepta

/\* 8 \*/**new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", ***resultadoCorrecto***, 0, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **false**, 0 }, //prueba Vendedor Existente y cancela

/\* 9 \*/**new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", ***resultadoCorrecto***, 0, 1, 0, 1, **new** SaveUpdateException(**new** Throwable()), **false**, 0 }, //prueba PersistenciaException

**public** **class** AltaVendedorControllerTest {

/\*\*

\* **@param** nombre

\* nombre del vendedor a crear

\* **@param** apellido

\* apellido del vendedor a crear

\* **@param** tipoDocumento

\* tipoDocumento del vendedor a crear

\* **@param** numeroDocumento

\* numeroDocumento del vendedor a crear

\* **@param** contraseña

\* contraseña ingresada

\* **@param** contraseña2

\* segunda contraseña ingresada

\* **@param** resultadoCrearVendedorEsperado

\* apellido del vendedor a crear

\* **@param** llamaAPresentadorVentanasPresentarError

\* indica si se debe presentar una ventana de error

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* indica si se debe presentar una ventana de excepción

\* **@param** llamaAPresentadorVentanasPresentarExcepcionInesperada

\* indica si se debe presentar una ventana de excepción inesperada

\* **@param** llamaACrearVendedor

\* indica si se debe llamar al método crear vendedor de la lógica

\* **@param** excepcion

\* excepción devuelta por la lógica

\* **@param** aceptarVentanaConfirmacion

\* indica si el usuario acepta la ventana de confirmación

\* **@param** llamaACambiarScene

\* indica si debe llamar al cambio de pantalla

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCrearVendedor(String nombre, String apellido, TipoDocumento tipoDocumento, String numeroDocumento, String contraseña, String contraseña2, ResultadoCrearVendedor resultadoCrearVendedorEsperado, Integer llamaAPresentadorVentanasPresentarError, Integer llamaAPresentadorVentanasPresentarExcepcion, Integer llamaAPresentadorVentanasPresentarExcepcionInesperada, Integer llamaACrearVendedor, Exception excepcion, Boolean aceptarVentanaConfirmacion, Integer llamaACambiarScene) **throws** Throwable {

//Se crean los mocks necesarios

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaErrorExcepcionInesperada ventanaErrorExcepcionInesperadaMock = *mock*(VentanaErrorExcepcionInesperada.**class**);

VentanaConfirmacion ventanaConfirmacionMock = *mock*(VentanaConfirmacion.**class**);

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

ModificarVendedorController modificarVendedorControllerMock = *mock*(ModificarVendedorController.**class**);

//Se setea lo que deben devolver los mocks cuando son invocados por la clase a probar

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarExcepcionInesperada(*any*(), *any*())).thenReturn(ventanaErrorExcepcionInesperadaMock);

*when*(presentadorVentanasMock.presentarConfirmacion(*any*(), *any*(), *any*())).thenReturn(ventanaConfirmacionMock);

*when*(ventanaConfirmacionMock.acepta()).thenReturn(aceptarVentanaConfirmacion);

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(modificarVendedorControllerMock);

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

Vendedor vendedor = **new** Vendedor()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setPassword(contraseña);

*when*(coordinadorMock.crearVendedor(vendedor)).thenReturn(resultadoCrearVendedorEsperado);

**if**(excepcion != **null**){

*when*(coordinadorMock.crearVendedor(vendedor)).thenThrow(excepcion);

}

ArrayList<TipoDocumento> tipos = **new** ArrayList<>();

tipos.add(tipoDocumento);

*when*(coordinadorMock.obtenerTiposDeDocumento()).thenReturn(tipos);

//Controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos que ingresaría el usuario en la vista

AltaVendedorController altaVendedorController = **new** AltaVendedorController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = **new** PresentadorVentanas();

**this**.presentador = presentadorVentanasMock;

setScenographyChanger(scenographyChangerMock);

**super**.inicializar(location, resources);

}

@Override

**public** **void** acceptAction() {

**this**.coordinador = coordinadorMock;

**this**.textFieldNombre.setText(nombre);

**this**.textFieldApellido.setText(apellido);

**this**.comboBoxTipoDocumento.getSelectionModel().select(tipoDocumento);

**this**.textFieldNumeroDocumento.setText(numeroDocumento);

**this**.passwordFieldContraseña.setText(contraseña);

**this**.passwordFieldRepiteContraseña.setText(contraseña);

**super**.acceptAction();

};

@Override

**protected** **void** setTitulo(String titulo) {

}

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AltaVendedorController.***URLVista***, altaVendedorController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Método a probar

altaVendedorController.acceptAction();

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(coordinadorMock).obtenerTiposDeDocumento();

Mockito.*verify*(coordinadorMock, *times*(llamaACrearVendedor)).crearVendedor(*any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcionInesperada)).presentarExcepcionInesperada(*eq*(excepcion), *any*());

Mockito.*verify*(scenographyChangerMock, *times*(llamaACambiarScene)).cambiarScenography(ModificarVendedorController.***URLVista***, **false**);

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

Código del archivo ModificarVendedorController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga al vendedor y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, éstos se muestran al usuario.

\*/

**public** **void** acceptAction() {

//Se toman los datos de la vista

String nombre = textFieldNombre.getText().trim();

String apellido = textFieldApellido.getText().trim();

String numeroDocumento = textFieldNumeroDocumento.getText().trim();

String passwordAntigua = passwordFieldContraseñaAntigua.getText();

String password1 = passwordFieldContraseñaNueva.getText();

String password2 = passwordFieldRepiteContraseña.getText();

TipoDocumento tipoDoc = comboBoxTipoDocumento.getValue();

StringBuilder error = **new** StringBuilder("");

//Se validan los campos

**if**(nombre.isEmpty()){

error.append("Inserte un nombre").append("\r\n");

}

**if**(apellido.isEmpty()){

error.append("Inserte un apellido").append("\r\n");

}

**if**(tipoDoc == **null**){

error.append("Elija un tipo de documento").append("\r\n");

}

**if**(numeroDocumento.isEmpty()){

error.append("Inserte un numero de documento").append("\r\n");

}

//Si el usuario quiere cambiar la contraseña, se validan también su contraseña antigua y nueva

**if**(checkBoxCambiarContraseña.isSelected()){

**if**(passwordAntigua.isEmpty()){

error.append("Ingrese su antigua contraseña\r\n");

}

**if**(!encriptador.encriptar(passwordAntigua.toCharArray(), vendedor.getSalt()).equals(vendedor.getPassword()) && !passwordAntigua.isEmpty()){

error.append("Su contraseña antigua es incorrecta\r\n");

}

**if**(password1.isEmpty() && password2.isEmpty()){

error.append("Ingrese su nueva contraseña").append("\r\n");

}

**if**(!password1.equals(password2)){

error.append("Sus nuevas contraseñas no coinciden. Ingréselas nuevamente").append("\r\n");

}

}

//Si hay errores se muestra una ventana con un mensaje explicativo de ellos

**if**(!error.toString().isEmpty()){

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

//Si no hay errores, se crea un vendedor para luego pasarselo a la capa lógica

vendedor.setNombre(nombre)

.setApellido(apellido)

.setNumeroDocumento(numeroDocumento)

.setTipoDocumento(tipoDoc);

**if**(checkBoxCambiarContraseña.isSelected()){

vendedor.setPassword(encriptador.encriptar(passwordFieldContraseñaNueva.getText().toCharArray(), vendedor.getSalt()));

}

ResultadoModificarVendedor resultadoModificarVendedor = **null**;

**try**{

//Se llama a la capa lógica para que cree el vendedor

resultadoModificarVendedor = coordinador.modificarVendedor(vendedor);

List<ErrorModificarVendedor> listaErrores = resultadoModificarVendedor.getErrores();

//Se convierten los errores devueltos a un mensaje a mostrar al usuario

**if**(listaErrores.contains(ErrorModificarVendedor.***Formato\_Nombre\_Incorrecto***)){

error.append("Nombre Incorrecto").append("\r\n");

}

**if**(listaErrores.contains(ErrorModificarVendedor.***Formato\_Apellido\_Incorrecto***)){

error.append("Apellido Incorrecto").append("\r\n");

}

**if**(listaErrores.contains(ErrorModificarVendedor.***Formato\_Documento\_Incorrecto***)){

error.append("Documento Incorrecto").append("\r\n");

}

**if**(listaErrores.contains(ErrorModificarVendedor.***Otro\_Vendedor\_Posee\_Mismo\_Documento\_Y\_Tipo***)){

error.append("Ya existe otro vendedor registrado con ese documento").append("\r\n");

}

//Si hay errores se muestran al usuario, si no, se presenta una notificación de éxito

**if**(!error.toString().isEmpty()){

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

/\*

\* primero sale, y después presenta el Toast

\* para saber en que posición colocarlo según el tamaño de la ventana padre (puede ser Login o administrar vendedor)

\*/

salir();

**if**(esAltaNuevamente){

presentador.presentarToast("Se ha dado de alta el vendedor con éxito", stage);

}

**else**{

presentador.presentarToast("Se ha modificado el vendedor con éxito", stage);

}

}

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e, stage);

}

}

}

Prueba de unidad del archivo ModificarVendedorControllerTest.java

//Casos de prueba

//nombre, apellido, tipoDocumento, numeroDocumento, resultadoEncriptar, contraseñaAntigua, contraseñaNueva, contraseñaNueva2, resultadoModificarVendedorEsperado, llamaAPresentadorVentanasPresentarError, llamaAPresentadorVentanasPresentarExcepcion, llamaAPresentadorVentanasPresentarExcepcionInesperada, llamaAModificarVendedor, excepcion, llamaACambiarScene, checkBoxSeleccionado

/\* 0 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", "abc", "abc", ***resultadoCorrecto***, 0, 0, 0, 1, **null**, 0, **false** }, //prueba correcta

/\* 1 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "ac", "abc", "abc", "abc", ***resultadoCorrecto***, 1, 0, 0, 0, **null**, 0, **true** }, //prueba contraseña antigua incorrecta

/\* 2 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", "abc", "abdc", ***resultadoCorrecto***, 1, 0, 0, 0, **null**, 0, **true** }, //prueba contraseñas nuevas no coinciden

/\* 3 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", "abc", "abc", ***resultadoModificarNombreIncorrecto***, 1, 0, 0, 1, **null**, 0, **false** }, //prueba nombre incorrecto

/\* 4 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", "abc", "abc", ***resultadoModificarApellidoIncorrecto***, 1, 0, 0, 1, **null**, 0, **false** }, //prueba apellido incorrecto

/\* 5 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", "abc", "abc", ***resultadoModificarDocumentoIncorrecto***, 1, 0, 0, 1, **null**, 0, **false** }, //prueba documento incorrecto

/\* 6 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", "abc", "abc", ***resultadoCrearYaExiste***, 1, 0, 0, 1, **null**, 0, **false** }, //prueba ya existe vendedor

/\* 7 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", "abc", "abc", **new** ResultadoModificarVendedor(ErrorModificarVendedor.***Formato\_Nombre\_Incorrecto***, ErrorModificarVendedor.***Formato\_Apellido\_Incorrecto***), 1, 0, 0, 1, **null**, 0, **false** }, //prueba nombre y apellido incorrectos

/\* 8 \*/ **new** Object[] { "", "Perez", doc, "12345678", "abc", "abc", "abc", "abc", **null**, 1, 0, 0, 0, **null**, 0, **false** }, //prueba nombre vacio

/\* 9 \*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "abc", "abc", "abc", "abc", ***resultadoCorrecto***, 0, 1, 0, 1, **new** SaveUpdateException(**new** Throwable()), 0, **false** }, //prueba PersistenciaException

/\*\*

\* **@param** nombre

\* nombre del vendedor a modificar

\* **@param** apellido

\* apellido del vendedor a modificar

\* **@param** tipoDocumento

\* tipo de documenteo del vendedor a modificar

\* **@param** numeroDocumento

\* número de documento del vendedor a modificar

\* **@param** resultadoEncriptar

\* resultado devuelto por el mock encriptadorPassword

\* **@param** contraseñaAntigua

\* contraseña antigua del vendedor

\* **@param** contraseñaNueva

\* contraseña nueva del vendedor

\* **@param** contraseñaNueva2

\* campo repetir contraseña del vendedor

\* **@param** resultadoModificarVendedorEsperado

\* resultado devuelto por la lógica

\* **@param** llamaAPresentadorVentanasPresentarError

\* indica si se debe llamar a presentar una ventana de error

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* indica si se debe llamar a presentar una ventana de excepción

\* **@param** llamaAPresentadorVentanasPresentarExcepcionInesperada

\* indica si se debe llamar a presentar una ventana de excepción inesperada

\* **@param** llamaAModificarVendedor

\* indica si se debe llamar a modificar un vendedor en la capa lógica

\* **@param** excepcion

\* excepción devuelta por la capa lógica

\* **@param** llamaACambiarScene

\* indica si se debe llamar a cambiar de pantalla

\* **@param** checkBoxSeleccionado

\* checkBox de cambiar contraseña seleccionado o no seleccionado

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testModificarVendedor(String nombre, String apellido, TipoDocumento tipoDocumento, String numeroDocumento, String resultadoEncriptar, String contraseñaAntigua, String contraseñaNueva, String contraseñaNueva2, ResultadoModificarVendedor resultadoModificarVendedorEsperado, Integer llamaAPresentadorVentanasPresentarError, Integer llamaAPresentadorVentanasPresentarExcepcion, Integer llamaAPresentadorVentanasPresentarExcepcionInesperada, Integer llamaAModificarVendedor, Exception excepcion, Integer llamaACambiarScene, Boolean checkBoxSeleccionado) **throws** Throwable {

//Se crean los mocks necesarios

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaErrorExcepcionInesperada ventanaErrorExcepcionInesperadaMock = *mock*(VentanaErrorExcepcionInesperada.**class**);

EncriptadorPassword encriptadorPasswordMock = *mock*(EncriptadorPassword.**class**);

//Se setea lo que deben devolver los mocks cuando son invocados por la clase a probar

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarExcepcionInesperada(*any*(), *any*())).thenReturn(ventanaErrorExcepcionInesperadaMock);

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

*when*(encriptadorPasswordMock.encriptar(*eq*(contraseñaAntigua.toCharArray()), *any*())).thenReturn(resultadoEncriptar);

Vendedor vendedor = **new** Vendedor()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setPassword(contraseñaAntigua);

*when*(coordinadorMock.modificarVendedor(vendedor)).thenReturn(resultadoModificarVendedorEsperado);

**if**(excepcion != **null**){

*when*(coordinadorMock.modificarVendedor(vendedor)).thenThrow(excepcion);

}

ArrayList<TipoDocumento> tipos = **new** ArrayList<>();

tipos.add(tipoDocumento);

*when*(coordinadorMock.obtenerTiposDeDocumento()).thenReturn(tipos);

//Controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos que ingresaría el usuario en la vista

ModificarVendedorController modificarVendedorController = **new** ModificarVendedorController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

**this**.encriptador = encriptadorPasswordMock;

**this**.setVendedor(vendedor);

**super**.inicializar(location, resources);

}

@Override

**public** **void** acceptAction() {

**this**.textFieldNombre.setText(nombre);

**this**.textFieldApellido.setText(apellido);

**this**.comboBoxTipoDocumento.getSelectionModel().select(tipoDocumento);

**this**.textFieldNumeroDocumento.setText(numeroDocumento);

**this**.passwordFieldContraseñaAntigua.setText(contraseñaAntigua);

**this**.passwordFieldContraseñaNueva.setText(contraseñaNueva);

**this**.passwordFieldRepiteContraseña.setText(contraseñaNueva2);

**this**.checkBoxCambiarContraseña.setSelected(checkBoxSeleccionado);

**super**.acceptAction();

};

@Override

**protected** **void** setTitulo(String titulo) {

}

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(ModificarVendedorController.***URLVista***, modificarVendedorController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Método a probar

modificarVendedorController.acceptAction();

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(coordinadorMock).obtenerTiposDeDocumento();

Mockito.*verify*(coordinadorMock, *times*(llamaAModificarVendedor)).modificarVendedor(*any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcionInesperada)).presentarExcepcionInesperada(*eq*(excepcion), *any*());

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

### Taskcard 4 Vista principal para la administración

Código del archivo BaseController.java

/\*\*

\* Método que se llama al hacer click en el botón ver mis datos

\*/

@FXML

public void verMisDatos() {

//Cambia a la pantalla de modificar vendedor

ModificarVendedorController nuevaPantalla = (ModificarVendedorController) cambiarScene(background, ModificarVendedorController.URLVista, ventanaInicio);

//Le seteamos el vendedor logueado para que vea y modifique sus datos

nuevaPantalla.setVendedor(vendedorLogueado);

//Le permitimos ver sus ventas

nuevaPantalla.mostrarBotonVerVentas();

}

/\*\*

\* Método que se llama al hacer click en el botón clientes

\*/

@FXML

public void verClientes() {

//Cambia a la pantalla de administración de clientes

cambiarScene(background, AdministrarClienteController.URLVista);

}

/\*\*

\* Método que se llama al hacer click en el botón inmuebles

\*/

@FXML

public void verInmuebles() {

//Cambia a la pantalla de administración de clientes

cambiarScene(background, AdministrarInmuebleController.URLVista);

}

/\*\*

\* Método que se llama al hacer click en el botón vendedores

\*/

@FXML

public void verVendedores() {

//Cambia a la pantalla de administración de vendedores

cambiarScene(background, AdministrarVendedorController.URLVista);

}

/\*\*

\* Método que se llama al hacer click en el botón propietarios

\*/

@FXML

public void verPropietarios() {

//Cambia a la pantalla de administración de propietarios

cambiarScene(background, AdministrarPropietarioController.URLVista);

}

### Taskcard 5 Lógica alta, modificación y baja vendedor

Código del archivo GestorVendedor.java

/\*\*

\* Se encarga de validar los datos de un vendedor a crear y, en caso de que no haya errores,

\* delegar el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** vendedor

\* vendedor a crear

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\* **@throws** GestionException

\* se lanza una excepción EntidadExistenteConEstadoBaja cuando se encuentra que ya existe un vendedor con la misma identificación pero tiene estado BAJA

\*/

**public** ResultadoCrearVendedor crearVendedor(Vendedor vendedor) **throws** PersistenciaException, GestionException {

ArrayList<ErrorCrearVendedor> errores = **new** ArrayList<>();

// valida formato de datos

**if**(!validador.validarNombre(vendedor.getNombre())){

errores.add(ErrorCrearVendedor.***Formato\_Nombre\_Incorrecto***);

}

**if**(!validador.validarApellido(vendedor.getApellido())){

errores.add(ErrorCrearVendedor.***Formato\_Apellido\_Incorrecto***);

}

**if**(!validador.validarDocumento(vendedor.getTipoDocumento(), vendedor.getNumeroDocumento())){

errores.add(ErrorCrearVendedor.***Formato\_Documento\_Incorrecto***);

}

//valida si existe un vendedor con ese tipo y número de documento

Vendedor vendedorAuxiliar = persistidorVendedor.obtenerVendedor(**new** FiltroVendedor(vendedor.getTipoDocumento().getTipo(), vendedor.getNumeroDocumento()));

**if**(**null** != vendedorAuxiliar){

**if**(vendedorAuxiliar.getEstado().getEstado().equals(EstadoStr.***ALTA***)){

errores.add(ErrorCrearVendedor.***Ya\_Existe\_Vendedor***); // si existe y tiene estado alta

}

**else**{ // si existe y tiene estado baja

**throw** **new** EntidadExistenteConEstadoBajaException();

}

}

**if**(errores.isEmpty()){ //si no hay errores

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***ALTA***)){

vendedor.setEstado(e); //seteo el estado en alta

}

}

persistidorVendedor.guardarVendedor(vendedor);

}

**return** **new** ResultadoCrearVendedor(errores.toArray(**new** ErrorCrearVendedor[0]));

}

/\*\*

\* Se encarga de validar los datos de un vendedor a modificar y, en caso de que no haya errores,

\* delegar el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** vendedor

\* vendedor a modificar

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ResultadoModificarVendedor modificarVendedor(Vendedor vendedor) **throws** PersistenciaException {

ArrayList<ErrorModificarVendedor> errores = **new** ArrayList<>();

// valida formato de datos

**if**(!validador.validarNombre(vendedor.getNombre())){

errores.add(ErrorModificarVendedor.***Formato\_Nombre\_Incorrecto***);

}

**if**(!validador.validarApellido(vendedor.getApellido())){

errores.add(ErrorModificarVendedor.***Formato\_Apellido\_Incorrecto***);

}

**if**(!validador.validarDocumento(vendedor.getTipoDocumento(), vendedor.getNumeroDocumento())){

errores.add(ErrorModificarVendedor.***Formato\_Documento\_Incorrecto***);

}

//verifica si existe otro vendedor con los nuevos tipo y número de documento

Vendedor vendedorAuxiliar = persistidorVendedor.obtenerVendedor(**new** FiltroVendedor(vendedor.getTipoDocumento().getTipo(), vendedor.getNumeroDocumento()));

**if**(vendedorAuxiliar != **null** && !vendedor.equals(vendedorAuxiliar)){

errores.add(ErrorModificarVendedor.***Otro\_Vendedor\_Posee\_Mismo\_Documento\_Y\_Tipo***);

}

**if**(errores.isEmpty()){ //si no hay errores

**if**(vendedor.getEstado().getEstado().equals(EstadoStr.***BAJA***)){

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***ALTA***)){

vendedor.setEstado(e); //si el estado es baja, se setea en alta

}

}

}

persistidorVendedor.modificarVendedor(vendedor);

}

**return** **new** ResultadoModificarVendedor(errores.toArray(**new** ErrorModificarVendedor[0]));

}

/\*\*

\* Se encarga de validar que exista el vendedor a eliminar, se setea el estado en BAJA y,

\* en caso de que no haya errores, delegar el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** vendedor

\* vendedor a eliminar

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ResultadoEliminarVendedor eliminarVendedor(Vendedor vendedor) **throws** PersistenciaException {

//se setea el estado en baja y se manda a guardar

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***BAJA***)){

vendedor.setEstado(e);

}

}

persistidorVendedor.modificarVendedor(vendedor);

**return** **new** ResultadoEliminarVendedor();

}

Código del test GestorVendedorTest.java

**protected** Object[] parametersForTestCrearVendedor() {

//Se carga vendedor con datos básicos solo para evitar punteros nulos

//Las validaciones se hacen en el validador, por lo que se setean luego en los mocks los valores esperados

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

*vendedor* = **new** Vendedor();

*vendedor*.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setSalt("abcd")

.setPassword("1234");

//Parámetros de JUnitParams

**return** **new** Object[] {

//validoNombre,validoApellido,validoDocumento,resultadoObtenerVendedor,guardar,resultadoEsperado

/\*0\*/ **new** Object[] { **true**, **true**, **true**, **null**, 1, ***resultadoCorrecto*** },

/\*1\*/ **new** Object[] { **false**, **true**, **true**, **null**, 0, ***resultadoCrearNombreIncorrecto*** },

/\*2\*/ **new** Object[] { **true**, **false**, **true**, **null**, 0, ***resultadoCrearApellidoIncorrecto*** },

/\*3\*/ **new** Object[] { **true**, **true**, **false**, **null**, 0, ***resultadoCrearDocumentoIncorrecto*** },

/\*4\*/ **new** Object[] { **false**, **false**, **true**, **null**, 0, **new** ResultadoCrearVendedor(ErrorCrearVendedor.***Formato\_Nombre\_Incorrecto***, ErrorCrearVendedor.***Formato\_Apellido\_Incorrecto***) },

/\*5\*/ **new** Object[] { **true**, **true**, **true**, *vendedor*, 0, ***resultadoCrearYaExiste*** }

};

}

/\*\*

\* Test para la función crearVendedor

\*

\* **@param** resValNombre

\* resultado que va a devolver el validador mock al validar nombre

\* **@param** resValApellido

\* resultado que va a devolver el validador mock al validar apellido

\* **@param** resValDocumento

\* resultado que va a devolver el validador mock al validar documento

\* **@param** resObtenerVendedor

\* resultado que va a devolver el persistidor mock al obtener vendedor

\* **@param** guardar

\* 1 si va a guardar, 0 si no

\* **@param** resultadoCrearVendedorEsperado

\* resultado esperado que retorne el método a probar

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testCrearVendedor(Boolean resValNombre, Boolean resValApellido, Boolean resValDocumento, Vendedor resObtenerVendedor, Integer guardar, ResultadoCrearVendedor resultadoCrearVendedorEsperado) **throws** Exception {

//Inicialización de los mocks

VendedorService vendedorServiceMock = *mock*(VendedorService.**class**);

ValidadorFormato validadorFormatoMock = *mock*(ValidadorFormato.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

//Clase anónima necesaria para inyectar dependencias

GestorVendedor gestorVendedor = **new** GestorVendedor() {

{

**this**.persistidorVendedor = vendedorServiceMock;

**this**.validador = validadorFormatoMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***ALTA***));

//Setear valores esperados a los mocks

*when*(validadorFormatoMock.validarNombre(*vendedor*.getNombre())).thenReturn(resValNombre);

*when*(validadorFormatoMock.validarApellido(*vendedor*.getApellido())).thenReturn(resValApellido);

*when*(validadorFormatoMock.validarDocumento(*vendedor*.getTipoDocumento(), *vendedor*.getNumeroDocumento())).thenReturn(resValDocumento);

*when*(vendedorServiceMock.obtenerVendedor(*any*())).thenReturn(resObtenerVendedor);

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

*doNothing*().when(vendedorServiceMock).guardarVendedor(*vendedor*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoCrearVendedor resultadoCrearVendedor = gestorVendedor.crearVendedor(*vendedor*);

//Comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

*assertEquals*(resultadoCrearVendedorEsperado, resultadoCrearVendedor);

**if**(guardar.equals(1)){

*assertEquals*(EstadoStr.***ALTA***, *vendedor*.getEstado().getEstado());

}

*verify*(validadorFormatoMock).validarNombre(*vendedor*.getNombre());

*verify*(validadorFormatoMock).validarApellido(*vendedor*.getApellido());

*verify*(validadorFormatoMock).validarDocumento(*vendedor*.getTipoDocumento(), *vendedor*.getNumeroDocumento());

*verify*(gestorDatosMock, *times*(guardar)).obtenerEstados();

*verify*(vendedorServiceMock, *times*(guardar)).guardarVendedor(*vendedor*);

}

**protected** Object[] parametersForTestModificarVendedor() {

//Se carga vendedor con datos básicos solo para evitar punteros nulos

//Las validaciones se hacen en el validador, por lo que se setean luego en los mocks los valores esperados

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

Estado est = **new** Estado(EstadoStr.***ALTA***);

*vendedorM* = **new** Vendedor() {

**public** Vendedor setId(Integer id) {

**this**.id = id;

**return** **this**;

}

{

**this**.setId(1)

.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setSalt("abcd")

.setPassword("1234")

.setEstado(est);

}

};

*vendedorM2* = **new** Vendedor() {

**public** Vendedor setId(Integer id) {

**this**.id = id;

**return** **this**;

}

{

**this**.setId(2)

.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setSalt("abcd")

.setPassword("1234");

}

};

//Parámetros de JUnitParams

**return** **new** Object[] {

//validoNombre,validoApellido,validoDocumento,resultadoObtenerVendedor,modificar,resultadoEsperado

/\*0\*/ **new** Object[] { **true**, **true**, **true**, *vendedorM*, 1, ***resultadoCorrectoModificar*** },

/\*0\*/ **new** Object[] { **false**, **true**, **true**, *vendedorM*, 0, ***resultadoModificarNombreIncorrecto*** },

/\*0\*/ **new** Object[] { **true**, **false**, **true**, *vendedorM*, 0, ***resultadoModificarApellidoIncorrecto*** },

/\*0\*/ **new** Object[] { **true**, **true**, **false**, *vendedorM*, 0, ***resultadoModificarDocumentoIncorrecto*** },

/\*0\*/ **new** Object[] { **false**, **false**, **true**, *vendedorM*, 0, **new** ResultadoModificarVendedor(ErrorModificarVendedor.***Formato\_Nombre\_Incorrecto***, ErrorModificarVendedor.***Formato\_Apellido\_Incorrecto***) },

/\*0\*/ **new** Object[] { **true**, **true**, **true**, *vendedorM2*, 0, ***resultadoModificarYaExiste*** }

};

}

/\*\*

\* Test para la función modificarVendedor

\*

\* **@param** resValNombre

\* resultado que va a devolver el validador mock al validar nombre

\* **@param** resValApellido

\* resultado que va a devolver el validador mock al validar apellido

\* **@param** resValDocumento

\* resultado que va a devolver el validador mock al validar documento

\* **@param** resObtenerVendedor

\* resultado que va a devolver el persistidor mock al obtener vendedor

\* **@param** modificar

\* 1 si va a modificar, 0 si no

\* **@param** resultadoModificarVendedorEsperado

\* resultado esperado que retorne el método a probar

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testModificarVendedor(Boolean resValNombre, Boolean resValApellido, Boolean resValDocumento, Vendedor resObtenerVendedor, Integer modificar, ResultadoModificarVendedor resultadoModificarVendedorEsperado) **throws** Exception {

//Inicialización de los mocks

VendedorService vendedorServiceMock = *mock*(VendedorService.**class**);

ValidadorFormato validadorFormatoMock = *mock*(ValidadorFormato.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

//Clase anónima necesaria para inyectar dependencias

GestorVendedor gestorVendedor = **new** GestorVendedor() {

{

**this**.persistidorVendedor = vendedorServiceMock;

**this**.validador = validadorFormatoMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***ALTA***));

//Setear valores esperados a los mocks

*when*(validadorFormatoMock.validarNombre(*vendedorM*.getNombre())).thenReturn(resValNombre);

*when*(validadorFormatoMock.validarApellido(*vendedorM*.getApellido())).thenReturn(resValApellido);

*when*(validadorFormatoMock.validarDocumento(*vendedorM*.getTipoDocumento(), *vendedor*.getNumeroDocumento())).thenReturn(resValDocumento);

*when*(vendedorServiceMock.obtenerVendedor(*any*())).thenReturn(resObtenerVendedor);

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

*doNothing*().when(vendedorServiceMock).modificarVendedor(*vendedorM*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoModificarVendedor resultadoModificarVendedor = gestorVendedor.modificarVendedor(*vendedorM*);

//Comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

*assertEquals*(resultadoModificarVendedorEsperado, resultadoModificarVendedor);

**if**(modificar.equals(1)){

*assertEquals*(EstadoStr.***ALTA***, *vendedorM*.getEstado().getEstado());

}

*verify*(validadorFormatoMock).validarNombre(*vendedorM*.getNombre());

*verify*(validadorFormatoMock).validarApellido(*vendedorM*.getApellido());

*verify*(validadorFormatoMock).validarDocumento(*vendedorM*.getTipoDocumento(), *vendedorM*.getNumeroDocumento());

*verify*(vendedorServiceMock, *times*(modificar)).modificarVendedor(*vendedorM*);

}

**protected** Object[] parametersForTestEliminarVendedor() {

//Se carga vendedor con datos básicos solo para evitar punteros nulos

//Las validaciones se hacen en el validador, por lo que se setean luego en los mocks los valores esperados

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

Estado est = **new** Estado(EstadoStr.***ALTA***);

*vendedorE* = **new** Vendedor();

*vendedorE*.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setSalt("abcd")

.setPassword("1234")

.setEstado(est);

//Parámetros de JUnitParams

**return** **new** Object[] {

//resultadoObtenerVendedor, eliminar, resultadoEsperado

/\*0\*/ **new** Object[] { *vendedorE*, 1, ***resultadoCorrectoEliminar*** },

};

}

@Test

@Parameters

**public** **void** testEliminarVendedor(Vendedor resObtenerVendedor, Integer eliminar, ResultadoEliminarVendedor resultadoEliminarVendedorEsperado) **throws** Exception {

//Inicialización de los mocks

VendedorService vendedorServiceMock = *mock*(VendedorService.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

//Clase anónima necesaria para inyectar dependencias

GestorVendedor gestorVendedor = **new** GestorVendedor() {

{

**this**.persistidorVendedor = vendedorServiceMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***BAJA***));

//Setear valores esperados a los mocks

*when*(vendedorServiceMock.obtenerVendedor(*any*())).thenReturn(resObtenerVendedor);

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

*doNothing*().when(vendedorServiceMock).modificarVendedor(*vendedorE*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoEliminarVendedor resultadoEliminarVendedor = gestorVendedor.eliminarVendedor(*vendedorE*);

//Comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

*assertEquals*(resultadoEliminarVendedorEsperado, resultadoEliminarVendedor);

**if**(eliminar.equals(1)){

*assertEquals*(EstadoStr.***BAJA***, *vendedorE*.getEstado().getEstado());

}

*verify*(gestorDatosMock, *times*(eliminar)).obtenerEstados();

*verify*(vendedorServiceMock, *times*(eliminar)).modificarVendedor(*vendedorE*);

}

### Taskcard 6 Persistidor vendedor

Código del archivo en VendedorServiceImpl.java

/\*\*

\* Permite persistir y obtener un vendedor, y listar vendedores

\* Task card 6 de la iteración 1, historia de usuario 1

\*

\*/

@Repository

**public** **class** VendedorServiceImpl **implements** VendedorService {

**private** SessionFactory sessionFactory;

@Autowired

**public** VendedorServiceImpl(SessionFactory sessionFactory) {

**this**.sessionFactory = sessionFactory;

}

**public** SessionFactory getSessionFactory() {

**return** sessionFactory;

}

@Override

@Transactional(rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** **void** guardarVendedor(Vendedor vendedor) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.save(vendedor);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

@Override

@Transactional(rollbackFor = PersistenciaException.**class**)//si falla hace rollback de la transacción

**public** **void** modificarVendedor(Vendedor vendedor) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.update(vendedor);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

@Override

@Transactional(readOnly = **true**, rollbackFor = PersistenciaException.**class**)

**public** Vendedor obtenerVendedor(FiltroVendedor filtro) **throws** PersistenciaException {

Vendedor vendedor = **null**;

Session session = getSessionFactory().getCurrentSession();

**try**{ //named query ubicada en entidad vendedor

vendedor = (Vendedor) session.getNamedQuery("obtenerVendedor").setParameter("tipoDocumento", filtro.getTipoDocumento()).setParameter("documento", filtro.getDocumento()).uniqueResult();

} **catch**(NoResultException e){

**return** **null**;

} **catch**(NonUniqueResultException e){

**return** **null**;

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** vendedor;

}

@Override

@Transactional(readOnly = **true**, rollbackFor = PersistenciaException.**class**)

**public** ArrayList<Vendedor> listarVendedores() **throws** PersistenciaException {

ArrayList<Vendedor> vendedores = **new** ArrayList<>();

Session session = getSessionFactory().getCurrentSession();

**try**{ //named query ubicada en entidad vendedor

**for**(Object o: session.getNamedQuery("listarVendedores").list()){

**if**(o **instanceof** Vendedor){

vendedores.add((Vendedor) o);

}

}

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** vendedores;

}

}

### Taskcard 7 Entidad Propietario

Código del archivo Propietario.java

@NamedQueries(value = { @NamedQuery(name = "obtenerPropietarios", query = "SELECT p FROM Propietario p WHERE p.estado.estado = 'ALTA'"),

@NamedQuery(name = "obtenerPropietario", query = "SELECT p FROM Propietario p WHERE p.numeroDocumento = :documento AND p.tipoDocumento.tipo = :tipoDocumento") })

@Entity

@Table(name = "propietario", uniqueConstraints = @UniqueConstraint(name = "propietario\_numerodocumento\_idtipo\_uk", columnNames = { "numerodocumento", "idtipo" }))

/\*\*

\* Entidad que modela a un propietario

\*/

**public** **class** Propietario {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "nombre", length = 30)

**private** String nombre;

@Column(name = "apellido", length = 30)

**private** String apellido;

@Column(name = "numerodocumento", length = 30)

**private** String numeroDocumento;

@Column(name = "telefono", length = 30)

**private** String telefono;

@Column(name = "email", length = 30)

**private** String email;

//Reclaciones

@ManyToOne(fetch = FetchType.***LAZY***)

@JoinColumn(name = "idtipo", referencedColumnName = "id", foreignKey = @ForeignKey(name = "propietario\_idtipo\_fk"))

**private** TipoDocumento tipoDocumento;

@ManyToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "iddireccion", referencedColumnName = "id", foreignKey = @ForeignKey(name = "propietario\_iddireccion\_fk"))

**private** Direccion direccion;

//Opcionales

@OneToMany(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***, orphanRemoval = **true**, mappedBy = "propietario")

**private** Set<Inmueble> inmuebles;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idestado", referencedColumnName = "id", foreignKey = @ForeignKey(name = "propietario\_idestado\_fk"), nullable = **false**)

**private** Estado estado;

### Taskcard 8 Vista alta, modificar y baja propietario

Código del alta en AltaPropietarioController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga al propietario y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, se muestran al usuario.

\*/

@FXML

**public** **void** acceptAction() {

StringBuilder error = **new** StringBuilder("");

//obtengo datos introducidos por el usuario

String nombre = textFieldNombre.getText().trim();

String apellido = textFieldApellido.getText().trim();

String numeroDocumento = textFieldNumeroDocumento.getText().trim();

String alturaCalle = textFieldAlturaCalle.getText().trim();

String piso = textFieldPiso.getText().trim();

String departamento = textFieldDepartamento.getText().trim();

String telefono = textFieldTelefono.getText().trim();

String correoElectronico = textFieldCorreoElectronico.getText().trim();

String otros = textFieldOtros.getText().trim();

Localidad localidad = comboBoxLocalidad.getValue();

TipoDocumento tipoDoc = comboBoxTipoDocumento.getValue();

Barrio barrio = comboBoxBarrio.getValue();

Calle calle = comboBoxCalle.getValue();

//verifico que no estén vacíos

**if**(nombre.isEmpty()){

error.append("Inserte un nombre").append("\n");

}

**if**(apellido.isEmpty()){

error.append("Inserte un apellido").append("\n ");

}

**if**(numeroDocumento.isEmpty()){

error.append("Inserte un numero de documento").append("\n ");

}

**if**(alturaCalle.isEmpty()){

error.append("Inserte una altura").append("\n ");

}

**if**(telefono.isEmpty()){

error.append("Inserte un telefono").append("\n ");

}

**if**(correoElectronico.isEmpty()){

error.append("Inserte un correo electrónico").append("\n ");

}

**if**(calle == **null**){

error.append("Elija una calle").append("\n");

}

**if**(barrio == **null**){

error.append("Elija un barrio").append("\n");

}

**if**(tipoDoc == **null**){

error.append("Elija un tipo de documento").append("\n");

}

**if**(localidad == **null**){

error.append("Elija una localidad").append("\n");

}

**if**(!error.toString().isEmpty()){ //si hay algún error lo muestro al usuario

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

//Si no hay errores se crean las entidades con los datos introducidos

Direccion direccion = **new** Direccion();

direccion.setNumero(alturaCalle)

.setCalle(calle)

.setBarrio(barrio)

.setPiso(piso)

.setDepartamento(departamento)

.setLocalidad(localidad)

.setOtros(otros);

Propietario propietario = **new** Propietario();

propietario.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setEmail(correoElectronico)

.setDireccion(direccion);

**try**{ //relevo la operación a capa lógica

ResultadoCrearPropietario resultado = coordinador.crearPropietario(propietario);

**if**(resultado.hayErrores()){ // si hay algún error se muestra al usuario

StringBuilder stringErrores = **new** StringBuilder();

**for**(ErrorCrearPropietario err: resultado.getErrores()){

**switch**(err) {

**case** ***Formato\_Nombre\_Incorrecto***:

stringErrores.append("Formato de nombre incorrecto.\n");

**break**;

**case** ***Formato\_Apellido\_Incorrecto***:

stringErrores.append("Formato de apellido incorrecto.\n");

**break**;

**case** ***Formato\_Telefono\_Incorrecto***:

stringErrores.append("Formato de teléfono incorrecto.\n");

**break**;

**case** ***Formato\_Documento\_Incorrecto***:

stringErrores.append("Tipo y formato de documento incorrecto.\n");

**break**;

**case** ***Formato\_Email\_Incorrecto***:

stringErrores.append("Formato de email incorrecto.\n");

**break**;

**case** ***Formato\_Direccion\_Incorrecto***:

stringErrores.append("Formato de dirección incorrecto.\n");

**break**;

**case** ***Ya\_Existe\_Propietario***:

stringErrores.append("Ya existe un cliente con ese tipo y número de documento.\n");

**break**;

}

}

presentador.presentarError("Revise sus campos", stringErrores.toString(), stage);

}

**else**{ //si no hay errores se muestra notificación y se vuelve a la pantalla de listar propietarios

presentador.presentarToast("Se ha creado el propietario con éxito", stage);

cambiarmeAScene(AdministrarPropietarioController.***URLVista***);

}

} **catch**(GestionException e){ //excepción originada en gestor

**if**(e.getClass().equals(EntidadExistenteConEstadoBajaException.**class**)){

//el propietario existe pero fué dado de baja

VentanaConfirmacion ventana = presentador.presentarConfirmacion("El propietario ya existe", "El propietario ya existía anteriormente pero fué dado de baja.\n ¿Desea volver a darle de alta?", stage);

**if**(ventana.acepta()){

//usuario acepta volver a darle de alta. Se pasa a la pantalla de modificar propietario

ModificarPropietarioController controlador = (ModificarPropietarioController) cambiarmeAScene(ModificarPropietarioController.***URLVista***);

controlador.setPropietarioEnModificacion(propietario);

}

}

} **catch**(PersistenciaException e){ //excepción originada en la capa de persistencia

presentador.presentarExcepcion(e, stage);

}

}

}

Código del test del alta en AltaPropietarioControllerTest.java

**protected** Object[] parametersForTestCrearPropietario() {

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

Pais pais = **new** Pais("Argentina");

Provincia provincia = **new** Provincia("Santa fé", pais);

Localidad localidad = **new** Localidad("Capital", provincia);

Direccion dir = **new** Direccion("1865", "1", "B", **new** Calle("San martín", localidad), **new** Barrio("Centro", localidad), localidad, "");

**return** **new** Object[] {

//nombre,apellido,tipoDocumento,nroDocumento,telefono,email,direccion,resultadoEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,llamaACrearPropietario,excepcion,aceptarVentanaConfirmacion,llamaACambiarScene

//prueba correcta

/\*0\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 0, 1, **null**, **true**, 0 },

//prueba nombre incorrecto

/\*1\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearNombreIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba apellido incorrecto

/\*2\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearApellidoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba documento incorrecto

/\*3\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearDocumentoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba teléfono incorrecto

/\*4\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearTelefonoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba email incorrecto

/\*5\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearEmailIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba direccion incorrecta

/\*6\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearDireccionIncorrecta***, 1, 0, 1, **null**, **true**, 0 },

//prueba ya existe propietario

/\*7\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearYaExiste***, 1, 0, 1, **null**, **true**, 0 },

//prueba ya existe propietario

/\*8\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, **new** ResultadoCrearPropietario(ErrorCrearPropietario.***Formato\_Nombre\_Incorrecto***, ErrorCrearPropietario.***Formato\_Apellido\_Incorrecto***), 1, 0, 1, **null**, **true**, 0 },

//prueba nombre vacio

/\*9\*/ **new** Object[] { "", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, **null**, 1, 0, 0, **null**, **true**, 0 },

//prueba propietario Existente y acepta

/\*10\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **true**, 1 },

//prueba propietario Existente y cancela

/\*11\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **false**, 0 },

//prueba PersistenciaException

/\*12\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 1, 1, **new** SaveUpdateException(**new** Throwable()), **false**, 0 }

};

}

/\*\*

\* Test para probar al controlador de alta propietario cuando el usuario presiona aceptar

\*

\* **@param** nombre

\* nombre que se introduce por el usuario

\* **@param** apellido

\* apellido que se introduce por el usuario

\* **@param** tipoDocumento

\* tipo de documento que se introduce por el usuario

\* **@param** numeroDocumento

\* número de documento que se introduce por el usuario

\* **@param** telefono

\* teléfono que se introduce por el usuario

\* **@param** email

\* email que se introduce por el usuario

\* **@param** direccion

\* dirección que se introduce por el usuario

\* **@param** resultadoCrearPropietarioEsperado

\* resultado que retornará el mock de capa lógica

\* **@param** llamaAPresentadorVentanasPresentarError

\* 1 si llama al método presentar error del presentador de ventanas, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* 1 si llama al método presentar excepción del presentador de ventanas, 0 si no

\* **@param** llamaACrearPropietario

\* 1 si llama al método crear propietario de la capa lógica, 0 si no

\* **@param** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@param** aceptarVentanaConfirmacion

\* si el usuario acepta la ventana de confirmación

\* **@param** llamaACambiarScene

\* 1 si llama al método cambiar scene, 0 si no

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCrearPropietario(String nombre,

String apellido,

TipoDocumento tipoDocumento,

String numeroDocumento,

String telefono,

String email,

Direccion direccion,

ResultadoCrearPropietario resultadoCrearPropietarioEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Integer llamaACrearPropietario,

Exception excepcion,

Boolean aceptarVentanaConfirmacion,

Integer llamaACambiarScene) **throws** Throwable {

CoordinadorJavaFX coordinadorMock = Mockito.*mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaConfirmacion ventanaConfirmacionMock = *mock*(VentanaConfirmacion.**class**);

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

ModificarPropietarioController modificarPropietarioControllerMock = *mock*(ModificarPropietarioController.**class**);

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarConfirmacion(*any*(), *any*(), *any*())).thenReturn(ventanaConfirmacionMock);

*when*(ventanaConfirmacionMock.acepta()).thenReturn(aceptarVentanaConfirmacion);

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(modificarPropietarioControllerMock);

*doNothing*().when(modificarPropietarioControllerMock).setPropietarioEnModificacion(*any*());

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

*propietario* = **new** Propietario()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setEmail(email)

.setDireccion(direccion);

*when*(coordinadorMock.crearPropietario(*propietario*)).thenReturn(resultadoCrearPropietarioEsperado);

**if**(excepcion != **null**){

*when*(coordinadorMock.crearPropietario(*propietario*)).thenThrow(excepcion);

}

ArrayList<TipoDocumento> tipos = **new** ArrayList<>();

tipos.add(tipoDocumento);

Mockito.*when*(coordinadorMock.obtenerTiposDeDocumento()).thenReturn(tipos);

AltaPropietarioController altaPropietarioController = **new** AltaPropietarioController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

setScenographyChanger(scenographyChangerMock);

**super**.inicializar(location, resources);

}

@Override

**public** **void** acceptAction() {

**this**.textFieldNombre.setText(nombre);

**this**.textFieldApellido.setText(apellido);

**this**.comboBoxTipoDocumento.getSelectionModel().select(tipoDocumento);

**this**.textFieldNumeroDocumento.setText(numeroDocumento);

**this**.textFieldTelefono.setText(telefono);

**this**.textFieldCorreoElectronico.setText(email);

**this**.comboBoxPais.getSelectionModel().select(direccion.getLocalidad().getProvincia().getPais());

**this**.comboBoxProvincia.getSelectionModel().select(direccion.getLocalidad().getProvincia());

**this**.comboBoxLocalidad.getSelectionModel().select(direccion.getLocalidad());

**this**.comboBoxBarrio.getSelectionModel().select(direccion.getBarrio());

**this**.comboBoxCalle.getSelectionModel().select(direccion.getCalle());

**this**.textFieldAlturaCalle.setText(direccion.getNumero());

**this**.textFieldDepartamento.setText(direccion.getDepartamento());

**this**.textFieldPiso.setText(direccion.getPiso());

**this**.textFieldOtros.setText(direccion.getOtros());

**super**.acceptAction();

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(AltaPropietarioController.***URLVista***, altaPropietarioController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

altaPropietarioController.acceptAction();

Mockito.*verify*(coordinadorMock).obtenerTiposDeDocumento();

Mockito.*verify*(coordinadorMock, Mockito.*times*(llamaACrearPropietario)).crearPropietario(Mockito.*any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

Mockito.*verify*(scenographyChangerMock, *times*(llamaACambiarScene)).cambiarScenography(ModificarPropietarioController.***URLVista***, **false**);

Mockito.*verify*(modificarPropietarioControllerMock, *times*(llamaACambiarScene)).setPropietarioEnModificacion(*propietario*);

}

};

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

Código del modificar en ModificarPropietarioController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga al propietario y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, se muestran al usuario.

\*/

@FXML

**public** **void** acceptAction() {

StringBuilder error = **new** StringBuilder("");

//obtengo datos introducidos por el usuario

String nombre = textFieldNombre.getText().trim();

String apellido = textFieldApellido.getText().trim();

String numeroDocumento = textFieldNumeroDocumento.getText().trim();

String alturaCalle = textFieldAlturaCalle.getText().trim();

String piso = textFieldPiso.getText().trim();

String departamento = textFieldDepartamento.getText().trim();

String telefono = textFieldTelefono.getText().trim();

String correoElectronico = textFieldCorreoElectronico.getText().trim();

String otros = textFieldOtros.getText().trim();

Localidad localidad = comboBoxLocalidad.getValue();

TipoDocumento tipoDoc = comboBoxTipoDocumento.getValue();

Barrio barrio = comboBoxBarrio.getValue();

Calle calle = comboBoxCalle.getValue();

//verifico que no estén vacíos

**if**(nombre.isEmpty()){

error.append("Inserte un nombre").append("\n");

}

**if**(apellido.isEmpty()){

error.append("Inserte un apellido").append("\n ");

}

**if**(numeroDocumento.isEmpty()){

error.append("Inserte un numero de documento").append("\n ");

}

**if**(alturaCalle.isEmpty()){

error.append("Inserte una altura").append("\n ");

}

**if**(telefono.isEmpty()){

error.append("Inserte un telefono").append("\n ");

}

**if**(correoElectronico.isEmpty()){

error.append("Inserte un correo electrónico").append("\n ");

}

**if**(calle == **null**){

error.append("Elija una calle").append("\n");

}

**if**(barrio == **null**){

error.append("Elija un barrio").append("\n");

}

**if**(tipoDoc == **null**){

error.append("Elija un tipo de documento").append("\n");

}

**if**(localidad == **null**){

error.append("Elija una localidad").append("\n");

}

**if**(!error.toString().isEmpty()){//si hay algún error lo muestro al usuario

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

//Si no hay errores se modifican las entidades con los datos introducidos

propietarioEnModificacion.getDireccion().setNumero(alturaCalle)

.setCalle(calle)

.setBarrio(barrio)

.setPiso(piso)

.setDepartamento(departamento)

.setLocalidad(localidad)

.setOtros(otros);

propietarioEnModificacion.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setEmail(correoElectronico);

**try**{ //relevo la operación a capa lógica

ResultadoModificarPropietario resultado = coordinador.modificarPropietario(propietarioEnModificacion);

**if**(resultado.hayErrores()){ // si hay algún error se muestra al usuario

StringBuilder stringErrores = **new** StringBuilder();

**for**(ErrorModificarPropietario err: resultado.getErrores()){

**switch**(err) {

**case** ***Formato\_Nombre\_Incorrecto***:

stringErrores.append("Formato de nombre incorrecto.\n");

**break**;

**case** ***Formato\_Apellido\_Incorrecto***:

stringErrores.append("Formato de apellido incorrecto.\n");

**break**;

**case** ***Formato\_Telefono\_Incorrecto***:

stringErrores.append("Formato de teléfono incorrecto.\n");

**break**;

**case** ***Formato\_Documento\_Incorrecto***:

stringErrores.append("Tipo y formato de documento incorrecto.\n");

**break**;

**case** ***Formato\_Email\_Incorrecto***:

stringErrores.append("Formato de email incorrecto.\n");

**break**;

**case** ***Formato\_Direccion\_Incorrecto***:

stringErrores.append("Formato de dirección incorrecto.\n");

**break**;

**case** ***Ya\_Existe\_Propietario***:

stringErrores.append("Otro cliente posee ese tipo y número de documento.\n");

**break**;

}

}

presentador.presentarError("Revise sus campos", stringErrores.toString(), stage);

}

**else**{ //si no hay errores se muestra notificación y se vuelve a la pantalla de listar propietarios

presentador.presentarToast("Se ha modificado el propietario con éxito", stage);

cambiarmeAScene(AdministrarPropietarioController.***URLVista***);

}

} **catch**(PersistenciaException e){ //excepción originada en la capa de persistencia

presentador.presentarExcepcion(e, stage);

}

}

}

Código del test de modificar en ModificarPropietarioController.java

**protected** Object[] parametersForTestModificarPropietario() {

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

Pais pais = **new** Pais("Argentina");

Provincia provincia = **new** Provincia("Santa fé", pais);

Localidad localidad = **new** Localidad("Capital", provincia);

Direccion dir = **new** Direccion("1865", "1", "B", **new** Calle("San martín", localidad), **new** Barrio("Centro", localidad), localidad, "");

**return** **new** Object[] {

//String nombre,apellido,tipoDocumento,nroDocumento,telefono,email,direccion,resultadoModificarPropietarioEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,excepcion

//prueba correcta

/\*0\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 0, **null** },

//prueba nombre incorrecto

/\*1\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoModificarNombreIncorrecto***, 1, 0, **null** },

//prueba apellido incorrecto

/\*2\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoModificarApellidoIncorrecto***, 1, 0, **null** },

//prueba documento incorrecto

/\*3\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoModificarDocumentoIncorrecto***, 1, 0, **null** },

//prueba teléfono incorrecto

/\*4\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoModificarTelefonoIncorrecto***, 1, 0, **null** },

//prueba email incorrecto

/\*5\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoModificarEmailIncorrecto***, 1, 0, **null** },

//prueba direccion incorrecta

/\*6\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoModificarDireccionIncorrecta***, 1, 0, **null** },

//prueba ya existe propietario

/\*7\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoModificarYaExiste***, 1, 0, **null** },

//prueba ya existe propietario

/\*8\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, **new** ResultadoModificarPropietario(ErrorModificarPropietario.***Formato\_Nombre\_Incorrecto***, ErrorModificarPropietario.***Formato\_Apellido\_Incorrecto***), 1, 0, **null** },

//prueba nombre vacio

/\*9\*/ **new** Object[] { "", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, **null**, 1, 0, **null** },

//prueba PersistenciaException

/\*10\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 1, **new** SaveUpdateException(**new** Throwable()) }

};

}

/\*\*

\* Test para probar al controlador de modificar propietario cuando el usuario presiona aceptar

\* **@param** nombre

\* nombre que se introduce por el usuario

\* **@param** apellido

\* apellido que se introduce por el usuario

\* **@param** tipoDocumento

\* tipo de documento que se introduce por el usuario

\* **@param** numeroDocumento

\* número de documento que se introduce por el usuario

\* **@param** telefono

\* teléfono que se introduce por el usuario

\* **@param** email

\* email que se introduce por el usuario

\* **@param** direccion

\* dirección que se introduce por el usuario

\* **@param** resultadoModificarPropietarioEsperado

\* resultado que retornará el mock de capa lógica

\* **@param** llamaAPresentadorVentanasPresentarError

\* 1 si llama al método presentar error del presentador de ventanas, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* 1 si llama al método presentar excepción del presentador de ventanas, 0 si no

\* **@param** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testModificarPropietario(String nombre,

String apellido,

TipoDocumento tipoDocumento,

String numeroDocumento,

String telefono,

String email,

Direccion direccion,

ResultadoModificarPropietario resultadoModificarPropietarioEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Exception excepcion) **throws** Throwable {

CoordinadorJavaFX coordinadorMock = Mockito.*mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaConfirmacion ventanaConfirmacionMock = *mock*(VentanaConfirmacion.**class**);

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

ModificarPropietarioController modificarPropietarioControllerMock = *mock*(ModificarPropietarioController.**class**);

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarConfirmacion(*any*(), *any*(), *any*())).thenReturn(ventanaConfirmacionMock);

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(modificarPropietarioControllerMock);

*doNothing*().when(modificarPropietarioControllerMock).setPropietarioEnModificacion(*any*());

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

*propietario* = **new** Propietario()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setEmail(email)

.setDireccion(direccion);

*when*(coordinadorMock.modificarPropietario(*propietario*)).thenReturn(resultadoModificarPropietarioEsperado);

**if**(excepcion != **null**){

*when*(coordinadorMock.modificarPropietario(*propietario*)).thenThrow(excepcion);

}

ArrayList<TipoDocumento> tipos = **new** ArrayList<>();

tipos.add(tipoDocumento);

Mockito.*when*(coordinadorMock.obtenerTiposDeDocumento()).thenReturn(tipos);

ModificarPropietarioController modificarPropietarioController = **new** ModificarPropietarioController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

setScenographyChanger(scenographyChangerMock);

**super**.inicializar(location, resources);

}

@Override

**public** **void** acceptAction() {

**this**.textFieldNombre.setText(nombre);

**this**.textFieldApellido.setText(apellido);

**this**.comboBoxTipoDocumento.getSelectionModel().select(tipoDocumento);

**this**.textFieldNumeroDocumento.setText(numeroDocumento);

**this**.textFieldTelefono.setText(telefono);

**this**.textFieldCorreoElectronico.setText(email);

**this**.comboBoxPais.getSelectionModel().select(direccion.getLocalidad().getProvincia().getPais());

**this**.comboBoxProvincia.getSelectionModel().select(direccion.getLocalidad().getProvincia());

**this**.comboBoxLocalidad.getSelectionModel().select(direccion.getLocalidad());

**this**.comboBoxBarrio.getSelectionModel().select(direccion.getBarrio());

**this**.comboBoxCalle.getSelectionModel().select(direccion.getCalle());

**this**.textFieldAlturaCalle.setText(direccion.getNumero());

**this**.textFieldDepartamento.setText(direccion.getDepartamento());

**this**.textFieldPiso.setText(direccion.getPiso());

**this**.textFieldOtros.setText(direccion.getOtros());

**super**.acceptAction();

}

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** setPropietarioEnModificacion(Propietario propietarioEnModificacion) {

**this**.propietarioEnModificacion = propietarioEnModificacion;

}

};

modificarPropietarioController.setPropietarioEnModificacion(*propietario*);

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(ModificarPropietarioController.***URLVista***, modificarPropietarioController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

modificarPropietarioController.acceptAction();

Mockito.*verify*(coordinadorMock).obtenerTiposDeDocumento();

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

}

};

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

Código de baja en AdministrarPropietarioController.java

/\*\*

\* Acción que se ejecuta al presionar el botón eliminar

\* Se muestra una ventana emergente para confirmar la operación

\*/

@FXML

**protected** ResultadoControlador handleEliminar() {

ArrayList<ErrorControlador> erroresControlador = **new** ArrayList<>();

**if**(tablaPropietarios.getSelectionModel().getSelectedItem() == **null**){

**return** **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***);

}

//solicito confirmación al usuario

VentanaConfirmacion ventana = presentador.presentarConfirmacion("Eliminar propietario", "Está a punto de eliminar al propietario.\n ¿Está seguro que desea hacerlo?", **this**.stage);

**if**(!ventana.acepta()){

**return** **new** ResultadoControlador(); //si no acepta

}

**try**{

ResultadoEliminarPropietario resultado = coordinador.eliminarPropietario(tablaPropietarios.getSelectionModel().getSelectedItem());

**if**(resultado.hayErrores()){ // si hay errores lo muestro al usuario

StringBuilder stringErrores = **new** StringBuilder();

**for**(ErrorEliminarPropietario err: resultado.getErrores()){

**switch**(err) {

}

}

presentador.presentarError("No se pudo eliminar el propietario", stringErrores.toString(), stage);

}

**else**{ //si no hay errores muestro notificación

presentador.presentarToast("Se ha eliminado el propietario con éxito", stage);

}

//actualizo la tabla

tablaPropietarios.getItems().clear();

tablaPropietarios.getItems().addAll(coordinador.obtenerPropietarios());

**return** **new** ResultadoControlador(erroresControlador.toArray(**new** ErrorControlador[0]));

} **catch**(PersistenciaException e){ //falla en la capa de persistencia

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

} **catch**(Exception e){ // alguna otra excepción inesperada

presentador.presentarExcepcionInesperada(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

}

Código del test de baja en AdministrarPropietarioControllerTest.java

**protected** Object[] parametersForTestEliminarPropietario() {

Propietario propietario = **new** Propietario();

Boolean acepta = **true**;

ResultadoControlador resultadoControladorCorrecto = **new** ResultadoControlador();

ResultadoControlador resultadoControladorErrorPersistencia = **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

ResultadoControlador resultadoControladorErrorDesconocido = **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

ResultadoEliminarPropietario resultadoLogicaCorrecto = **new** ResultadoEliminarPropietario();

Throwable excepcionPersistencia = **new** ObjNotFoundException("", **new** Exception());

Throwable excepcionInesperada = **new** Exception();

**return** **new** Object[] {

//propietario, acepta, resultadoControlador,resultadoLogica,excepcion

/\*0\*/ **new** Object[] { propietario, acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario acepta y el propietario se elimina correctamente

/\*1\*/ **new** Object[] { propietario, !acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario no acepta, pero de haber aceptado, se hubiese eliminado el propietario correctamente

/\*2\*/ **new** Object[] { propietario, acepta, resultadoControladorErrorPersistencia, **null**, excepcionPersistencia }, //test donde el controlador tira una excepción de persistencia

/\*3\*/ **new** Object[] { propietario, acepta, resultadoControladorErrorDesconocido, **null**, excepcionInesperada } //test donde el controlador tira unaexcepción inesperada

};

}

/\*\*

\* Test para probar eliminar propietario en el controlador administrar cuando el usuario presiona eliminar

\*

\* **@param** propietario

\* propietario a eliminar

\* **@param** acepta

\* si el usuario acepta la confirmación de eliminar

\* **@param** resultadoControlador

\* resultado que se espera que retorne el método a probar

\* **@param** resultadoLogica

\* resultado que devuelve la operación de capa lógica

\* **@param** excepcion

\*/

**public** **void** testEliminarPropietario(Propietario propietario, Boolean acepta, ResultadoControlador resultadoControlador, ResultadoEliminarPropietario resultadoLogica, Throwable excepcion) **throws** Exception {

CoordinadorJavaFX coordinadorMock = **new** CoordinadorJavaFX() {

@Override

**public** ResultadoEliminarPropietario eliminarPropietario(Propietario propietario) **throws** PersistenciaException {

**if**(resultadoLogica != **null**){

**return** resultadoLogica;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

@Override

**public** ArrayList<Propietario> obtenerPropietarios() **throws** PersistenciaException {

ArrayList<Propietario> propietarios = **new** ArrayList<>();

propietarios.add(propietario);

**return** propietarios;

}

};

PresentadorVentanas presentadorMock = **new** PresentadorVentanasMock(acepta);

AdministrarPropietarioController administrarPropietarioController = **new** AdministrarPropietarioController() {

@Override

**public** ResultadoControlador handleEliminar() {

tablaPropietarios.getSelectionModel().select(propietario);

**return** **super**.handleEliminar();

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

administrarPropietarioController.setCoordinador(coordinadorMock);

administrarPropietarioController.setPresentador(presentadorMock);

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarPropietarioController.***URLVista***, administrarPropietarioController);

administrarPropietarioController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

*assertEquals*(resultadoControlador, administrarPropietarioController.handleEliminar());

}

};

**try**{

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

### Taskcard 9 Lógica alta, modificar y baja propietario

Código del archivo GestorPropietario.java

/\*\*

\* Se encarga de validar los datos de un propietario a crear y, en caso de que no haya errores,

\* delegar el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** propietario

\* propietario a crear

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\* **@throws** GestionException

\* se lanza una excepción EntidadExistenteConEstadoBaja cuando se encuentra que ya existe un propietario con la misma identificación pero tiene estado BAJA

\*/

**public** ResultadoCrearPropietario crearPropietario(Propietario propietario) **throws** PersistenciaException, GestionException {

ArrayList<ErrorCrearPropietario> errores = **new** ArrayList<>();

validarDatosCrearPropietario(propietario, errores);

Propietario propietarioAuxiliar = persistidorPropietario.obtenerPropietario(**new** FiltroPropietario(propietario.getTipoDocumento().getTipo(), propietario.getNumeroDocumento()));

**if**(**null** != propietarioAuxiliar){

**if**(propietarioAuxiliar.getEstado().getEstado().equals(EstadoStr.***ALTA***)){

errores.add(ErrorCrearPropietario.***Ya\_Existe\_Propietario***);

}

**else**{

**throw** **new** EntidadExistenteConEstadoBajaException();

}

}

**if**(errores.isEmpty()){

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***ALTA***)){

propietario.setEstado(e);

**break**;

}

}

persistidorPropietario.guardarPropietario(propietario);

}

**return** **new** ResultadoCrearPropietario(errores.toArray(**new** ErrorCrearPropietario[0]));

}

/\*\*

\* Método auxiliar que encapsula la validación de los formatos de los atributos del propietario a crear

\*

\* **@param** propietario

\* propietario a validar

\* **@param** errores

\* lista de errores encontrados

\*/

**private** **void** validarDatosCrearPropietario(Propietario propietario, ArrayList<ErrorCrearPropietario> errores) {

**if**(!validador.validarNombre(propietario.getNombre())){

errores.add(ErrorCrearPropietario.***Formato\_Nombre\_Incorrecto***);

}

**if**(!validador.validarApellido(propietario.getApellido())){

errores.add(ErrorCrearPropietario.***Formato\_Apellido\_Incorrecto***);

}

**if**(!validador.validarDocumento(propietario.getTipoDocumento(), propietario.getNumeroDocumento())){

errores.add(ErrorCrearPropietario.***Formato\_Documento\_Incorrecto***);

}

**if**(!validador.validarTelefono(propietario.getTelefono())){

errores.add(ErrorCrearPropietario.***Formato\_Telefono\_Incorrecto***);

}

**if**(propietario.getEmail() != **null** && !validador.validarEmail(propietario.getEmail())){

errores.add(ErrorCrearPropietario.***Formato\_Email\_Incorrecto***);

}

**if**(!validador.validarDireccion(propietario.getDireccion())){

errores.add(ErrorCrearPropietario.***Formato\_Direccion\_Incorrecto***);

}

}

/\*\*

\* Se encarga de validar los datos de un propietario a modificar y, en caso de que no haya errores,

\* delegar el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** propietario

\* propietario a modificar

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ResultadoModificarPropietario modificarPropietario(Propietario propietario) **throws** PersistenciaException {

ArrayList<ErrorModificarPropietario> errores = **new** ArrayList<>();

validarDatosModificarPropietario(propietario, errores);

Propietario propietarioAuxiliar;

propietarioAuxiliar = persistidorPropietario.obtenerPropietario(**new** FiltroPropietario(propietario.getTipoDocumento().getTipo(), propietario.getNumeroDocumento()));

**if**(**null** != propietarioAuxiliar && !propietario.equals(propietarioAuxiliar)){

errores.add(ErrorModificarPropietario.***Ya\_Existe\_Propietario***);

}

**if**(errores.isEmpty()){

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***ALTA***)){

propietario.setEstado(e);

**break**;

}

}

persistidorPropietario.modificarPropietario(propietario);

}

**return** **new** ResultadoModificarPropietario(errores.toArray(**new** ErrorModificarPropietario[0]));

}

/\*\*

\* Método auxiliar que encapsula la validación de los formatos de los atributos del propietario a modificar

\*

\* **@param** propietario

\* propietario a validar

\* **@param** errores

\* lista de errores encontrados

\*/

**private** **void** validarDatosModificarPropietario(Propietario propietario, ArrayList<ErrorModificarPropietario> errores) {

**if**(!validador.validarNombre(propietario.getNombre())){

errores.add(ErrorModificarPropietario.***Formato\_Nombre\_Incorrecto***);

}

**if**(!validador.validarApellido(propietario.getApellido())){

errores.add(ErrorModificarPropietario.***Formato\_Apellido\_Incorrecto***);

}

**if**(!validador.validarDocumento(propietario.getTipoDocumento(), propietario.getNumeroDocumento())){

errores.add(ErrorModificarPropietario.***Formato\_Documento\_Incorrecto***);

}

**if**(!validador.validarTelefono(propietario.getTelefono())){

errores.add(ErrorModificarPropietario.***Formato\_Telefono\_Incorrecto***);

}

**if**(!validador.validarEmail(propietario.getEmail())){

errores.add(ErrorModificarPropietario.***Formato\_Email\_Incorrecto***);

}

**if**(!validador.validarDireccion(propietario.getDireccion())){

errores.add(ErrorModificarPropietario.***Formato\_Direccion\_Incorrecto***);

}

}

/\*\*

\* Se encarga de validar que exista el propietario a eliminar, se setea el estado en BAJA y,

\* en caso de que no haya errores, delegar el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** propietario

\* propietario a eliminar

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ResultadoEliminarPropietario eliminarPropietario(Propietario propietario) **throws** PersistenciaException {

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***BAJA***)){

propietario.setEstado(e);

**break**;

}

}

persistidorPropietario.modificarPropietario(propietario);

**return** **new** ResultadoEliminarPropietario();

}

/\*\*

\* Obtiene el listado de propietarios solicitándolo a la capa de acceso a datos

\*

\* **@return** el listado de propietarios solicitado

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ArrayList<Propietario> obtenerPropietarios() **throws** PersistenciaException {

**return** persistidorPropietario.listarPropietarios();

}

/\*\*

\* Obtiene un propietario en base a su tipo y número de documento solicitándolo a la capa de acceso a datos

\*

\* **@param** filtro

\* filtro que contiene el tipo y número de documento del propietario buscado

\*

\* **@return** el propietario solicitado

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** Propietario obtenerPropietario(FiltroPropietario filtro) **throws** PersistenciaException {

Propietario propietarioAuxiliar;

propietarioAuxiliar = persistidorPropietario.obtenerPropietario(**new** FiltroPropietario(filtro.getTipoDocumento(), filtro.getDocumento()));

**return** propietarioAuxiliar;

}

Prueba de unidad del archivo GestorPropietario.java

//Casos de prueba

//resValNombre, resValApellido, resValDocumento, resValTelefono, resValEmail, resValDireccion, resObtenerPropietario, guardar, resultadoCrearPropietarioEsperado, excepcion, excepcionEsperada

/\* 0 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **true**, **null**, 1, ***resultadoCrearCorrecto***, **null**, **null** }, //Test correcto

/\* 1 \*/ **new** Object[] { **false**, **true**, **true**, **true**, **true**, **true**, **null**, 0, ***resultadoCrearNombreIncorrecto***, **null**, **null** }, //Nombre Incorrecto

/\* 2 \*/ **new** Object[] { **true**, **false**, **true**, **true**, **true**, **true**, **null**, 0, ***resultadoCrearApellidoIncorrecto***, **null**, **null** }, //Apellido Incorrecto

/\* 3 \*/ **new** Object[] { **true**, **true**, **false**, **true**, **true**, **true**, **null**, 0, ***resultadoCrearDocumentoIncorrecto***, **null**, **null** }, //Documento Incorrecto

/\* 4 \*/ **new** Object[] { **true**, **true**, **true**, **false**, **true**, **true**, **null**, 0, ***resultadoCrearTelefonoIncorrecto***, **null**, **null** }, //Teléfono Incorrecto

/\* 5 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **false**, **true**, **null**, 0, ***resultadoCrearEmailIncorrecto***, **null**, **null** }, //Email Incorrecto

/\* 6 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **false**, **null**, 0, ***resultadoCrearDireccionIncorrecto***, **null**, **null** }, //Dirección incorrecta

/\* 7 \*/ **new** Object[] { **false**, **false**, **true**, **true**, **true**, **true**, **null**, 0, **new** ResultadoCrearPropietario(ErrorCrearPropietario.***Formato\_Nombre\_Incorrecto***, ErrorCrearPropietario.***Formato\_Apellido\_Incorrecto***), **null**, **null** }, //Nombre y apellido incorrectos

/\* 8 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **true**, *propietario*, 0, ***resultadoCrearYaExiste***, **null**, **null** }, //Ya existe un propietario con el mismo documento

/\* 9 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **true**, **null**, 1, **null**, **new** SaveUpdateException(**new** Exception()), **new** SaveUpdateException(**new** Exception()) }, //La base de datos tira una excepción

/\* 10 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **true**, propietario2, 0, **null**, **null**, **new** EntidadExistenteConEstadoBajaException() } //El propietario ya existe pero con estado BAJA

/\*\*

\* Test para el método crear propietario

\*

\* **@param** resValNombre

\* resultado devuelto por el mock validador de formato al validar nombre

\* **@param** resValApellido

\* resultado devuelto por el mock validador de formato al validar apellido

\* **@param** resValDocumento

\* resultado devuelto por el mock validador de formato al validar documento

\* **@param** resValTelefono

\* resultado devuelto por el mock validador de formato al validar teléfono

\* **@param** resValEmail

\* resultado devuelto por el mock validador de formato al validar email

\* **@param** resValDireccion

\* resultado devuelto por el mock validador de formato al validar dirección

\* **@param** resObtenerPropietario

\* resultado devuelto por la base de datos al obtener un propietario

\* **@param** guardar

\* indica si se debe mandar a guardar el propietario

\* **@param** resultadoCrearPropietarioEsperado

\* resultado esperado del test

\* **@param** excepcion

\* excepción devuelta por la base de datos

\* **@param** excepcionEsperada

\* excepción esperada que debería lanzar el gestor

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testCrearPropietario(Boolean resValNombre, Boolean resValApellido, Boolean resValDocumento, Boolean resValTelefono, Boolean resValEmail, Boolean resValDireccion, Propietario resObtenerPropietario, Integer guardar, ResultadoCrearPropietario resultadoCrearPropietarioEsperado, Throwable excepcion, Throwable excepcionEsperada) **throws** Exception {

//Inicialización de los mocks

PropietarioService propietarioServiceMock = *mock*(PropietarioService.**class**);

ValidadorFormato validadorFormatoMock = *mock*(ValidadorFormato.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

//Clase anónima necesaria para inyectar dependencias

GestorPropietario gestorPropietario = **new** GestorPropietario() {

{

**this**.persistidorPropietario = propietarioServiceMock;

**this**.validador = validadorFormatoMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***BAJA***));

estados.add(**new** Estado(EstadoStr.***ALTA***));

//Setear valores esperados a los mocks

*when*(validadorFormatoMock.validarNombre(*propietario*.getNombre())).thenReturn(resValNombre);

*when*(validadorFormatoMock.validarApellido(*propietario*.getApellido())).thenReturn(resValApellido);

*when*(validadorFormatoMock.validarDocumento(*propietario*.getTipoDocumento(), *propietario*.getNumeroDocumento())).thenReturn(resValDocumento);

*when*(validadorFormatoMock.validarTelefono(*propietario*.getTelefono())).thenReturn(resValTelefono);

*when*(validadorFormatoMock.validarEmail(*propietario*.getEmail())).thenReturn(resValEmail);

*when*(validadorFormatoMock.validarDireccion(*propietario*.getDireccion())).thenReturn(resValDireccion);

*when*(propietarioServiceMock.obtenerPropietario(*any*())).thenReturn(resObtenerPropietario);

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

//Setear la excepcion devuelta por la base de datos, si corresponde

**if**(excepcion != **null**){

*doThrow*(excepcion).when(propietarioServiceMock).guardarPropietario(*propietario*);

}

**else**{

*doNothing*().when(propietarioServiceMock).guardarPropietario(*propietario*);

}

//Llamar al método a testear y comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

**if**(excepcionEsperada == **null**){

*assertEquals*(resultadoCrearPropietarioEsperado.getErrores(), gestorPropietario.crearPropietario(*propietario*).getErrores());

}

**else**{

**try**{

gestorPropietario.crearPropietario(*propietario*);

Assert.*fail*("Debería haber fallado!");

} **catch**(Exception e){

*assertEquals*(excepcionEsperada.getClass(), e.getClass());

}

}

*verify*(validadorFormatoMock).validarNombre(*propietario*.getNombre());

*verify*(validadorFormatoMock).validarApellido(*propietario*.getApellido());

*verify*(validadorFormatoMock).validarDocumento(*propietario*.getTipoDocumento(), *propietario*.getNumeroDocumento());

*verify*(validadorFormatoMock).validarTelefono(*propietario*.getTelefono());

*verify*(validadorFormatoMock).validarEmail(*propietario*.getEmail());

*verify*(validadorFormatoMock).validarDireccion(*propietario*.getDireccion());

*verify*(gestorDatosMock, *times*(guardar)).obtenerEstados();

*verify*(propietarioServiceMock, *times*(guardar)).guardarPropietario(*propietario*);

}

//Casos de prueba

//resValNombre, resValApellido, resValDocumento, resValTelefono, resValEmail, resValDireccion, resObtenerPropietario, modificar, resultadoModificarPropietarioEsperado, excepcion, excepcionEsperada

/\* 0 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **true**, **null**, 1, ***resultadoModificarCorrecto***, **null**, **null** }, //Test correcto (se modifica el documento)

/\* 1 \*/ **new** Object[] { **false**, **true**, **true**, **true**, **true**, **true**, **null**, 0, ***resultadoModificarNombreIncorrecto***, **null**, **null** }, //Nombre Incorrecto

/\* 2 \*/ **new** Object[] { **true**, **false**, **true**, **true**, **true**, **true**, **null**, 0, ***resultadoModificarApellidoIncorrecto***, **null**, **null** }, //Apellido Incorrecto

/\* 3 \*/ **new** Object[] { **true**, **true**, **false**, **true**, **true**, **true**, **null**, 0, ***resultadoModificarDocumentoIncorrecto***, **null**, **null** }, //Documento Incorrecto

/\* 4 \*/ **new** Object[] { **true**, **true**, **true**, **false**, **true**, **true**, **null**, 0, ***resultadoModificarTelefonoIncorrecto***, **null**, **null** }, //Teléfono Incorrecto

/\* 5 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **false**, **true**, **null**, 0, ***resultadoModificarEmailIncorrecto***, **null**, **null** }, //Email Incorrecto

/\* 6 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **false**, **null**, 0, ***resultadoModificarDireccionIncorrecto***, **null**, **null** }, //Dirección incorrecta

/\* 7 \*/ **new** Object[] { **false**, **false**, **true**, **true**, **true**, **true**, **null**, 0, **new** ResultadoModificarPropietario(ErrorModificarPropietario.***Formato\_Nombre\_Incorrecto***, ErrorModificarPropietario.***Formato\_Apellido\_Incorrecto***), **null**, **null** }, //Nombre y apellido incorrectos

/\* 8 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **true**, propietario2, 0, ***resultadoModificarYaSePoseeMismoDocumento***, **null**, **null** }, //Ya existe un propietario con el mismo documento

/\* 9 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **true**, *propietario*, 1, ***resultadoModificarCorrecto***, **null**, **null** }, //Test correcto, no se cambió el documento

/\* 10 \*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **true**, **null**, 1, **null**, **new** SaveUpdateException(**new** Exception()), **new** SaveUpdateException(**new** Exception()) } //La base de datos tira una excepción

/\*\*

\* Test para el método modificar propietario

\*

\* **@param** resValNombre

\* resultado devuelto por el mock validador de formato al validar nombre

\* **@param** resValApellido

\* resultado devuelto por el mock validador de formato al validar apellido

\* **@param** resValDocumento

\* resultado devuelto por el mock validador de formato al validar documento

\* **@param** resValTelefono

\* resultado devuelto por el mock validador de formato al validar teléfono

\* **@param** resValEmail

\* resultado devuelto por el mock validador de formato al validar email

\* **@param** resValDireccion

\* resultado devuelto por el mock validador de formato al validar dirección

\* **@param** resObtenerPropietario

\* resultado devuelto por la base de datos al obtener un propietario

\* **@param** modificar

\* indica si se debe mandar a guardar el propietario modificado

\* **@param** resultadoModificarPropietarioEsperado

\* resultado esperado del test

\* **@param** excepcion

\* excepción devuelta por la base de datos

\* **@param** excepcionEsperada

\* excepción esperada que debería lanzar el gestor

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testModificarPropietario(Boolean resValNombre, Boolean resValApellido, Boolean resValDocumento, Boolean resValTelefono, Boolean resValEmail, Boolean resValDireccion, Propietario resObtenerPropietario, Integer modificar, ResultadoModificarPropietario resultadoModificarPropietarioEsperado, Throwable excepcion, Throwable excepcionEsperada) **throws** Exception {

//Inicialización de los mocks

PropietarioService propietarioServiceMock = *mock*(PropietarioService.**class**);

ValidadorFormato validadorFormatoMock = *mock*(ValidadorFormato.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

//Clase anónima necesaria para inyectar dependencias

GestorPropietario gestorPropietario = **new** GestorPropietario() {

{

**this**.persistidorPropietario = propietarioServiceMock;

**this**.validador = validadorFormatoMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***BAJA***));

estados.add(**new** Estado(EstadoStr.***ALTA***));

//Setear valores esperados a los mocks

*when*(validadorFormatoMock.validarNombre(*propietario*.getNombre())).thenReturn(resValNombre);

*when*(validadorFormatoMock.validarApellido(*propietario*.getApellido())).thenReturn(resValApellido);

*when*(validadorFormatoMock.validarDocumento(*propietario*.getTipoDocumento(), *propietario*.getNumeroDocumento())).thenReturn(resValDocumento);

*when*(validadorFormatoMock.validarTelefono(*propietario*.getTelefono())).thenReturn(resValTelefono);

*when*(validadorFormatoMock.validarEmail(*propietario*.getEmail())).thenReturn(resValEmail);

*when*(validadorFormatoMock.validarDireccion(*propietario*.getDireccion())).thenReturn(resValDireccion);

*when*(propietarioServiceMock.obtenerPropietario(*any*())).thenReturn(resObtenerPropietario);

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

//Setear la excepcion devuelta por la base de datos, si corresponde

**if**(excepcion != **null**){

*doThrow*(excepcion).when(propietarioServiceMock).modificarPropietario(*propietario*);

}

**else**{

*doNothing*().when(propietarioServiceMock).modificarPropietario(*propietario*);

}

//Llamar al método a testear y comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

**if**(excepcionEsperada == **null**){

*assertEquals*(resultadoModificarPropietarioEsperado.getErrores(), gestorPropietario.modificarPropietario(*propietario*).getErrores());

}

**else**{

**try**{

gestorPropietario.modificarPropietario(*propietario*);

Assert.*fail*("Debería haber fallado!");

} **catch**(Exception e){

*assertEquals*(excepcionEsperada.getClass(), e.getClass());

}

}

*verify*(validadorFormatoMock).validarNombre(*propietario*.getNombre());

*verify*(validadorFormatoMock).validarApellido(*propietario*.getApellido());

*verify*(validadorFormatoMock).validarDocumento(*propietario*.getTipoDocumento(), *propietario*.getNumeroDocumento());

*verify*(validadorFormatoMock).validarTelefono(*propietario*.getTelefono());

*verify*(validadorFormatoMock).validarEmail(*propietario*.getEmail());

*verify*(validadorFormatoMock).validarDireccion(*propietario*.getDireccion());

*verify*(propietarioServiceMock).obtenerPropietario(*any*());

*verify*(gestorDatosMock, *times*(modificar)).obtenerEstados();

*verify*(propietarioServiceMock, *times*(modificar)).modificarPropietario(*propietario*);

}

//Casos de prueba

//resObtenerPropietario, eliminar, resultadoEliminarPropietarioEsperado, excepcion, excepcionEsperada

/\* 0 \*/ **new** Object[] { *propietario*, 1, ***resultadoEliminarCorrecto***, **null**, **null** }, //Test correcto

/\* 1 \*/ **new** Object[] { *propietario*, 1, **null**, **new** SaveUpdateException(**new** Exception()), **new** SaveUpdateException(**new** Exception()) } //La base de datos tira una excepción

/\* @param resObtenerPropietario

\* resultado devuelto por la base de datos al obtener un propietario

\* @param eliminar

\* indica si se debe eliminar el propietario

\* @param resultadoEliminarPropietarioEsperado

\* resultado esperado del test

\* @param excepcion

\* excepción devuelta por la base de datos

\* @param excepcionEsperada

\* excepción esperada que debería lanzar el gestor

\* @throws Exception

\*/

@Test

@Parameters

**public** **void** testEliminarPropietario(Propietario resObtenerPropietario, Integer eliminar, ResultadoEliminarPropietario resultadoEliminarPropietarioEsperado, Throwable excepcion, Throwable excepcionEsperada) **throws** Exception {

//Inicialización de los mocks

PropietarioService propietarioServiceMock = *mock*(PropietarioService.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

//Clase anónima necesaria para inyectar dependencias

GestorPropietario gestorPropietario = **new** GestorPropietario() {

{

**this**.persistidorPropietario = propietarioServiceMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***ALTA***));

estados.add(**new** Estado(EstadoStr.***BAJA***));

//Setear valores esperados a los mocks

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

//Setear la excepcion devuelta por la base de datos, si corresponde

**if**(excepcion != **null**){

*doThrow*(excepcion).when(propietarioServiceMock).modificarPropietario(*propietario*);

}

**else**{

*doNothing*().when(propietarioServiceMock).modificarPropietario(*propietario*);

}

//Llamar al método a testear y comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

**if**(excepcionEsperada == **null**){

*assertEquals*(resultadoEliminarPropietarioEsperado.getErrores(), gestorPropietario.eliminarPropietario(*propietario*).getErrores());

}

**else**{

**try**{

gestorPropietario.eliminarPropietario(*propietario*);

Assert.*fail*("Debería haber fallado!");

} **catch**(Exception e){

*assertEquals*(excepcionEsperada.getClass(), e.getClass());

}

}

**if**(eliminar != 0){

*assertEquals*(EstadoStr.***BAJA***, *propietario*.getEstado().getEstado());

}

*verify*(gestorDatosMock, *times*(eliminar)).obtenerEstados();

*verify*(propietarioServiceMock, *times*(eliminar)).modificarPropietario(*propietario*);

}

### Taskcard 10 Persistidor Propietario

Código del archivo PropietarioServiceImpl.java

/\*

\* Método para guardar en la base de datos un propietario

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** **void** guardarPropietario(Propietario propietario) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.save(propietario);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

/\*

\* Método para modificar en la base de datos un propietario

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** **void** modificarPropietario(Propietario propietario) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.update(propietario);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

/\*

\* Método para obtener de la base de datos un propietario

\*/

@Override

@Transactional(readOnly = **true**, rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** Propietario obtenerPropietario(FiltroPropietario filtro) **throws** PersistenciaException {

Propietario propietario = **null**;

Session session = getSessionFactory().getCurrentSession();

**try**{

//named query ubicada en entidad propietario

propietario = (Propietario) session.getNamedQuery("obtenerPropietario").setParameter("tipoDocumento", filtro.getTipoDocumento()).setParameter("documento", filtro.getDocumento()).uniqueResult();

} **catch**(NoResultException e){

**return** **null**;

} **catch**(NonUniqueResultException e){

**return** **null**;

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** propietario;

}

/\*

\* Método para obtener de la base de datos todos los propietarios con estado ALTA

\*/

@Override

@Transactional(readOnly = **true**, rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** ArrayList<Propietario> listarPropietarios() **throws** PersistenciaException {

ArrayList<Propietario> propietarios = **new** ArrayList<>();

Session session = getSessionFactory().getCurrentSession();

**try**{

//named query ubicada en entidad propietario

**for**(Object o: session.getNamedQuery("obtenerPropietarios").list()){

**if**(o **instanceof** Propietario){

propietarios.add((Propietario) o);

}

}

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** propietarios;

}

### Taskcard 11 Entidad inmueble

Código del archivo Inmueble.java

@Entity

@NamedQuery(name = "obtenerInmuebles", query = "SELECT i FROM Inmueble i WHERE i.estado.estado = 'ALTA'")

@Table(name = "inmueble")

/\*\*

\* Entidad que modela un inmueble

\* Pertenece a la taskcard 11 de la iteración 1 y a la historia 3

\*/

**public** **class** Inmueble {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** Integer id; //ID

@Column(name = "observaciones", length = 300)

**private** String observaciones;

@Temporal(TemporalType.***DATE***)

@Column(name = "fecha\_carga", nullable = **false**)

**private** Date fechaCarga;

@Column(name = "precio", nullable = **false**)

**private** Double precio;

@Column(name = "frente")

**private** Double frente; // en metros

@Column(name = "fondo")

**private** Double fondo; // en metros

@Column(name = "superficie")

**private** Double superficie; // en metros cuadrados

//Relaciones

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idtipo", referencedColumnName = "id", foreignKey = @ForeignKey(name = "inmueble\_idtipo\_fk"), nullable = **false**)

**private** TipoInmueble tipo;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idorientacion", referencedColumnName = "id", foreignKey = @ForeignKey(name = "inmueble\_idorientacion\_fk"))

**private** Orientacion orientacion;

@OneToOne(fetch = FetchType.***EAGER***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "iddireccion", referencedColumnName = "id", foreignKey = @ForeignKey(name = "inmueble\_iddireccion\_fk"), nullable = **false**, unique = **true**)

**private** Direccion direccion;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idpropietario", referencedColumnName = "id", foreignKey = @ForeignKey(name = "inmueble\_idpropietario\_fk"), nullable = **false**)

**private** Propietario propietario;

@ManyToMany(fetch = FetchType.***EAGER***, cascade = CascadeType.***ALL***)

@JoinTable(name = "inmueble\_imagen", joinColumns = @JoinColumn(name = "idinmueble"), foreignKey = @ForeignKey(name = "inmueble\_imagen\_idinmueble\_fk"), inverseJoinColumns = @JoinColumn(name = "idimagen"), inverseForeignKey = @ForeignKey(name = "inmueble\_imagen\_idimagen\_fk"))

**private** Set<Imagen> fotos;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idestado", referencedColumnName = "id", foreignKey = @ForeignKey(name = "inmueble\_idestado\_fk"), nullable = **false**)

**private** Estado estado;

@OneToOne(mappedBy = "inmueble", cascade = CascadeType.***ALL***, orphanRemoval = **true**, optional = **false**)

**private** DatosEdificio datosEdificio;

//getters, setters, constructores y otros métodos

(…)

Código del archivo DatosEdificio.java

@Entity

@Table(name = "datos\_edificio")

/\*

\* Entidad que modela los datos que tiene un inmueble.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** DatosEdificio **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** Integer id; //ID

@OneToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idinmueble", referencedColumnName = "id", foreignKey = @ForeignKey(name = "datos\_edificio\_idinmueble\_fk"), nullable = **false**)

**private** Inmueble inmueble;

@Column(name = "superficie")

**private** Double superficie; // en metros cuadrados

@Column(name = "antiguedad")

**private** Integer antiguedad; // en años

@Column(name = "dormitorios")

**private** Integer dormitorios;

@Column(name = "baños")

**private** Integer baños;

@Column(name = "garaje")

**private** Boolean garaje;

@Column(name = "patio")

**private** Boolean patio;

@Column(name = "piscina")

**private** Boolean piscina;

@Column(name = "telefono")

**private** Boolean telefono;

@Column(name = "propiedad\_horizontal")

**private** Boolean propiedadHorizontal;

@Column(name = "agua\_corriente")

**private** Boolean aguaCorriente;

@Column(name = "cloacas")

**private** Boolean cloacas;

@Column(name = "gas\_natural")

**private** Boolean gasNatural;

@Column(name = "agua\_caliente")

**private** Boolean aguaCaliente;

@Column(name = "lavadero")

**private** Boolean lavadero;

@Column(name = "pavimento")

**private** Boolean pavimento;

//getters, setters, constructores y otros métodos

(…)

Código del archivo HistorialInmueble.java

@Entity

@Table(name = "historial\_inmueble")

/\*\*

\* Entidad que modela un estado anterior de un inmueble

\* Pertenece a la taskcard 11 de la iteración 1 y a la historia 3

\*/

**public** **class** HistorialInmueble {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** Long id; //ID

@Temporal(TemporalType.***TIMESTAMP***)

@Column(name = "fecha\_hora\_cambio", nullable = **false**)

**private** Date fechaYHoraCambio;

@Column(name = "observaciones", length = 300)

**private** String observaciones;

@Temporal(TemporalType.***DATE***)

@Column(name = "fecha\_carga", nullable = **false**)

**private** Date fechaCarga;

@Column(name = "precio", nullable = **false**)

**private** Double precio;

@Column(name = "frente")

**private** Double frente; // en metros

@Column(name = "fondo")

**private** Double fondo; // en metros

@Column(name = "superficie")

**private** Double superficie; // en metros cuadrados

//Relaciones

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idtipo", referencedColumnName = "id", foreignKey = @ForeignKey(name = "historial\_inmueble\_idtipo\_fk"), nullable = **false**)

**private** TipoInmueble tipo;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idorientacion", referencedColumnName = "id", foreignKey = @ForeignKey(name = "historial\_inmueble\_idorientacion\_fk"))

**private** Orientacion orientacion;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "iddireccion", referencedColumnName = "id", foreignKey = @ForeignKey(name = "historial\_inmueble\_iddireccion\_fk"), nullable = **false**)

**private** Direccion direccion;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idpropietario", referencedColumnName = "id", foreignKey = @ForeignKey(name = "historial\_inmueble\_idpropietario\_fk"), nullable = **false**)

**private** Propietario propietario;

@ManyToMany(fetch = FetchType.***EAGER***, cascade = CascadeType.***ALL***)

@JoinTable(name = "historial\_inmueble\_imagen", joinColumns = @JoinColumn(name = "idhistorial\_inmueble"), foreignKey = @ForeignKey(name = "historial\_inmueble\_imagen\_idinmueble\_fk"), inverseJoinColumns = @JoinColumn(name = "idimagen"), inverseForeignKey = @ForeignKey(name = "historial\_inmueble\_imagen\_idimagen\_fk"))

**private** Set<Imagen> fotos;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idestado", referencedColumnName = "id", foreignKey = @ForeignKey(name = "historial\_inmueble\_idestado\_fk"), nullable = **false**)

**private** Estado estado;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idinmueble", referencedColumnName = "id", foreignKey = @ForeignKey(name = "historial\_inmueble\_idinmueble\_fk"), nullable = **false**)

**private** Inmueble inmueble;

@OneToOne(mappedBy = "historialInmueble", cascade = CascadeType.***ALL***, orphanRemoval = **true**, optional = **false**)

**private** HistorialDatosEdificio historialDatosEdificio;

//getters, setters, constructores y otros métodos

(…)

### Taskcard 12 Clases de datos

Código del archivo Direccion.java

/\*

\* Entidad que modela una direccion.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** Direccion {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "numero", length = 30, nullable = **false**)

**private** String numero;

@Column(name = "piso", length = 30)

**private** String piso;

@Column(name = "departamento", length = 30)

**private** String departamento;

@Column(name = "otros", length = 100)

**private** String otros;

//Relaciones

@ManyToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "idcalle", referencedColumnName = "id", foreignKey = @ForeignKey(name = "direccion\_idcalle\_fk"))

**private** Calle calle;

@ManyToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "idbarrio", referencedColumnName = "id", foreignKey = @ForeignKey(name = "direccion\_idbarrio\_fk"))

**private** Barrio barrio;

@ManyToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "idlocalidad", referencedColumnName = "id", foreignKey = @ForeignKey(name = "direccion\_idlocalidad\_fk"))

**private** Localidad localidad;

Código del archivo Barrio.java

/\*\*

\* Entidad que modela un barrio

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** Barrio {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "nombre", length = 50, nullable = **false**)

**private** String nombre;

//Relaciones

@ManyToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "idlocalidad", referencedColumnName = "id", foreignKey = @ForeignKey(name = "barrio\_idlocalidad\_fk"), nullable = **false**)

**private** Localidad localidad;

Código del archivo Calle.java

/\*

\* Entidad que modela una calle.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** Calle {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "nombre", length = 50, nullable = **false**)

**private** String nombre;

//Relaciones

@ManyToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "idlocalidad", referencedColumnName = "id", foreignKey = @ForeignKey(name = "calle\_idlocalidad\_fk"), nullable = **false**)

**private** Localidad localidad;

Código del archivo País.java

/\*

\* Entidad que modela un país.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** Pais {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "nombre", length = 50, nullable = **false**)

**private** String nombre;

Código del archivo Provincia.java

/\*

\* Entidad que modela una provincia.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** Provincia {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "nombre", length = 50, nullable = **false**)

**private** String nombre;

//Relaciones

@ManyToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "idpais", referencedColumnName = "id", foreignKey = @ForeignKey(name = "provincia\_idpais\_fk"), nullable = **false**)

**private** Pais pais;

Código del archivo Localidad.java

/\*

\* Entidad que modela una localidad.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** Localidad {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "nombre", length = 50, nullable = **false**)

**private** String nombre;

//Relaciones

@ManyToOne(fetch = FetchType.***LAZY***, cascade = CascadeType.***ALL***)

@JoinColumn(name = "idprovincia", referencedColumnName = "id", foreignKey = @ForeignKey(name = "localidad\_idprovincia\_fk"), nullable = **false**)

**private** Provincia provincia;

Código del archivo TipoDocumento.java

/\*

\* Entidad que modela el tipo de un documento.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** TipoDocumento {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "tipo\_documento", nullable = **false**)

@Enumerated(EnumType.***STRING***)

**private** TipoDocumentoStr tipo;

Código del archivo TipoInmueble.java

/\*

\* Entidad que modela el tipo de un inmueble.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** TipoInmueble {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Enumerated(EnumType.***STRING***)

@Column(name = "tipo", length = 30)

**private** TipoInmuebleStr tipo;

Código del archivo Orientacion.java

/\*

\* Entidad que modela la orientacion física que tiene un inmueble.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** Orientacion {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "orientacion\_enum", length = 10)

@Enumerated(EnumType.***STRING***)

**private** OrientacionStr orientacion;

### Taskcard 13 Vista alta, modificar y baja inmueble

Código del archivo AdministrarInmuebleController.java

/\*\*

\* Método que se llama al hacer click en el botón agregar.

\*/

@FXML

**public** **void** agregar() {

//Cambia a la pantalla de agregar inmueble

cambiarmeAScene(NMVInmuebleController.***URLVista***, ***URLVista***);

}

/\*\*

\* Método que se llama al hacer click en el botón modificar.

\*/

@FXML

**public** **void** modificar() {

Inmueble inmueble = tablaInmuebles.getSelectionModel().getSelectedItem();

**if**(inmueble == **null** || inmueble.getEstadoInmueble().getEstado().equals(EstadoInmuebleStr.***VENDIDO***)){

return;

}

//Cambia a la pantalla de modificar inmueble

NMVInmuebleController nuevaPantalla = (NMVInmuebleController) cambiarmeAScene(NMVInmuebleController.***URLVista***, ***URLVista***);

//Le seteamos el inmueble que queremos modificar

nuevaPantalla.formatearModificarInmueble(inmueble);

}

/\*\*

\* Método que permite eliminar un inmueble

\* Pertenece a la taskcard 13 de la iteración 1 y a la historia 3

\*

\* **@return** ResultadoControlador que resume lo que hizo el controlador

\*/

@FXML

**public** ResultadoControlador eliminarInmueble() {

//Inicialización de variables

ArrayList<ErrorControlador> erroresControlador = **new** ArrayList<>();

ResultadoEliminarInmueble resultado;

StringBuffer erroresBfr = **new** StringBuffer();

//Toma de datos de la vista

Inmueble inmueble = tablaInmuebles.getSelectionModel().getSelectedItem();

**if**(inmueble == **null**){

**return** **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***);

}

//Se pregunta si quiere eliminar el inmueble

VentanaConfirmacion ventana = presentador.presentarConfirmacion("Eliminar inmueble", "Está a punto de eliminar a el inmueble.\n ¿Está seguro que desea hacerlo?", **this**.stage);

**if**(!ventana.acepta()){

**return** **new** ResultadoControlador();

}

**try**{

//Se llama a la lógica para eliminar el inmueble y se recibe el resultado de las validaciones y datos extras de ser necesarios

resultado = coordinador.eliminarInmueble(inmueble);

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

//Procesamiento de errores de la lógica

**if**(resultado.hayErrores()){

**for**(ErrorEliminarInmueble err: resultado.getErrores()){

**switch**(err) {

}

}

//Se muestran los errores

presentador.presentarError("No se pudo crear el inmueble", erroresBfr.toString(), stage);

}

else{

//Se muestra una notificación de que se eliminó correctamente el inmueble

presentador.presentarToast("Se ha eliminado el inmueble con éxito", stage);

//Se quita el inmueble de la vista

tablaInmuebles.getItems().remove(inmueble);

}

**return** **new** ResultadoControlador(erroresControlador.toArray(**new** ErrorControlador[0]));

}

Prueba de unidad del archivo AdministrarInmuebleControllerTest.java

//Casos de prueba //inmueble,acepta,resultadoControlador,resultadoLogica,excepcion

/\* 0 \*/**new** Object[] { inmueble, acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario acepta y el inmueble se elimina correctamente

/\* 1 \*/**new** Object[] { inmueble, !acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario no acepta, pero de haber aceptado, se hubiese eliminado el inmueble correctamente

/\* 2 \*/**new** Object[] { inmueble, acepta, resultadoControladorErrorPersistencia, **null**, excepcionPersistencia }, //test donde el controlador tira una excepción de persistencia

/\* 3 \*/**new** Object[] { inmueble, acepta, resultadoControladorErrorDesconocido, **null**, excepcionInesperada } //test donde el controlador tira una excepción inesperada

/\*\*

\* Prueba el método eliminarInmueble(), el cual corresponde con la taskcard 13 de la iteración 1 y a la historia 3

\*

\* **@param** inmueble

\* inmueble a crear

\* **@param** acepta

\* si el usuario acepta la eliminacion

\* **@param** resultadoControlador

\* resultado que se espera que devuelva el metodo a probar

\* **@param** resultadoLogica

\* resultado que devuelve el gestor

\* **@param** excepcion

\*/

@Test

@Parameters

**public** **void** testEliminarInmueble(Inmueble inmueble, Boolean acepta, ResultadoControlador resultadoControlador, ResultadoEliminarInmueble resultadoLogica, Throwable excepcion) **throws** Exception {

//Se crean los mocks de la prueba

CoordinadorJavaFX coordinadorMock = **new** CoordinadorJavaFX() {

@Override

**public** ResultadoEliminarInmueble eliminarInmueble(Inmueble inmbueble) **throws** PersistenciaException {

**if**(resultadoLogica != **null**){

**return** resultadoLogica;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

@Override

**public** ArrayList<Inmueble> obtenerInmuebles() **throws** PersistenciaException {

ArrayList<Inmueble> inmuebles = **new** ArrayList<>();

inmuebles.add(inmueble);

**return** inmuebles;

}

};

PresentadorVentanas presentadorMock = **new** PresentadorVentanasMock(acepta);

//se crea el controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos que ingresaría el usuario en la vista

AdministrarInmuebleController administrarInmuebleController = **new** AdministrarInmuebleController() {

@Override

**public** ResultadoControlador eliminarInmueble() {

tablaInmuebles.getSelectionModel().select(inmueble);

**return** **super**.eliminarInmueble();

}

@Override

**protected** **void** inicializar(URL location, ResourceBundle resources) {

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

administrarInmuebleController.setCoordinador(coordinadorMock);

administrarInmuebleController.setPresentador(presentadorMock);

//Se crea lo necesario para correr la prueba en el hilo de JavaFX porque los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarInmuebleController.***URLVista***, administrarInmuebleController);

administrarInmuebleController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

*assertEquals*(resultadoControlador, administrarInmuebleController.eliminarInmueble());

}

};

**try**{

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

Código del archivo NMVInmuebleController.java

/\*\*

\* Método que se llama al hacer click en el botón agregar foto.

\*/

@FXML

public void agregarFoto() {

//Se abre un cuadro de diálogo para pedir la ruta del archivo

File imagen = solicitarArchivo();

if(imagen == null){

return;

}

try{

//Se carga la imagen y se la muestra

final ImageView imageView = new ImageView(imagen.toURI().toURL().toExternalForm());

imageView.setPreserveRatio(true);

imageView.setFitHeight(100);

imageView.setOnMouseClicked((event) -> {

seleccionarImagen(imageView);

});

panelFotos.getChildren().add(imageView);

archivosImagenesNuevas.put(imageView, imagen);

} catch(MalformedURLException e){

//Si ocurre algún error se lo muestra en pantalla

presentador.presentarExcepcionInesperada(e, stage);

}

}

/\*\*

\* Método que se llama al hacer click en el botón quitar foto.

\*/

@FXML

**public** **void** quitarFoto() {

//Se quita la imagen de la vista

panelFotos.getChildren().remove(imagenSeleccionada);

**if**(archivosImagenesPreExistentes.get(imagenSeleccionada) != **null**){

archivosImagenesPreExistentes.remove(imagenSeleccionada);

}

imagenSeleccionada = **null**;

btQuitarFoto.setDisable(**true**);

}

/\*\*

\* Método que permite guardar los cambios hechos en la vista

\* Pertenece a la taskcard 13 de la iteración 1 y a la historia 3

\*

\* **@return** ResultadoControlador que resume lo que hizo el controlador

\*/

@FXML

**public** ResultadoControlador aceptar() {

ResultadoControlador resultado;

**if**(inmueble == **null**){

resultado = crearInmueble();

}

else{

resultado = modificarInmueble();

}

**if**(!resultado.hayErrores()){

salir();

}

**return** resultado;

}

/\*\*

\* Método que permite crear un inmueble

\* Pertenece a la taskcard 13 de la iteración 1 y a la historia 3

\*

\* **@return** ResultadoControlador que resume lo que hizo el controlador

\*/

**private** ResultadoControlador crearInmueble() {

//Inicialización de variables

Set<ErrorControlador> erroresControlador = **new** HashSet<>();

ResultadoCrearInmueble resultado;

StringBuffer erroresBfr = **new** StringBuffer();

Inmueble inmueble = **new** Inmueble();

//Toma de datos de la vista

DatosEdificio datos = **new** DatosEdificio()

.setSuperficie((!tfSuperficieEdificio.getText().isEmpty()) ? (Double.*parseDouble*(tfSuperficieEdificio.getText())) : (**null**))

.setAntiguedad((!tfAntiguedad.getText().isEmpty()) ? (Integer.*parseInt*(tfAntiguedad.getText())) : (**null**))

.setDormitorios((!tfDormitorios.getText().isEmpty()) ? (Integer.*parseInt*(tfDormitorios.getText())) : (**null**))

.setBaños((!tfBaños.getText().isEmpty()) ? (Integer.*parseInt*(tfBaños.getText())) : (**null**))

.setPropiedadHorizontal(cbPropiedadHorizontal.isSelected())

.setGaraje(cbGarage.isSelected())

.setPatio(cbPatio.isSelected())

.setPiscina(cbPiscina.isSelected())

.setAguaCorriente(cbAguaCorriente.isSelected())

.setCloacas(cbCloaca.isSelected())

.setGasNatural(cbGasNatural.isSelected())

.setAguaCaliente(cbAguaCaliente.isSelected())

.setTelefono(cbTelefono.isSelected())

.setLavadero(cbLavadero.isSelected())

.setPavimento(cbPavimento.isSelected())

.setInmueble(inmueble);

Localidad localidad = cbLocalidad.getValue();

Barrio barrio = cbBarrio.getValue();

Calle calle = cbCalle.getValue();

Date fechaCarga = **new** Date();

Orientacion orientacion = cbOrientacion.getValue();

Propietario propietario = cbPropietario.getValue();

TipoInmueble tipo = cbTipoInmueble.getValue();

Direccion direccion = **new** Direccion()

.setLocalidad(localidad)

.setCalle(calle)

.setNumero(tfAltura.getText().toLowerCase().trim())

.setBarrio(barrio)

.setDepartamento(tfDepartamento.getText().toLowerCase().trim())

.setOtros(tfOtros.getText().toLowerCase().trim())

.setPiso(tfPiso.getText().toLowerCase().trim());

//Guardar fotos

ArrayList<Imagen> fotos = **new** ArrayList<>();

**for**(Node nodo: panelFotos.getChildren()){

**if**(nodo **instanceof** ImageView){

ImageView imagen = (ImageView) nodo;

File file = archivosImagenesNuevas.get(imagen);

**if**(file != **null**){

**byte**[] bFile = **new** **byte**[(**int**) file.length()];

**try**{

FileInputStream fileInputStream = **new** FileInputStream(file);

//convert file into array of bytes

fileInputStream.read(bFile);

fileInputStream.close();

} **catch**(Exception e){

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

fotos.add((Imagen) **new** Imagen().setArchivo(bFile));

}

}

}

//Se cargan los datos de la vista al inmueble a guardar

inmueble.setDatosEdificio(datos)

.setFechaCarga(fechaCarga)

.setEstado(**new** Estado(EstadoStr.***ALTA***))

.setDireccion(direccion)

.setTipo(tipo)

.setOrientacion(orientacion)

.setPropietario(propietario)

.setPrecio((!tfPrecioVenta.getText().isEmpty()) ? (Double.*parseDouble*(tfPrecioVenta.getText())) : (**null**))

.setFrente((!tfFrente.getText().isEmpty()) ? (Double.*parseDouble*(tfFrente.getText())) : (**null**))

.setFondo((!tfFondo.getText().isEmpty()) ? (Double.*parseDouble*(tfFondo.getText())) : (**null**))

.setSuperficie((!tfSuperficie.getText().isEmpty()) ? (Double.*parseDouble*(tfSuperficie.getText())) : (**null**))

.setObservaciones(taObservaciones.getText())

.getFotos().addAll(fotos);

**try**{

//Se llama a la lógica para persistir el inmueble y se recibe el resultado de las validaciones y datos extras de ser necesarios

resultado = coordinador.crearInmueble(inmueble);

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

//Procesamiento de errores de la lógica

**if**(resultado.hayErrores()){

**for**(ErrorCrearInmueble err: resultado.getErrores()){

**switch**(err) {

case Fecha\_Vacia:

erroresBfr.append("Fecha no ingresada.\n");

erroresControlador.add(ErrorControlador.***Campos\_Vacios***);

**break**;

case Fondo\_Incorrecto:

erroresBfr.append("Formato del campo Fondo incorrecto.\n");

erroresControlador.add(ErrorControlador.***Datos\_Incorrectos***);

**break**;

case Formato\_Direccion\_Incorrecto:

erroresBfr.append("Formato de dirección incorrecto.\n");

erroresControlador.add(ErrorControlador.***Datos\_Incorrectos***);

**break**;

case Frente\_Incorrecto:

erroresBfr.append("Formato del campo Frente incorrecto.\n");

erroresControlador.add(ErrorControlador.***Datos\_Incorrectos***);

**break**;

case Precio\_Vacio:

erroresBfr.append("Precio no ingresado.\n");

erroresControlador.add(ErrorControlador.***Campos\_Vacios***);

**break**;

case Precio\_Incorrecto:

erroresBfr.append("Formato de precio incorrecto.\n");

erroresControlador.add(ErrorControlador.***Datos\_Incorrectos***);

**break**;

case Propietario\_Inexistente:

erroresBfr.append("El propietario seleccionado no existe en el sistema.\n");

erroresControlador.add(ErrorControlador.***Entidad\_No\_Encontrada***);

**break**;

case Propietario\_Vacio:

erroresBfr.append("Elija el propietario.\n");

erroresControlador.add(ErrorControlador.***Campos\_Vacios***);

**break**;

case Superficie\_Incorrecta:

erroresBfr.append("Formato superficie de Inmueble incorrecto.\n");

erroresControlador.add(ErrorControlador.***Datos\_Incorrectos***);

**break**;

case *Tipo\_Vacio*:

erroresBfr.append("Elija el tipo de Inmueble.\n");

erroresControlador.add(ErrorControlador.***Campos\_Vacios***);

**break**;

case Datos\_Edificio\_Incorrectos:

erroresBfr.append("Formato de los datos de edificio incorrectos.\n");

erroresControlador.add(ErrorControlador.***Datos\_Incorrectos***);

**break**;

}

}

//Se muestran los errores

presentador.presentarError("No se pudo crear el inmueble", erroresBfr.toString(), stage);

}

else{

//Se muestra una notificación de que se creó correctamente el inmueble

presentador.presentarToast("Se ha creado el inmueble con éxito", stage);

}

**return** **new** ResultadoControlador(erroresControlador.toArray(**new** ErrorControlador[0]));

}

/\*\*

\* Método que permite modificar un inmueble

\* Pertenece a la taskcard 13 de la iteración 1 y a la historia 3

\*

\* @return ResultadoControlador que resume lo que hizo el controlador

\*/

private ResultadoControlador modificarInmueble() {

//Inicialización de variables

ArrayList<ErrorControlador> erroresControlador = new ArrayList<>();

ResultadoModificarInmueble resultado;

StringBuffer erroresBfr = new StringBuffer();

//Toma de datos de la vista

DatosEdificio datos = new DatosEdificio()

.setSuperficie((!tfSuperficieEdificio.getText().isEmpty()) ? (Double.parseDouble(tfSuperficieEdificio.getText())) : (null))

.setAntiguedad((!tfAntiguedad.getText().isEmpty()) ? (Integer.parseInt(tfAntiguedad.getText())) : (null))

.setDormitorios((!tfDormitorios.getText().isEmpty()) ? (Integer.parseInt(tfDormitorios.getText())) : (null))

.setBaños((!tfBaños.getText().isEmpty()) ? (Integer.parseInt(tfBaños.getText())) : (null))

.setPropiedadHorizontal(cbPropiedadHorizontal.isSelected())

.setGaraje(cbGarage.isSelected())

.setPatio(cbPatio.isSelected())

.setPiscina(cbPiscina.isSelected())

.setAguaCorriente(cbAguaCorriente.isSelected())

.setCloacas(cbCloaca.isSelected())

.setGasNatural(cbGasNatural.isSelected())

.setAguaCaliente(cbAguaCaliente.isSelected())

.setTelefono(cbTelefono.isSelected())

.setLavadero(cbLavadero.isSelected())

.setPavimento(cbPavimento.isSelected())

.setInmueble(inmueble);

Localidad localidad = cbLocalidad.getValue();

Barrio barrio = cbBarrio.getValue();

Calle calle = cbCalle.getValue();

Orientacion orientacion = cbOrientacion.getValue();

Propietario propietario = cbPropietario.getValue();

TipoInmueble tipo = cbTipoInmueble.getValue();

Direccion direccion = new Direccion()

.setLocalidad(localidad)

.setCalle(calle)

.setNumero(tfAltura.getText().toLowerCase().trim())

.setBarrio(barrio)

.setDepartamento(tfDepartamento.getText().toLowerCase().trim())

.setOtros(tfOtros.getText().toLowerCase().trim())

.setPiso(tfPiso.getText().toLowerCase().trim());

//Fotos eliminadas

ArrayList<Imagen> imagenesEliminadas = new ArrayList<>(inmueble.getFotos());

imagenesEliminadas.removeAll(archivosImagenesPreExistentes.values());

//Guardar fotos

ArrayList<Imagen> fotos = new ArrayList<>();

for(Node nodo: panelFotos.getChildren()){

if(nodo instanceof ImageView){

ImageView imagen = (ImageView) nodo;

File file = archivosImagenesNuevas.get(imagen);

if(file != null){

byte[] bFile = new byte[(int) file.length()];

try{

FileInputStream fileInputStream = new FileInputStream(file);

//convert file into array of bytes

fileInputStream.read(bFile);

fileInputStream.close();

} catch(Exception e){

presentador.presentarExcepcion(e, stage);

return new ResultadoControlador(ErrorControlador.Error\_Desconocido);

}

fotos.add((Imagen) new Imagen().setArchivo(bFile));

}

}

}

//Se cargan los datos de la vista al inmueble a modificar

inmueble.getFotos().removeAll(imagenesEliminadas);

inmueble.setDatosEdificio(datos)

.setDireccion(direccion)

.setTipo(tipo)

.setOrientacion(orientacion)

.setPropietario(propietario)

.setPrecio((!tfPrecioVenta.getText().isEmpty()) ? (Double.parseDouble(tfPrecioVenta.getText())) : (null))

.setFrente((!tfFrente.getText().isEmpty()) ? (Double.parseDouble(tfFrente.getText())) : (null))

.setFondo((!tfFondo.getText().isEmpty()) ? (Double.parseDouble(tfFondo.getText())) : (null))

.setSuperficie((!tfSuperficie.getText().isEmpty()) ? (Double.parseDouble(tfSuperficie.getText())) : (null))

.setObservaciones(taObservaciones.getText())

.getFotos().addAll(fotos);

try{

//Se llama a la lógica para persistir el inmueble modificado y se recibe el resultado de las validaciones y datos extras de ser necesarios

resultado = coordinador.modificarInmueble(inmueble);

} catch(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

return new ResultadoControlador(ErrorControlador.Error\_Persistencia);

} catch(Exception e){

presentador.presentarExcepcionInesperada(e, stage);

return new ResultadoControlador(ErrorControlador.Error\_Desconocido);

}

//Procesamiento de errores de la lógica

if(resultado.hayErrores()){

for(ErrorModificarInmueble err: resultado.getErrores()){

switch(err) {

case Fecha\_Vacia:

erroresBfr.append("Fecha no ingresada.\n");

erroresControlador.add(ErrorControlador.Campos\_Vacios);

break;

case Fondo\_Incorrecto:

erroresBfr.append("Formato del campo Fondo incorrecto.\n");

erroresControlador.add(ErrorControlador.Datos\_Incorrectos);

break;

case Formato\_Direccion\_Incorrecto:

erroresBfr.append("Formato de dirección incorrecto.\n");

erroresControlador.add(ErrorControlador.Datos\_Incorrectos);

break;

case Frente\_Incorrecto:

erroresBfr.append("Formato del campo Frente incorrecto.\n");

erroresControlador.add(ErrorControlador.Datos\_Incorrectos);

break;

case Precio\_Vacio:

erroresBfr.append("Precio no ingresado.\n");

erroresControlador.add(ErrorControlador.Campos\_Vacios);

break;

case Precio\_Incorrecto:

erroresBfr.append("Formato de precio incorrecto.\n");

erroresControlador.add(ErrorControlador.Datos\_Incorrectos);

break;

case Propietario\_Inexistente:

erroresBfr.append("El propietario seleccionado no existe en el sistema.\n");

erroresControlador.add(ErrorControlador.Entidad\_No\_Encontrada);

break;

case Propietario\_Vacio:

erroresBfr.append("Elija el propietario.\n");

erroresControlador.add(ErrorControlador.Campos\_Vacios);

break;

case Superficie\_Incorrecta:

erroresBfr.append("Formato superficie de Inmueble incorrecto.\n");

erroresControlador.add(ErrorControlador.Datos\_Incorrectos);

break;

case Tipo\_Vacio:

erroresBfr.append("Elija el tipo de Inmueble.\n");

erroresControlador.add(ErrorControlador.Campos\_Vacios);

break;

case Datos\_Edificio\_Incorrectos:

erroresBfr.append("Formato de los datos de edificio incorrectos.\n");

erroresControlador.add(ErrorControlador.Datos\_Incorrectos);

break;

case Inmueble\_Inexistente:

erroresBfr.append("El inmueble ya no existe en el sistema");

erroresControlador.add(ErrorControlador.Entidad\_No\_Encontrada);

break;

}

}

//Se muestran los errores

presentador.presentarError("No se pudo modificar el inmueble", erroresBfr.toString(), stage);

}

else{

//Se muestra una notificación de que se modificó correctamente el inmueble

presentador.presentarToast("Se ha modificado el inmueble con éxito", stage);

}

return new ResultadoControlador(erroresControlador.toArray(new ErrorControlador[0]));

}

Prueba de unidad del archivo NMVInmuebleControllerTest.java

//Casos de prueba

//propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControlador, resultadoLogica, excepcion

/\* 0 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test Todo correcto

/\* 1 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaDatosEdificioIncorrectos, **null** }, //test Datos de edificio incorrectos

/\* 2 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCamposVacios, resultadoLogicaFechaVacia, **null** }, //test fecha vacía

/\* 3 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaFondoIncorrecto, **null** }, //test formato de fondo incorrecto

/\* 4 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaFormatoDireccionIncorrecto, **null** }, //test formato de dirección incorrecto

/\* 5 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaFrenteIncorrecto, **null** }, //test formato de frente incorrecto

/\* 6 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaPrecioIncorrecto, **null** }, //test formato de precio incorrecto

/\* 7 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCamposVacios, resultadoLogicaPrecioVacio, **null** }, //test precio vacío

/\* 8 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorEntidadNoEncontrada, resultadoLogicaPropietarioInexistente, **null** }, //test Propietario seleccionado inexistente en el sistema

/\* 9 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCamposVacios, resultadoLogicaPropietarioVacio, **null** }, //test propietario vacío

/\* 10 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaSuperficieIncorrecta, **null** }, //test formato de superficie de inmueble incorrecto

/\* 11 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCamposVacios, resultadoLogicaTipoVacio, **null** }, //test tipo de inmueble vacío

/\* 12 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectosYCamposVacios, resultadoLogicaTipoVacioYSuperficieIncorrecta, **null** }, //test tipo de inmueble vacío y formato de superficie de inmueble incorrecto

/\* 13 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorErrorPersistencia, **null**, excepcionPersistencia }, //test excepción de persistencia

/\* 14 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorErrorDesconocido, **null**, excepcionInesperada } //test escepción inesperada

/\*\*

\* Prueba el método crearInmueble(), el cual corresponde con la taskcard 13 de la iteración 1 y a la historia 3

\*

\* **@param** propietario

\* que se usará en el test

\* **@param** direccion

\* que se usará en el test

\* **@param** tipoInmueble

\* que se usará en el test

\* **@param** precio

\* que se usará en el test

\* **@param** orientacion

\* que se usará en el test

\* **@param** frente

\* que se usará en el test

\* **@param** fondo

\* que se usará en el test

\* **@param** superficie

\* que se usará en el test

\* **@param** propiedadHorizontal

\* que se usará en el test

\* **@param** superficieEdificio

\* que se usará en el test

\* **@param** antigüedadEdificio

\* que se usará en el test

\* **@param** dormitorios

\* que se usarán en el test

\* **@param** baños

\* que se usarán en el test

\* **@param** garaje

\* que se usará en el test

\* **@param** patio

\* que se usará en el test

\* **@param** piscina

\* que se usará en el test

\* **@param** aguaCorriente

\* que se usará en el test

\* **@param** cloacas

\* que se usarán en el test

\* **@param** gasNatural

\* que se usará en el test

\* **@param** aguaCaliente

\* que se usará en el test

\* **@param** teléfono

\* que se usará en el test

\* **@param** lavadero

\* que se usará en el test

\* **@param** pavimento

\* que se usará en el test

\* **@param** observaciones

\* que se usarán en el test

\* **@param** resultadoControlador

\* es lo que se espera que devuelva el metodo

\* **@param** resultadoLogica

\* es lo que el mock de la lógica debe devolver en el test y que el controlador recibe

\* **@param** excepcion

\* es la excepcion que debe lanzar el mock de la lógica, si la prueba involucra procesar una excepcion de dicha lógica, debe ser nulo resultadoLogica para que se use

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testCrearInmueble(Propietario propietario,

Direccion direccion,

TipoInmueble tipoInmueble,

Double precio,

Orientacion orientacion,

Double frente,

Double fondo,

Double superficie,

Boolean propiedadHorizontal,

Double superficieEdificio,

Integer antigüedadEdificio,

Integer dormitorios,

Integer baños,

Boolean garaje,

Boolean patio,

Boolean piscina,

Boolean aguaCorriente,

Boolean cloacas,

Boolean gasNatural,

Boolean aguaCaliente,

Boolean teléfono,

Boolean lavadero,

Boolean pavimento,

String observaciones,

ResultadoControlador resultadoControlador,

ResultadoCrearInmueble resultadoLogica,

Throwable excepcion) **throws** Exception {

//Se crean los mocks de la prueba

CoordinadorJavaFX coordinadorMock = **new** CoordinadorJavaFX() {

@Override

**public** ResultadoCrearInmueble crearInmueble(Inmueble inmueble) **throws** PersistenciaException {

**if**(resultadoLogica != **null**){

**return** resultadoLogica;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

@Override

**public** ArrayList<Pais> obtenerPaises() **throws** PersistenciaException {

ArrayList<Pais> objetos = **new** ArrayList<>();

objetos.add(direccion.getLocalidad().getProvincia().getPais());

**return** objetos;

}

@Override

**public** ArrayList<Provincia> obtenerProvinciasDe(Pais pais) **throws** PersistenciaException {

ArrayList<Provincia> objetos = **new** ArrayList<>();

objetos.add(direccion.getLocalidad().getProvincia());

**return** objetos;

}

@Override

**public** ArrayList<Localidad> obtenerLocalidadesDe(Provincia prov) **throws** PersistenciaException {

ArrayList<Localidad> objetos = **new** ArrayList<>();

objetos.add(direccion.getLocalidad());

**return** objetos;

}

@Override

**public** ArrayList<Calle> obtenerCallesDe(Localidad localidad) **throws** PersistenciaException {

ArrayList<Calle> objetos = **new** ArrayList<>();

objetos.add(direccion.getCalle());

**return** objetos;

}

@Override

**public** ArrayList<Barrio> obtenerBarriosDe(Localidad localidad) **throws** PersistenciaException {

ArrayList<Barrio> objetos = **new** ArrayList<>();

objetos.add(direccion.getBarrio());

**return** objetos;

}

@Override

**public** ArrayList<Propietario> obtenerPropietarios() **throws** PersistenciaException {

ArrayList<Propietario> objetos = **new** ArrayList<>();

objetos.add(propietario);

**return** objetos;

}

@Override

**public** ArrayList<TipoInmueble> obtenerTiposInmueble() **throws** PersistenciaException {

ArrayList<TipoInmueble> objetos = **new** ArrayList<>();

objetos.add(tipoInmueble);

**return** objetos;

}

@Override

**public** ArrayList<Orientacion> obtenerOrientaciones() **throws** PersistenciaException {

ArrayList<Orientacion> objetos = **new** ArrayList<>();

objetos.add(orientacion);

**return** objetos;

}

};

PresentadorVentanas presentadorMock = **new** PresentadorVentanasMock();

//se crea el controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos que ingresaría el usuario en la vista

NMVInmuebleController nMVInmuebleController = **new** NMVInmuebleController() {

@Override

**public** ResultadoControlador aceptar() {

cbAguaCaliente.setSelected(aguaCaliente);

cbAguaCorriente.setSelected(aguaCorriente);

cbCloaca.setSelected(cloacas);

cbGarage.setSelected(garaje);

cbGasNatural.setSelected(gasNatural);

cbLavadero.setSelected(lavadero);

cbPatio.setSelected(patio);

cbPavimento.setSelected(pavimento);

cbPiscina.setSelected(piscina);

cbPropiedadHorizontal.setSelected(propiedadHorizontal);

cbTelefono.setSelected(teléfono);

cbPais.getSelectionModel().select(**null**);

cbPais.getSelectionModel().select(direccion.getLocalidad().getProvincia().getPais());

cbProvincia.getSelectionModel().select(direccion.getLocalidad().getProvincia());

cbLocalidad.getSelectionModel().select(direccion.getLocalidad());

cbBarrio.getSelectionModel().select(direccion.getBarrio());

cbCalle.getSelectionModel().select(direccion.getCalle());

cbOrientacion.getSelectionModel().select(orientacion);

cbPropietario.getSelectionModel().select(propietario);

cbTipoInmueble.getSelectionModel().select(tipoInmueble);

taObservaciones.setText(observaciones);

tfAltura.setText(direccion.getNumero());

tfAntiguedad.setText(antigüedadEdificio.toString());

tfBaños.setText(baños.toString());

tfCodigo.setText(**new** Integer(1).toString());

tfDepartamento.setText(direccion.getDepartamento());

tfDormitorios.setText(dormitorios.toString());

tfFechaCarga.setText(**new** Date().toString());

tfFondo.setText(fondo.toString());

tfFrente.setText(frente.toString());

tfOtros.setText(direccion.getOtros());

tfPiso.setText(direccion.getPiso());

tfPrecioVenta.setText(precio.toString());

tfSuperficie.setText(superficie.toString());

tfSuperficieEdificio.setText(superficieEdificio.toString());

**return** **super**.aceptar();

}

@Override

**protected** **void** inicializar(URL location, ResourceBundle resources) {

}

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** salir() {

}

};

nMVInmuebleController.setCoordinador(coordinadorMock);

nMVInmuebleController.setPresentador(presentadorMock);

//Se crea lo necesario para correr la prueba en el hilo de JavaFX porque los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(NMVInmuebleController.***URLVista***, nMVInmuebleController);

nMVInmuebleController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

*assertEquals*(resultadoControlador, nMVInmuebleController.aceptar());

}

};

**try**{

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

//Casos de prueba

//propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControlador, resultadoLogica, excepcion

/\* 0 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test Todo correcto

/\* 1 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaDatosEdificioIncorrectos, **null** }, //test Datos de edificio incorrectos

/\* 2 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCamposVacios, resultadoLogicaFechaVacia, **null** }, //test fecha vacía

/\* 3 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaFondoIncorrecto, **null** }, //test formato de fondo incorrecto

/\* 4 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaFormatoDireccionIncorrecto, **null** }, //test formato de dirección incorrecto

/\* 5 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaFrenteIncorrecto, **null** }, //test formato de frente incorrecto

/\* 6 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaPrecioIncorrecto, **null** }, //test formato de precio incorrecto

/\* 7 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCamposVacios, resultadoLogicaPrecioVacio, **null** }, //test precio vacío

/\* 8 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorEntidadNoEncontrada, resultadoLogicaPropietarioInexistente, **null** }, //test Propietario seleccionado inexistente en el sistema

/\* 9 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCamposVacios, resultadoLogicaPropietarioVacio, **null** }, //test propietario vacío

/\* 10 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectos, resultadoLogicaSuperficieIncorrecta, **null** }, //test formato de superficie de inmueble incorrecto

/\* 11 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorCamposVacios, resultadoLogicaTipoVacio, **null** }, //test tipo de inmueble vacío

/\* 12 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorEntidadNoEncontrada, resultadoLogicaInmuebleInexistente, **null** }, //test inmueble eleccionado inexistente en el sistema

/\* 13 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorDatosIncorrectosYCamposVacios, resultadoLogicaTipoVacioYSuperficieIncorrecta, **null** }, //test tipo de inmueble vacío y formato de superficie de inmueble incorrecto

/\* 14 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorErrorPersistencia, **null**, excepcionPersistencia }, //test excepción de persistencia

/\* 15 \*/**new** Object[] { propietario, direccion, tipoInmueble, precio, orientacion, frente, fondo, superficie, propiedadHorizontal, superficieEdificio, antigüedadEdificio, dormitorios, baños, garaje, patio, piscina, aguaCorriente, cloacas, gasNatural, aguaCaliente, teléfono, lavadero, pavimento, observaciones, resultadoControladorErrorDesconocido, **null**, excepcionInesperada } //test escepción inesperada

/\*\*

\* Prueba el método modificarInmueble(), el cual corresponde con la taskcard 13 de la iteración 1 y a la historia 3

\*

\* **@param** propietario

\* que se usará en el test

\* **@param** direccion

\* que se usará en el test

\* **@param** tipoInmueble

\* que se usará en el test

\* **@param** precio

\* que se usará en el test

\* **@param** orientacion

\* que se usará en el test

\* **@param** frente

\* que se usará en el test

\* **@param** fondo

\* que se usará en el test

\* **@param** superficie

\* que se usará en el test

\* **@param** propiedadHorizontal

\* que se usará en el test

\* **@param** superficieEdificio

\* que se usará en el test

\* **@param** antigüedadEdificio

\* que se usará en el test

\* **@param** dormitorios

\* que se usarán en el test

\* **@param** baños

\* que se usarán en el test

\* **@param** garaje

\* que se usará en el test

\* **@param** patio

\* que se usará en el test

\* **@param** piscina

\* que se usará en el test

\* **@param** aguaCorriente

\* que se usará en el test

\* **@param** cloacas

\* que se usarán en el test

\* **@param** gasNatural

\* que se usará en el test

\* **@param** aguaCaliente

\* que se usará en el test

\* **@param** teléfono

\* que se usará en el test

\* **@param** lavadero

\* que se usará en el test

\* **@param** pavimento

\* que se usará en el test

\* **@param** observaciones

\* que se usarán en el test

\* **@param** resultadoControlador

\* es lo que se espera que devuelva el metodo

\* **@param** resultadoLogica

\* es lo que el mock de la lógica debe devolver en el test y que el controlador recibe

\* **@param** excepcion

\* es la excepcion que debe lanzar el mock de la lógica, si la prueba involucra procesar una excepcion de dicha lógica, debe ser nulo resultadoLogica para que se use

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testModificarInmueble(Propietario propietario,

Direccion direccion,

TipoInmueble tipoInmueble,

Double precio,

Orientacion orientacion,

Double frente,

Double fondo,

Double superficie,

Boolean propiedadHorizontal,

Double superficieEdificio,

Integer antigüedadEdificio,

Integer dormitorios,

Integer baños,

Boolean garaje,

Boolean patio,

Boolean piscina,

Boolean aguaCorriente,

Boolean cloacas,

Boolean gasNatural,

Boolean aguaCaliente,

Boolean teléfono,

Boolean lavadero,

Boolean pavimento,

String observaciones,

ResultadoControlador resultadoControlador,

ResultadoModificarInmueble resultadoLogica,

Throwable excepcion) **throws** Exception {

//Se crean los mocks de la prueba

CoordinadorJavaFX coordinadorMock = **new** CoordinadorJavaFX() {

@Override

**public** ResultadoModificarInmueble modificarInmueble(Inmueble inmueble) **throws** PersistenciaException {

**if**(resultadoLogica != **null**){

**return** resultadoLogica;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

@Override

**public** ArrayList<Pais> obtenerPaises() **throws** PersistenciaException {

ArrayList<Pais> objetos = **new** ArrayList<>();

objetos.add(direccion.getLocalidad().getProvincia().getPais());

**return** objetos;

}

@Override

**public** ArrayList<Provincia> obtenerProvinciasDe(Pais pais) **throws** PersistenciaException {

ArrayList<Provincia> objetos = **new** ArrayList<>();

objetos.add(direccion.getLocalidad().getProvincia());

**return** objetos;

}

@Override

**public** ArrayList<Localidad> obtenerLocalidadesDe(Provincia prov) **throws** PersistenciaException {

ArrayList<Localidad> objetos = **new** ArrayList<>();

objetos.add(direccion.getLocalidad());

**return** objetos;

}

@Override

**public** ArrayList<Calle> obtenerCallesDe(Localidad localidad) **throws** PersistenciaException {

ArrayList<Calle> objetos = **new** ArrayList<>();

objetos.add(direccion.getCalle());

**return** objetos;

}

@Override

**public** ArrayList<Barrio> obtenerBarriosDe(Localidad localidad) **throws** PersistenciaException {

ArrayList<Barrio> objetos = **new** ArrayList<>();

objetos.add(direccion.getBarrio());

**return** objetos;

}

@Override

**public** ArrayList<Propietario> obtenerPropietarios() **throws** PersistenciaException {

ArrayList<Propietario> objetos = **new** ArrayList<>();

objetos.add(propietario);

**return** objetos;

}

@Override

**public** ArrayList<TipoInmueble> obtenerTiposInmueble() **throws** PersistenciaException {

ArrayList<TipoInmueble> objetos = **new** ArrayList<>();

objetos.add(tipoInmueble);

**return** objetos;

}

@Override

**public** ArrayList<Orientacion> obtenerOrientaciones() **throws** PersistenciaException {

ArrayList<Orientacion> objetos = **new** ArrayList<>();

objetos.add(orientacion);

**return** objetos;

}

};

PresentadorVentanas presentadorMock = **new** PresentadorVentanasMock();

//Se crea el controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos que ingresaría el usuario en la vista

NMVInmuebleController nMVInmuebleController = **new** NMVInmuebleController() {

{

**this**.inmueble = **new** Inmueble();

}

@Override

**public** ResultadoControlador aceptar() {

cbAguaCaliente.setSelected(aguaCaliente);

cbAguaCorriente.setSelected(aguaCorriente);

cbCloaca.setSelected(cloacas);

cbGarage.setSelected(garaje);

cbGasNatural.setSelected(gasNatural);

cbLavadero.setSelected(lavadero);

cbPatio.setSelected(patio);

cbPavimento.setSelected(pavimento);

cbPiscina.setSelected(piscina);

cbPropiedadHorizontal.setSelected(propiedadHorizontal);

cbTelefono.setSelected(teléfono);

cbPais.getSelectionModel().select(**null**);

cbPais.getSelectionModel().select(direccion.getLocalidad().getProvincia().getPais());

cbProvincia.getSelectionModel().select(direccion.getLocalidad().getProvincia());

cbLocalidad.getSelectionModel().select(direccion.getLocalidad());

cbBarrio.getSelectionModel().select(direccion.getBarrio());

cbCalle.getSelectionModel().select(direccion.getCalle());

cbOrientacion.getSelectionModel().select(orientacion);

cbPropietario.getSelectionModel().select(propietario);

cbTipoInmueble.getSelectionModel().select(tipoInmueble);

taObservaciones.setText(observaciones);

tfAltura.setText(direccion.getNumero());

tfAntiguedad.setText(antigüedadEdificio.toString());

tfBaños.setText(baños.toString());

tfCodigo.setText(**new** Integer(1).toString());

tfDepartamento.setText(direccion.getDepartamento());

tfDormitorios.setText(dormitorios.toString());

tfFechaCarga.setText(**new** Date().toString());

tfFondo.setText(fondo.toString());

tfFrente.setText(frente.toString());

tfOtros.setText(direccion.getOtros());

tfPiso.setText(direccion.getPiso());

tfPrecioVenta.setText(precio.toString());

tfSuperficie.setText(superficie.toString());

tfSuperficieEdificio.setText(superficieEdificio.toString());

**return** **super**.aceptar();

}

@Override

**protected** **void** inicializar(URL location, ResourceBundle resources) {

}

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** salir() {

}

};

nMVInmuebleController.setCoordinador(coordinadorMock);

nMVInmuebleController.setPresentador(presentadorMock);

//Se crea lo necesario para correr la prueba en el hilo de JavaFX porque los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(NMVInmuebleController.***URLVista***, nMVInmuebleController);

nMVInmuebleController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

*assertEquals*(resultadoControlador, nMVInmuebleController.aceptar());

}

};

**try**{

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

### Taskcard 14 Lógica alta, modificación y baja inmueble

Código del archivo PropietarioServiceImpl.java

/\*\*

\* Método para crear un inmueble. Primero se validan las reglas de negocia y luego se persiste.

\* Pertenece a la taskcard 14 de la iteración 1 y a la historia 3

\*

\* **@param** inmueble

\* a guardar

\* **@return** resultado de la operación

\* **@throws** PersistenciaException

\* si falló al persistir

\*/

**public** ResultadoCrearInmueble crearInmueble(Inmueble inmueble) **throws** PersistenciaException {

Set<ErrorCrearInmueble> errores = **new** HashSet<>();

validarInmuebleAlta(inmueble, errores);

**if**(errores.isEmpty()){

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***ALTA***)){

inmueble.setEstado(e);

**break**;

}

}

persistidorInmueble.guardarInmueble(inmueble);

}

**return** **new** ResultadoCrearInmueble(errores.toArray(**new** ErrorCrearInmueble[0]));

}

/\*\*

\* Método auxiliar que encapsula la validación de los atributos del inmueble a crear

\*

\* **@param** inmueble

\* inmueble a validar

\* **@param** errores

\* lista de errores encontrados

\*/

**private** **void** validarInmuebleAlta(Inmueble inmueble, Set<ErrorCrearInmueble> errores) **throws** PersistenciaException {

**if**(inmueble.getFechaCarga() == **null**){

errores.add(ErrorCrearInmueble.***Fecha\_Vacia***);

}

**if**(inmueble.getPropietario() != **null**){

Propietario propietario = gestorPropietario.obtenerPropietario(**new** FiltroPropietario(inmueble.getPropietario().getTipoDocumento().getTipo(), inmueble.getPropietario().getNumeroDocumento()));

**if**(propietario == **null**){

errores.add(ErrorCrearInmueble.***Propietario\_Inexistente***);

}

}

**else**{

errores.add(ErrorCrearInmueble.***Propietario\_Vacio***);

}

**if**(inmueble.getPrecio() == **null**){

errores.add(ErrorCrearInmueble.***Precio\_Vacio***);

}

**else** **if**(!validador.validarDoublePositivo(inmueble.getPrecio())){

errores.add(ErrorCrearInmueble.***Precio\_Incorrecto***);

}

**if**(inmueble.getFondo() != **null** && !validador.validarDoublePositivo(inmueble.getFondo())){

errores.add(ErrorCrearInmueble.***Fondo\_Incorrecto***);

}

**if**(inmueble.getFrente() != **null** && !validador.validarDoublePositivo(inmueble.getFrente())){

errores.add(ErrorCrearInmueble.***Frente\_Incorrecto***);

}

**if**(inmueble.getSuperficie() != **null** && !validador.validarDoublePositivo(inmueble.getSuperficie())){

errores.add(ErrorCrearInmueble.***Superficie\_Incorrecta***);

}

**if**(inmueble.getTipo() == **null**){

errores.add(ErrorCrearInmueble.***Tipo\_Vacio***);

}

**if**(!validador.validarDireccion(inmueble.getDireccion())){

errores.add(ErrorCrearInmueble.***Formato\_Direccion\_Incorrecto***);

}

**if**(inmueble.getDireccion() == **null** || inmueble.getDireccion().getBarrio() == **null** || inmueble.getDireccion().getBarrio().getLocalidad() == **null** || inmueble.getDireccion().getLocalidad() == **null** || inmueble.getDireccion().getLocalidad().getProvincia() == **null** || inmueble.getDireccion().getLocalidad().getProvincia().getPais() == **null** || inmueble.getDireccion().getCalle() == **null** || inmueble.getDireccion().getCalle().getLocalidad() == **null**){

errores.add(ErrorCrearInmueble.***Formato\_Direccion\_Incorrecto***);

}

**if**(!validarDatosEdificio(inmueble.getDatosEdificio())){

errores.add(ErrorCrearInmueble.***Datos\_Edificio\_Incorrectos***);

}

}

/\*\*

\* Método que se encarga de la modificación de los datos de un inmueble.

\* Se validan todos los datos correspondientes al inmueble, corroborando los que son obligatorios.

\* Se valida que exista el inmueble en la base de datos y en caso de que no haya errores, delega el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** inmueble

\* inmueble a modificar.

\*

\* **@return** un resultado informando errores correspondientes en caso de que los haya.

\*

\* **@throws** PersistenciaException

\* Se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos.

\*/

**public** ResultadoModificarInmueble modificarInmueble(Inmueble inmueble) **throws** PersistenciaException {

ArrayList<ErrorModificarInmueble> errores = **new** ArrayList<>();

validarInmuebleModificar(inmueble, errores);

Inmueble inmuebleAuxiliar = persistidorInmueble.obtenerInmueble(inmueble.getId());

**if**(inmuebleAuxiliar == **null**){

errores.add(ErrorModificarInmueble.***Inmueble\_Inexistente***);

}

**if**(errores.isEmpty()){

HistorialInmueble historialInmuebleAuxiliar = **new** HistorialInmueble();

DatosEdificio datosEdificioAuxiliar = inmuebleAuxiliar.getDatosEdificio();

HistorialDatosEdificio historialDatosEdificioAuxiliar = **new** HistorialDatosEdificio()

.setAguaCaliente(datosEdificioAuxiliar.getAguaCaliente())

.setAguaCorriente(datosEdificioAuxiliar.getAguaCaliente())

.setAntiguedad(datosEdificioAuxiliar.getAntiguedad())

.setBaños(datosEdificioAuxiliar.getBaños())

.setCloacas(datosEdificioAuxiliar.getCloacas())

.setDormitorios(datosEdificioAuxiliar.getDormitorios())

.setGaraje(datosEdificioAuxiliar.getGaraje())

.setGasNatural(datosEdificioAuxiliar.getGasNatural())

.setLavadero(datosEdificioAuxiliar.getLavadero())

.setPatio(datosEdificioAuxiliar.getPatio())

.setPavimento(datosEdificioAuxiliar.getPavimento())

.setPiscina(datosEdificioAuxiliar.getPiscina())

.setPropiedadHorizontal(datosEdificioAuxiliar.getPropiedadHorizontal())

.setSuperficie(datosEdificioAuxiliar.getSuperficie())

.setTelefono(datosEdificioAuxiliar.getTelefono())

.setDatosEdificio(datosEdificioAuxiliar)

.setHistorialInmueble(historialInmuebleAuxiliar);

historialInmuebleAuxiliar

.setDireccion(inmuebleAuxiliar.getDireccion())

.setEstado(inmuebleAuxiliar.getEstado())

.setFechaCarga(inmuebleAuxiliar.getFechaCarga())

.setFechaYHoraCambio(**new** Date())

.setFondo(inmuebleAuxiliar.getFondo())

.setFrente(inmuebleAuxiliar.getFrente())

.setObservaciones(inmuebleAuxiliar.getObservaciones())

.setOrientacion(inmuebleAuxiliar.getOrientacion())

.setPrecio(inmuebleAuxiliar.getPrecio())

.setPropietario(inmuebleAuxiliar.getPropietario())

.setSuperficie(inmuebleAuxiliar.getSuperficie())

.setTipo(inmuebleAuxiliar.getTipo())

.setInmueble(inmuebleAuxiliar)

.setHistorialDatosEdificio(historialDatosEdificioAuxiliar)

.getFotos().addAll(inmuebleAuxiliar.getFotos());

persistidorHistorial.guardarHistorialInmueble(historialInmuebleAuxiliar);

persistidorInmueble.modificarInmueble(inmueble);

}

**return** **new** ResultadoModificarInmueble(errores.toArray(**new** ErrorModificarInmueble[0]));

}

/\*\*

\* Método auxiliar que encapsula la validación de los atributos del inmueble a modificar

\*

\* **@param** inmueble

\* inmueble a validar

\* **@param** errores

\* lista de errores encontrados

\*/

**private** **void** validarInmuebleModificar(Inmueble inmueble, ArrayList<ErrorModificarInmueble> errores)

**throws** PersistenciaException {

**if**(inmueble.getFechaCarga() == **null**){

errores.add(ErrorModificarInmueble.***Fecha\_Vacia***);

}

**if**(inmueble.getPropietario() != **null**){

Propietario propietario = gestorPropietario.obtenerPropietario(**new** FiltroPropietario(inmueble.getPropietario().getTipoDocumento().getTipo(), inmueble.getPropietario().getNumeroDocumento()));

**if**(propietario == **null**){

errores.add(ErrorModificarInmueble.***Propietario\_Inexistente***);

}

}

**else**{

errores.add(ErrorModificarInmueble.***Propietario\_Vacio***);

}

**if**(inmueble.getPrecio() == **null**){

errores.add(ErrorModificarInmueble.***Precio\_Vacio***);

}

**if**(!validador.validarDoublePositivo(inmueble.getPrecio())){

errores.add(ErrorModificarInmueble.***Precio\_Incorrecto***);

}

**if**(inmueble.getFondo() != **null** && !validador.validarDoublePositivo(inmueble.getFondo())){

errores.add(ErrorModificarInmueble.***Fondo\_Incorrecto***);

}

**if**(inmueble.getFrente() != **null** && !validador.validarDoublePositivo(inmueble.getFrente())){

errores.add(ErrorModificarInmueble.***Frente\_Incorrecto***);

}

**if**(inmueble.getSuperficie() != **null** && !validador.validarDoublePositivo(inmueble.getSuperficie())){

errores.add(ErrorModificarInmueble.***Superficie\_Incorrecta***);

}

**if**(inmueble.getTipo() == **null**){

errores.add(ErrorModificarInmueble.***Tipo\_Vacio***);

}

**if**(!validador.validarDireccion(inmueble.getDireccion())){

errores.add(ErrorModificarInmueble.***Formato\_Direccion\_Incorrecto***);

}

**if**(!validarDatosEdificio(inmueble.getDatosEdificio())){

errores.add(ErrorModificarInmueble.***Datos\_Edificio\_Incorrectos***);

}

}

/\*\*

\* Se encarga de validar que exista el inmueble a eliminar, se setea el estado en BAJA y,

\* en caso de que no haya errores, delegar el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** inmueble

\* inmueble a eliminar

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ResultadoEliminarInmueble eliminarInmueble(Inmueble inmueble) **throws** PersistenciaException {

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***BAJA***)){

inmueble.setEstado(e);

}

}

persistidorInmueble.modificarInmueble(inmueble);

**return** **new** ResultadoEliminarInmueble();

}

/\*\*

\* Obtiene el listado de inmuebles solicitándola a la capa de acceso a datos

\*

\* **@return** el listado de inmuebles solicitados

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ArrayList<Inmueble> obtenerInmuebles() **throws** PersistenciaException {

**return** persistidorInmueble.listarInmuebles();

}

/\*\*

\* Método auxiliar que encapsula la validación de los atributos del edificio

\*

\* **@param** datosEdificio

\* datos del edificio a validar

\*

\* **@return** True si los datos son correctos

\* False si no son correctos

\*/

**protected** Boolean validarDatosEdificio(DatosEdificio datosEdificio) {

**if**(datosEdificio == **null**){

**return** **false**;

}

**if**(datosEdificio.getAguaCaliente() == **null**){

**return** **false**;

}

**if**(datosEdificio.getAguaCorriente() == **null**){

**return** **false**;

}

**if**(datosEdificio.getCloacas() == **null**){

**return** **false**;

}

**if**(datosEdificio.getGaraje() == **null**){

**return** **false**;

}

**if**(datosEdificio.getGasNatural() == **null**){

**return** **false**;

}

**if**(datosEdificio.getLavadero() == **null**){

**return** **false**;

}

**if**(datosEdificio.getPatio() == **null**){

**return** **false**;

}

**if**(datosEdificio.getPavimento() == **null**){

**return** **false**;

}

**if**(datosEdificio.getPiscina() == **null**){

**return** **false**;

}

**if**(datosEdificio.getPropiedadHorizontal() == **null**){

**return** **false**;

}

**if**(datosEdificio.getTelefono() == **null**){

**return** **false**;

}

**if**(datosEdificio.getAntiguedad() != **null** && !validador.validarEnteroPositivo(datosEdificio.getAntiguedad())){

**return** **false**;

}

**if**(datosEdificio.getBaños() != **null** && !validador.validarEnteroPositivo(datosEdificio.getBaños())){

**return** **false**;

}

**if**(datosEdificio.getDormitorios() != **null** && !validador.validarEnteroPositivo(datosEdificio.getDormitorios())){

**return** **false**;

}

**if**(datosEdificio.getSuperficie() != **null** && !validador.validarDoublePositivo(datosEdificio.getSuperficie())){

**return** **false**;

}

**return** **true**;

}

Prueba de unidad del archivo GetorInmueble.java

//Casos de prueba

//inmueble, resultado, validadorMock, propietario, excepcion

/\* 0 \*/ **new** Object[] { inmuebleCorrecto, **new** ResultadoCrearInmueble(), validadorCorrecto, propietario, **null** }, //inmueble correcto

/\* 1 \*/ **new** Object[] { inmuebleSinFecha, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Fecha\_Vacia***), validadorCorrecto, propietario, **null** }, //inmueble sin fecha de carga

/\* 2 \*/ **new** Object[] { inmuebleSinPropietario, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Propietario\_Vacio***), validadorCorrecto, propietario, **null** }, //inmueble sin propietario

/\* 3 \*/ **new** Object[] { inmuebleSinTipo, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Tipo\_Vacio***), validadorCorrecto, propietario, **null** }, //inmueble sin TipoInmueble

/\* 4 \*/ **new** Object[] { inmuebleSinPrecio, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Precio\_Vacio***), validadorCorrecto, propietario, **null** }, //inmueble sin precio

/\* 5 \*/ **new** Object[] { inmuebleCorrecto, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Formato\_Direccion\_Incorrecto***), validadorFormatoDireccionIncorrecto, propietario, **null** }, //inmueble con formato de direccion incorrecta

/\* 6 \*/ **new** Object[] { inmueblePrecioIncorrecto, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Precio\_Incorrecto***), validadorDoubleIncorrecto, propietario, **null** }, //inmueble con formato de precio incorrecto

/\* 7 \*/ **new** Object[] { inmuebleFrenteIncorrecto, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Frente\_Incorrecto***), validadorDoubleIncorrecto, propietario, **null** }, //inmueble con formato de frente incorrecto

/\* 8 \*/ **new** Object[] { inmuebleFondoIncorrecto, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Fondo\_Incorrecto***), validadorDoubleIncorrecto, propietario, **null** }, //inmueble con formato de fondo incorrecto

/\* 9 \*/ **new** Object[] { inmuebleSuperficieIncorrecta, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Superficie\_Incorrecta***), validadorDoubleIncorrecto, propietario, **null** }, //inmueble con formato de superficie incorrecto

/\* 10 \*/ **new** Object[] { inmuebleDatosEdificioIncorrectos, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Datos\_Edificio\_Incorrectos***), validadorCorrecto, propietario, **null** }, //inmueble con datosEdificio Incorrectos

/\* 11 \*/ **new** Object[] { inmuebleCorrecto, **new** ResultadoCrearInmueble(ErrorCrearInmueble.***Propietario\_Inexistente***), validadorCorrecto, **null**, **null** }, //propietario del inmueble no está persistido

/\* 12 \*/ **new** Object[] { inmuebleCorrecto, **null**, validadorCorrecto, **null**, **new** ObjNotFoundException("", **new** Exception()) }, //el persistidor tira una PersistenciaException

/\* 13 \*/ **new** Object[] { inmuebleCorrecto, **null**, validadorCorrecto, **null**, **new** Exception() } //el persistidor tira una excepción inesperada

/\*\*

\* Prueba el método crearInmueble(), el cual corresponde con la taskcard 14 de la iteración 1 y a la historia 3

\*

\* **@param** inmueble

\* inmueble a crear

\* **@param** resultado

\* resultado que se espera que devuelva el gestor

\* **@param** validadorMock

\* clase vacía que utiliza el gestor para validar

\* **@param** propietario

\* propietario "en la base de datos" a comparar con el propietario del inmueble a guardar para verificar que exista en la base de datos

\* **@param** excepcion

\* es la excepcion que debe lanzar el mock del persistidor, si la prueba involucra procesar una excepcion de dicho persistidor, debe ser nulo propietario para que se use

\* **@throws** Exception

\*/

**public** **void** testCrearInmueble(Inmueble inmueble, ResultadoCrearInmueble resultado, ValidadorFormato validadorMock, Propietario propietario, Throwable excepcion) **throws** Exception {

GestorDatos gestorDatosMock = **new** GestorDatos() {

@Override

**public** ArrayList<Estado> obtenerEstados() **throws** PersistenciaException {

**return** **new** ArrayList<>();

}

@Override

**public** ArrayList<EstadoInmueble> obtenerEstadosInmueble() **throws** PersistenciaException {

**return** **new** ArrayList<>();

}

};

GestorPropietario gestorPropietarioMock = **new** GestorPropietario() {

@Override

**public** Propietario obtenerPropietario(FiltroPropietario filtro) **throws** PersistenciaException {

**if**(propietario != **null**){

**return** propietario;

}

**if**(excepcion == **null**){

**return** **null**;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

};

InmuebleService persistidorInmuebleMock = **new** InmuebleService() {

@Override

**public** Inmueble obtenerInmueble(Integer id) **throws** PersistenciaException {

**return** **null**;

}

@Override

**public** **void** modificarInmueble(Inmueble inmueble) **throws** PersistenciaException {

}

@Override

**public** ArrayList<Inmueble> listarInmuebles() **throws** PersistenciaException {

**return** **null**;

}

@Override

**public** **void** guardarInmueble(Inmueble inmueble) **throws** PersistenciaException {

}

@Override

**public** ArrayList<Inmueble> listarInmuebles(FiltroInmueble filtro) **throws** PersistenciaException {

**return** **null**;

}

};

GestorInmueble gestorInmueble = **new** GestorInmueble() {

{

**this**.gestorPropietario = gestorPropietarioMock;

**this**.persistidorInmueble = persistidorInmuebleMock;

**this**.gestorDatos = gestorDatosMock;

**this**.validador = validadorMock;

}

};

//creamos la prueba

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

**if**(resultado != **null**){

*assertEquals*(resultado, gestorInmueble.crearInmueble(inmueble));

}

**else**{

**try**{

gestorInmueble.crearInmueble(inmueble);

Assert.*fail*("Debería haber fallado!");

} **catch**(PersistenciaException e){

Assert.*assertEquals*((excepcion), e);

} **catch**(Exception e){

**if**(excepcion **instanceof** PersistenciaException){

Assert.*fail*("Debería haber tirado una PersistenciaException y tiro otra Exception!");

}

}

}

}

};

//Ejecutamos la prueba

**try**{

test.evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

//Casos de prueba

// inmueble, resultadoValidarFondo, resultadoValidarFrente, resultadoValidarSuperficie, resultadoValidarDireccion, resultadoValidarDatosEdificio, resultadoValidarPrecio, retornaInmueble, retornaPropietario, resultado, excepcion

/\* 0 \*/ **new** Object[] { inmuebleCorrecto, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **true**, ***resultadoModificarCorrecto***, **null** }, //inmueble correcto

/\* 1 \*/ **new** Object[] { inmuebleSinFecha, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **true**, ***resultadoModificarFecha\_Vacia***, **null** }, //inmueble sin fecha de carga

/\* 2 \*/ **new** Object[] { inmuebleSinPropietario, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **true**, ***resultadoModificarPropietario\_Vacio***, **null** }, //inmueble sin propietario

/\* 3 \*/ **new** Object[] { inmuebleSinTipo, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **true**, ***resultadoModificarTipo\_Vacio***, **null** }, //inmueble sin TipoInmueble

/\* 4 \*/ **new** Object[] { inmuebleSinPrecio, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **true**, ***resultadoModificarPrecio\_Vacio***, **null** }, //inmueble sin precio

/\* 5 \*/ **new** Object[] { inmuebleFondoIncorrecto, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **true**, ***resultadoModificarFondo\_Incorrecto***, **null** }, //inmueble con formato de fondo incorrecto

/\* 6 \*/ **new** Object[] { inmuebleFrenteIncorrecto, **true**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, ***resultadoModificarFrente\_Incorrecto***, **null** }, //inmueble con formato de frente incorrecto

/\* 7 \*/ **new** Object[] { inmuebleSuperficieIncorrecta, **true**, **true**, **false**, **true**, **true**, **true**, **true**, **true**, ***resultadoModificarSuperficie\_Incorrecta***, **null** }, //inmueble con formato de superficie incorrecto

/\* 8 \*/ **new** Object[] { inmuebleCorrecto, **true**, **true**, **true**, **false**, **true**, **true**, **true**, **true**, ***resultadoModificarFormato\_Direccion\_Incorrecto***, **null** }, //inmueble con formato de direccion incorrecta

/\* 9 \*/ **new** Object[] { inmuebleDatosEdificioIncorrectos, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **true**, ***resultadoModificarDatos\_Edificio\_Incorrectos***, **null** }, //inmueble con datosEdificio Incorrectos

/\* 10 \*/ **new** Object[] { inmuebleCorrecto, **true**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, ***resultadoModificarPrecio\_Incorrecto***, **null** }, //inmueble con formato de precio incorrecto

/\* 11 \*/ **new** Object[] { inmuebleCorrecto, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **false**, ***resultadoModificarPropietario\_Inexistente***, **null** }, //propietario del inmueble no está persistido

/\* 12 \*/ **new** Object[] { inmuebleCorrecto, **true**, **true**, **true**, **true**, **true**, **true**, **false**, **true**, ***resultadoModificarInmueble\_Inexistente***, **null** }, //Inmueble no está persistido

/\* 13 \*/ **new** Object[] { inmuebleCorrecto, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **null**, **new** ObjNotFoundException("", **new** Exception()) }, //el persistidor tira una PersistenciaException

/\* 14 \*/ **new** Object[] { inmuebleCorrecto, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **true**, **null**, **new** SaveUpdateException(**new** Exception()) }, //el persistidor tira una SaveUpdateException

/\*\*

\* Prueba el método modificarInmueble(), el cual corresponde con la taskcard 14 de la iteración 1 y a la historia 3

\*

\* **@param** inmueble

\* inmueble a modificar

\* **@param** resultadoValidarFondo

\* resultado devuelto por el mock validador al validar el fondo del inmueble

\* **@param** resultadoValidarFrente

\* resultado devuelto por el mock validador al validar el frente del inmueble

\* **@param** resultadoValidarSuperficie

\* resultado devuelto por el mock validador al validar la superficie del inmueble

\* **@param** resultadoValidarDireccion

\* resultado devuelto por el mock validador al validar la dirección del inmueble

\* **@param** resultadoValidarDatosEdificioEsperado

\* resultado devuelto por el mock validador al validar el fondo del inmueble

\* **@param** resultadoValidarPrecio

\* resultado devuelto por el mock validador al validar el precio del inmueble

\* **@param** retornaInmueble

\* indica si el persistidor devuelve un inmueble

\* **@param** retornaPropietario

\* indica si el persistidor devuelve un propietario

\* **@param** resultadoEsperado

\* resultado que se espera que devuelva el gestor

\* **@param** propietario

\* propietario "en la base de datos" a comparar con el propietario del inmueble a guardar para verificar que exista en la base de datos

\* **@param** excepcion

\* es la excepcion que debe lanzar el mock del persistidor, si la prueba involucra procesar una excepcion de dicho persistidor, debe ser nulo propietario para que se use

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testModificarInmueble(Inmueble inmueble, Boolean resultadoValidarFondo, Boolean resultadoValidarFrente, Boolean resultadoValidarSuperficie, Boolean resultadoValidarDireccion, Boolean resultadoValidarDatosEdificioEsperado, Boolean resultadoValidarPrecio, Boolean retornaInmueble, Boolean retornaPropietario, ResultadoModificarInmueble resultadoEsperado, Throwable excepcion) **throws** Exception {

GestorPropietario gestorPropietarioMock = Mockito.*mock*(GestorPropietario.**class**);

InmuebleService persistidorInmuebleMock = Mockito.*mock*(InmuebleService.**class**);

HistorialService persistidorHistorialMock = Mockito.*mock*(HistorialService.**class**);

ValidadorFormato validadorFormatoMock = Mockito.*mock*(ValidadorFormato.**class**);

Mockito.*when*(validadorFormatoMock.validarDoublePositivo(inmueble.getFondo())).thenReturn(resultadoValidarFondo);

Mockito.*when*(validadorFormatoMock.validarDoublePositivo(inmueble.getFrente())).thenReturn(resultadoValidarFrente);

Mockito.*when*(validadorFormatoMock.validarDoublePositivo(inmueble.getSuperficie())).thenReturn(resultadoValidarSuperficie);

Mockito.*when*(validadorFormatoMock.validarDireccion(inmueble.getDireccion())).thenReturn(resultadoValidarDireccion);

Mockito.*when*(validadorFormatoMock.validarDoublePositivo(inmueble.getPrecio())).thenReturn(resultadoValidarPrecio);

**if**(retornaPropietario){

Mockito.*when*(gestorPropietarioMock.obtenerPropietario(*any*())).thenReturn(inmueble.getPropietario());

}

**else**{

Mockito.*when*(gestorPropietarioMock.obtenerPropietario(*any*())).thenReturn(**null**);

}

**if**(retornaInmueble){

Mockito.*when*(persistidorInmuebleMock.obtenerInmueble(inmueble.getId())).thenReturn(inmueble);

}

**else**{

Mockito.*when*(persistidorInmuebleMock.obtenerInmueble(inmueble.getId())).thenReturn(**null**);

}

**if**(excepcion != **null**){

Mockito.*doThrow*(excepcion).when(persistidorInmuebleMock).modificarInmueble(inmueble);

}

GestorInmueble gestorInmueble = **new** GestorInmueble() {

{

**this**.validador = validadorFormatoMock;

**this**.gestorPropietario = gestorPropietarioMock;

**this**.persistidorInmueble = persistidorInmuebleMock;

**this**.persistidorHistorial = persistidorHistorialMock;

}

};

//creamos la prueba

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

**if**(excepcion == **null**){

*assertEquals*(resultadoEsperado, gestorInmueble.modificarInmueble(inmueble));

*assertEquals*(resultadoValidarDatosEdificioEsperado, gestorInmueble.validarDatosEdificio(inmueble.getDatosEdificio()));

**if**(!resultadoEsperado.hayErrores()){

Mockito.*verify*(persistidorInmuebleMock).modificarInmueble(*any*());

Mockito.*verify*(persistidorHistorialMock).guardarHistorialInmueble(*any*());

}

}

**else**{

**try**{

gestorInmueble.modificarInmueble(inmueble);

Assert.*fail*("Debería haber fallado!");

} **catch**(PersistenciaException e){

Assert.*assertEquals*((excepcion), e);

} **catch**(Exception e){

**if**(excepcion **instanceof** PersistenciaException){

Assert.*fail*("Debería haber tirado una PersistenciaException y tiro otra Exception!");

}

}

}

}

};

//Ejecutamos la prueba

**try**{

test.evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

### Taskcard 15 Persistidor inmueble

Código del archivo InmuebleServiceImpl.java

/\*

\* Método para guardar en la base de datos un inmueble

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**)

**public** **void** guardarInmueble(Inmueble inmueble) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.save(inmueble);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

/\*

\* Método para modificar en la base de datos un inmueble

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**)

**public** **void** modificarInmueble(Inmueble inmueble) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.update(inmueble);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

/\*

\* Método para listar todos los inmuebles en la base de datos con estado ALTA

\*/

@Override

@Transactional(readOnly = **true**, rollbackFor = PersistenciaException.**class**)

**public** ArrayList<Inmueble> listarInmuebles() **throws** PersistenciaException {

ArrayList<Inmueble> inmuebles = **new** ArrayList<>();

Session session = getSessionFactory().getCurrentSession();

**try**{

//named query ubicada en entidad inmueble

**for**(Object o: session.getNamedQuery("obtenerInmuebles").list()){

**if**(o **instanceof** Inmueble){

inmuebles.add((Inmueble) o);

}

}

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** inmuebles;

}

/\*

\* Método para obtener un inmueble de la base de datos según el id

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**)

**public** Inmueble obtenerInmueble(Integer id) **throws** PersistenciaException {

Inmueble inmueble = **null**;

Session session = getSessionFactory().getCurrentSession();

**try**{

inmueble = session.get(Inmueble.**class**, id);

} **catch**(EntityNotFoundException e){

**throw** **new** ObjNotFoundException("obtener", e);

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** inmueble;

}

### Taskcard 16 Entidad cliente

Código del archivo Cliente.java

@NamedQueries(value = { @NamedQuery(name = "obtenerClientes", query = "SELECT c FROM Cliente c WHERE c.estado.estado = 'ALTA'"), @NamedQuery(name = "obtenerCliente", query = "SELECT c FROM Cliente c WHERE c.numeroDocumento = :documento AND c.tipoDocumento.tipo = :tipoDocumento") })

@Entity

@Table(name = "cliente", uniqueConstraints = @UniqueConstraint(name = "cliente\_numerodocumento\_idtipodocumento\_uk", columnNames = { "numerodocumento", "idtipodocumento" }))

/\*\*

\* Entidad que modela un cliente

\* Pertenece a la taskcard 16 de la iteración 1 y a la historia 6

\*

\* Modificada en TaskCard 27 de la iteración 2

\*

\*/

**public** **class** Cliente {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**protected** Integer id; //ID

@Column(name = "nombre", length = 100, nullable = **true**)

**private** String nombre;

@Column(name = "apellido", length = 100, nullable = **true**)

**private** String apellido;

@Column(name = "numerodocumento", length = 30, nullable = **true**)

**private** String numeroDocumento;

@Column(name = "telefono", length = 30)

**private** String telefono;

//Relaciones

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idtipodocumento", referencedColumnName = "id", foreignKey = @ForeignKey(name = "cliente\_idtipodocumento\_fk"), nullable = **true**)

**private** TipoDocumento tipoDocumento;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idestado", referencedColumnName = "id", foreignKey = @ForeignKey(name = "cliente\_idestado\_fk"), nullable = **false**)

**private** Estado estado;

@OneToOne(mappedBy = "cliente", cascade = CascadeType.***ALL***, orphanRemoval = **true**, optional = **false**)

**private** InmuebleBuscado inmuebleBuscado;

//getters, setters, constructores y otros métodos

(…)

Código del archivo InmuebleBuscado.java

@Entity

@Table(name = "inmueble\_buscado")

/\*

\* Entidad que modela la busqueda de un inmueble, según las características que posee.

\* Pertenece a la taskcard 12 de la iteración 1 y a la historia de usuario 3

\*/

**public** **class** InmuebleBuscado **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** Integer id; //ID

@OneToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idcliente", referencedColumnName = "id", foreignKey = @ForeignKey(name = "inmueble\_buscado\_idcliente\_fk"), nullable = **false**)

**private** Cliente cliente;

@Column(name = "precio\_max")

**private** Double precioMax;

@Column(name = "superficie\_min")

**private** Double superficieMin; // en metros cuadrados

@Column(name = "antiguedad\_max")

**private** Integer antiguedadMax; // en años

@Column(name = "dormitorios\_min")

**private** Integer dormitoriosMin;

@Column(name = "baños\_min")

**private** Integer bañosMin;

@Column(name = "garaje")

**private** Boolean garaje;

@Column(name = "patio")

**private** Boolean patio;

@Column(name = "piscina")

**private** Boolean piscina;

@Column(name = "propiedad\_horizonal")

**private** Boolean propiedadHorizontal;

@Column(name = "agua\_corriente")

**private** Boolean aguaCorriente;

@Column(name = "cloacas")

**private** Boolean cloacas;

@Column(name = "gas\_natural")

**private** Boolean gasNatural;

@Column(name = "agua\_caliente")

**private** Boolean aguaCaliente;

@Column(name = "lavadero")

**private** Boolean lavadero;

@Column(name = "pavimento")

**private** Boolean pavimento;

@Column(name = "telefono")

**private** Boolean telefono;

//Relaciones

@ManyToMany(fetch = FetchType.***LAZY***)

@JoinTable(name = "inmueble\_buscado\_localidad", joinColumns = @JoinColumn(name = "idlocalidad"), foreignKey = @ForeignKey(name = "inmueble\_buscado\_localidad\_idlocalidadfk"), inverseJoinColumns = @JoinColumn(name = "idinmueblebuscado"), inverseForeignKey = @ForeignKey(name = "inmueble\_buscado\_inmueble\_idinmueblefk"))

**private** Set<Localidad> localidades;

@ManyToMany(fetch = FetchType.***LAZY***)

@JoinTable(name = "inmueble\_buscado\_barrio", joinColumns = @JoinColumn(name = "idbarrio"), foreignKey = @ForeignKey(name = "inmueble\_buscado\_barrio\_idbarriofk"), inverseJoinColumns = @JoinColumn(name = "idinmueblebuscado"), inverseForeignKey = @ForeignKey(name = "inmueble\_buscado\_barrio\_idinmueblefk"))

**private** Set<Barrio> barrios;

@ManyToMany(fetch = FetchType.***LAZY***)

@JoinTable(name = "inmueble\_buscado\_tipo\_inmueble", joinColumns = @JoinColumn(name = "idtipoinmueble"), foreignKey = @ForeignKey(name = "inmueble\_buscado\_tipo\_idtipofk"), inverseJoinColumns = @JoinColumn(name = "idinmueblebuscado"), inverseForeignKey = @ForeignKey(name = "inmueble\_buscado\_inmueble\_idinmueblefk"))

**private** Set<TipoInmueble> tiposInmueblesBuscados;

//getters, setters, constructores y otros métodos

(…)

### Taskcard 17 Vista alta, modificar y baja cliente

Código del alta en AltaClienteControllerTest.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga al cliente y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, se muestran al usuario.

\*/

@FXML

**public** **void** acceptAction() {

StringBuilder error = **new** StringBuilder("");

//obtengo datos introducidos por el usuario

String nombre = textFieldNombre.getText().trim();

String apellido = textFieldApellido.getText().trim();

String numeroDocumento = textFieldNumeroDocumento.getText().trim();

String telefono = textFieldTelefono.getText().trim();

String correo = textFieldCorreo.getText().trim();

TipoDocumento tipoDoc = comboBoxTipoDocumento.getValue();

//verifico que no estén vacíos

**if**(nombre.isEmpty()){

error.append("Inserte un nombre").append("\n");

}

**if**(apellido.isEmpty()){

error.append("Inserte un apellido").append("\n");

}

**if**(tipoDoc == **null**){

error.append("Elija un tipo de documento").append("\n");

}

**if**(numeroDocumento.isEmpty()){

error.append("Inserte un numero de documento").append("\n");

}

**if**(telefono.isEmpty()){

error.append("Inserte un telefono").append("\n");

}

**if**(correo.isEmpty()){

error.append("Inserte una dirección de correo electrónico").append("\n");

}

**if**(cliente.getInmuebleBuscado() == **null**){

error.append("Debe cargar un inmueble buscado al cliente").append("\n");

}

**if**(!error.toString().isEmpty()){ //si hay algún error lo muestro al usuario

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

//Si no hay errores se crean las entidades con los datos introducidos

cliente.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setCorreo(correo);

**try**{

//relevo la operación a capa lógica

ResultadoCrearCliente resultado = coordinador.crearCliente(cliente);

**if**(resultado.hayErrores()){

// si hay algún error se muestra al usuario

StringBuilder stringErrores = **new** StringBuilder();

**for**(ErrorCrearCliente err: resultado.getErrores()){

**switch**(err) {

**case** ***Formato\_Nombre\_Incorrecto***:

stringErrores.append("Formato de nombre incorrecto.\n");

**break**;

**case** ***Formato\_Apellido\_Incorrecto***:

stringErrores.append("Formato de apellido incorrecto.\n");

**break**;

**case** ***Formato\_Telefono\_Incorrecto***:

stringErrores.append("Formato de teléfono incorrecto.\n");

**break**;

**case** ***Formato\_Correo\_Incorrecto***:

stringErrores.append("Formato de correo electrónico incorrecto.\n");

**break**;

**case** ***Formato\_Documento\_Incorrecto***:

stringErrores.append("Tipo y formato de documento incorrecto.\n");

**break**;

**case** ***Ya\_Existe\_Cliente***:

stringErrores.append("Ya existe un cliente con ese tipo y número de documento.\n");

**break**;

}

}

presentador.presentarError("Revise sus campos", stringErrores.toString(), stage);

}

**else**{

//si no hay errores se muestra notificación y se vuelve a la pantalla de listar clientes

presentador.presentarToast("Se ha creado el cliente con éxito", stage);

cambiarmeAScene(AdministrarClienteController.***URLVista***);

}

} **catch**(GestionException e){ //excepción originada en gestor

**if**(e.getClass().equals(EntidadExistenteConEstadoBajaException.**class**)){

//el cliente existe pero fué dado de baja

VentanaConfirmacion ventana = presentador.presentarConfirmacion("El cliente ya existe", "El cliente ya existía anteriormente pero fué dado de baja.\n ¿Desea volver a darle de alta?", stage);

**if**(ventana.acepta()){

//usuario acepta volver a darle de alta. Se pasa a la pantalla de modificar cliente

ModificarClienteController controlador = (ModificarClienteController) cambiarmeAScene(ModificarClienteController.***URLVista***);

controlador.setClienteEnModificacion(cliente);

}

}

} **catch**(PersistenciaException e){//excepción originada en la capa de persistencia

presentador.presentarExcepcion(e, stage);

}

}

}

/\*\*

\* Acción que se ejecuta al presionar el botón cargar inmueble.

\* Se pasa a la pantalla de inmueble buscado

\*/

@FXML

**private** **void** cargarInmueble() {

//se guardan en el cliente los datos introducidos por el usuario

cliente.setNombre(textFieldNombre.getText().trim())

.setApellido(textFieldApellido.getText().trim())

.setTipoDocumento(comboBoxTipoDocumento.getValue())

.setNumeroDocumento(textFieldNumeroDocumento.getText().trim())

.setTelefono(textFieldTelefono.getText().trim())

.setCorreo(textFieldCorreo.getText().trim());

//se pasa a la pantalla de cargar inmueble

InmuebleBuscadoController controlador = (InmuebleBuscadoController) cambiarmeAScene(InmuebleBuscadoController.***URLVista***);

controlador.setCliente(cliente);

}

Código del test del alta en AltaClienteControllerTest.java

**protected** Object[] parametersForTestCrearCliente() {

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

InmuebleBuscado inm = **new** InmuebleBuscado(**null**, 1000000.00, 10.0, 10, 1, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

**return** **new** Object[] {

//nombre,apellido,tipoDocumento,numeroDocumento,telefono,correo,inmueble,resultadoCrearClienteEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,llamaACrearCliente,excepcion,aceptarVentanaConfirmacion,llamaACambiarScene

//prueba correcta

/\*0\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 0, 1, **null**, **true**, 0 },

//prueba nombre incorrecto

/\*1\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCrearNombreIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba apellido incorrecto

/\*2\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCrearApellidoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba documento incorrecto

/\*3\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCrearDocumentoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba teléfono incorrecto

/\*4\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCrearTelefonoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba correo incorrecto

/\*5\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCrearCorreoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba ya existe cliente

/\*6\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCrearYaExiste***, 1, 0, 1, **null**, **true**, 0 },

//prueba ya existe cliente

/\*7\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, **new** ResultadoCrearCliente(ErrorCrearCliente.***Formato\_Nombre\_Incorrecto***, ErrorCrearCliente.***Formato\_Apellido\_Incorrecto***), 1, 0, 1, **null**, **true**, 0 },

//prueba nombre vacio

/\*8\*/ **new** Object[] { "", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, **null**, 1, 0, 0, **null**, **true**, 0 },

//prueba cliente Existente y acepta

/\*9\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **true**, 1 },

//prueba cliente Existente y cancela

/\*10\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **false**, 0 },

//prueba PersistenciaException

/\*11\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 1, 1, **new** SaveUpdateException(**new** Throwable()), **false**, 0 }

};

}

/\*\*

\* Test para probar crear un cliente cuando el usuario presiona el botón aceptar

\*

\* **@param** nombre

\* nombre que es introducido por el usuario

\* **@param** apellido

\* appellido que es introducido por el usuario

\* **@param** tipoDocumento

\* tipo de documento que es introducido por el usuario

\* **@param** numeroDocumento

\* número de documento que es introducido por el usuario

\* **@param** telefono

\* teléfono que es introducido por el usuario

\* **@param** correo

\* correo que es introducido por el usuario

\* **@param** inmueble

\* inmueble buscado cargado por el usuario en la pantalla de cargar inmueble buscado

\* **@param** resultadoCrearClienteEsperado

\* resultado que retornará el mock de capa lógica

\* **@param** llamaAPresentadorVentanasPresentarError

\* 1 si llama al método presentar error del presentador de ventanas, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* 1 si llama al método presentar excepción del presentador de ventanas, 0 si no

\* **@param** llamaACrearCliente

\* 1 si llama al método crear cliente de la capa lógica, 0 si no

\* **@param** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@param** aceptarVentanaConfirmacion

\* si el usuario acepta la ventana de confirmación

\* **@param** llamaACambiarScene

\* 1 si llama al método cambiar scene, 0 si no

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCrearCliente(String nombre,

String apellido,

TipoDocumento tipoDocumento,

String numeroDocumento,

String telefono,

String correo,

InmuebleBuscado inmueble,

ResultadoCrearCliente resultadoCrearClienteEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Integer llamaACrearCliente,

Exception excepcion,

Boolean aceptarVentanaConfirmacion,

Integer llamaACambiarScene) **throws** Throwable {

CoordinadorJavaFX coordinadorMock = Mockito.*mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaConfirmacion ventanaConfirmacionMock = *mock*(VentanaConfirmacion.**class**);

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

ModificarClienteController modificarClienteControllerMock = *mock*(ModificarClienteController.**class**);

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarConfirmacion(*any*(), *any*(), *any*())).thenReturn(ventanaConfirmacionMock);

*when*(ventanaConfirmacionMock.acepta()).thenReturn(aceptarVentanaConfirmacion);

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(modificarClienteControllerMock);

*doNothing*().when(modificarClienteControllerMock).setClienteEnModificacion(*any*());

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

*cliente* = **new** Cliente()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setInmuebleBuscado(inmueble)

.setTelefono(telefono)

.setCorreo(correo);

inmueble.setCliente(*cliente*);

*when*(coordinadorMock.crearCliente(*cliente*)).thenReturn(resultadoCrearClienteEsperado);

**if**(excepcion != **null**){

*when*(coordinadorMock.crearCliente(*cliente*)).thenThrow(excepcion);

}

ArrayList<TipoDocumento> tipos = **new** ArrayList<>();

tipos.add(tipoDocumento);

Mockito.*when*(coordinadorMock.obtenerTiposDeDocumento()).thenReturn(tipos);

AltaClienteController altaClienteController = **new** AltaClienteController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

setScenographyChanger(scenographyChangerMock);

**super**.inicializar(location, resources);

}

@Override

**public** **void** acceptAction() {

**this**.textFieldNombre.setText(nombre);

**this**.textFieldApellido.setText(apellido);

**this**.comboBoxTipoDocumento.getSelectionModel().select(tipoDocumento);

**this**.textFieldNumeroDocumento.setText(numeroDocumento);

**this**.textFieldTelefono.setText(telefono);

**this**.textFieldCorreo.setText(correo);

**super**.acceptAction();

}

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** setCliente(Cliente cliente) {

**if**(cliente != **null**){

**this**.cliente = cliente;

}

**else**{

**this**.cliente = **new** Cliente();

}

}

};

altaClienteController.setCliente(*cliente*);

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(AltaClienteController.***URLVista***, altaClienteController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

altaClienteController.acceptAction();

Mockito.*verify*(coordinadorMock).obtenerTiposDeDocumento();

Mockito.*verify*(coordinadorMock, Mockito.*times*(llamaACrearCliente)).crearCliente(Mockito.*any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

Mockito.*verify*(scenographyChangerMock, *times*(llamaACambiarScene)).cambiarScenography(ModificarClienteController.***URLVista***, **false**);

Mockito.*verify*(modificarClienteControllerMock, *times*(llamaACambiarScene)).setClienteEnModificacion(*cliente*);

}

};

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

Código de modificar en ModificarClienteController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga al cliente y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, se muestran al usuario.

\*/

@FXML

**protected** **void** acceptAction() {

StringBuilder error = **new** StringBuilder("");

//obtengo datos introducidos por el usuario

String nombre = textFieldNombre.getText().trim();

String apellido = textFieldApellido.getText().trim();

String numeroDocumento = textFieldNumeroDocumento.getText().trim();

String telefono = textFieldTelefono.getText().trim();

String correo = textFieldCorreo.getText().trim();

TipoDocumento tipoDoc = comboBoxTipoDocumento.getValue();

//verifico que no estén vacíos

**if**(nombre.isEmpty()){

error.append("Inserte un nombre").append("\r\n");

}

**if**(apellido.isEmpty()){

error.append("Inserte un apellido").append("\r\n");

}

**if**(tipoDoc == **null**){

error.append("Elija un tipo de documento").append("\r\n");

}

**if**(numeroDocumento.isEmpty()){

error.append("Inserte un numero de documento").append("\r\n");

}

**if**(telefono.isEmpty()){

error.append("Inserte un telefono").append("\r\n");

}

**if**(correo.isEmpty()){

error.append("Inserte una dirección de correo electrónico").append("\n");

}

**if**(!error.toString().isEmpty()){ //si hay algún error lo muestro al usuario

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

//Si no hay errores se modifican las entidades con los datos introducidos

clienteEnModificacion.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setCorreo(correo);

**try**{

//relevo la operación a capa lógica

ResultadoModificarCliente resultado = coordinador.modificarCliente(clienteEnModificacion);

**if**(resultado.hayErrores()){

// si hay algún error se muestra al usuario

StringBuilder stringErrores = **new** StringBuilder();

**for**(ErrorModificarCliente err: resultado.getErrores()){

**switch**(err) {

**case** ***Formato\_Nombre\_Incorrecto***:

stringErrores.append("Formato de nombre incorrecto.\n");

**break**;

**case** ***Formato\_Apellido\_Incorrecto***:

stringErrores.append("Formato de apellido incorrecto.\n");

**break**;

**case** ***Formato\_Telefono\_Incorrecto***:

stringErrores.append("Formato de teléfono incorrecto.\n");

**break**;

**case** ***Formato\_Correo\_Incorrecto***:

stringErrores.append("Formato de correo electrónico incorrecto.\n");

**break**;

**case** ***Formato\_Documento\_Incorrecto***:

stringErrores.append("Tipo y formato de documento incorrecto.\n");

**break**;

**case** ***Otro\_Cliente\_Posee\_Mismo\_Documento\_Y\_Tipo***:

stringErrores.append("Otro cliente ya posee ese tipo y número de documento.\n");

**break**;

}

}

presentador.presentarError("Revise sus campos", stringErrores.toString(), stage);

}

**else**{

//si no hay errores se muestra notificación y se vuelve a la pantalla de listar clientes

presentador.presentarToast("Se ha modificado el cliente con éxito", stage);

cambiarmeAScene(AdministrarClienteController.***URLVista***);

}

} **catch**(PersistenciaException e){ //excepción originada en la capa de persistencia

presentador.presentarExcepcion(e, stage);

}

}

}

Código del test de modificar en ModificarClienteControllerTest.java

**protected** Object[] parametersForTestModificarCliente() {

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

InmuebleBuscado inm = **new** InmuebleBuscado(**null**, 1000000.00, 10.0, 10, 1, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

**return** **new** Object[] {

//nombre,apellido,tipoDocumento,numeroDocumento,telefono,correo,inmueble,resultadoModificarClienteEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,excepcion

//prueba correcta

/\*0\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 0, **null** },

//prueba nombre incorrecto

/\*1\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarNombreIncorrecto***, 1, 0, **null** },

//prueba apellido incorrecto

/\*2\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarApellidoIncorrecto***, 1, 0, **null** },

//prueba documento incorrecto

/\*3\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarDocumentoIncorrecto***, 1, 0, **null** },

//prueba teléfono incorrecto

/\*4\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarTelefonoIncorrecto***, 1, 0, **null** },

//prueba correo incorrecto

/\*5\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarCorreoIncorrecto***, 1, 0, **null** },

//prueba ya existe cliente

/\*6\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarYaExiste***, 1, 0, **null** },

//prueba ya existe cliente

/\*7\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, **new** ResultadoModificarCliente(ErrorModificarCliente.***Formato\_Nombre\_Incorrecto***, ErrorModificarCliente.***Formato\_Apellido\_Incorrecto***), 1, 0, **null** },

//prueba nombre vacio

/\*8\*/ **new** Object[] { "", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, **null**, 1, 0, **null** },

//prueba PersistenciaException

/\*9\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 1, **new** SaveUpdateException(**new** Throwable()) }

};

}

/\*\*

\* Test para probar crear un cliente cuando el usuario presiona el botón aceptar

\*

\* **@param** nombre

\* nombre que es introducido por el usuario

\* **@param** apellido

\* appellido que es introducido por el usuario

\* **@param** tipoDocumento

\* tipo de documento que es introducido por el usuario

\* **@param** numeroDocumento

\* número de documento que es introducido por el usuario

\* **@param** telefono

\* teléfono que es introducido por el usuario

\* **@param** correo

\* correo que es introducido por el usuario

\* **@param** inmueble

\* inmueble buscado cargado por el usuario en la pantalla de cargar inmueble buscado

\* **@param** resultadoModificarClienteEsperado

\* resultado que retornará el mock de capa lógica

\* **@param** llamaAPresentadorVentanasPresentarError

\* 1 si llama al método presentar error del presentador de ventanas, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* 1 si llama al método presentar excepción del presentador de ventanas, 0 si no

\* **@param** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testModificarCliente(String nombre,

String apellido,

TipoDocumento tipoDocumento,

String numeroDocumento,

String telefono,

String correo,

InmuebleBuscado inmueble,

ResultadoModificarCliente resultadoModificarClienteEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Exception excepcion) **throws** Throwable {

CoordinadorJavaFX coordinadorMock = Mockito.*mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaConfirmacion ventanaConfirmacionMock = *mock*(VentanaConfirmacion.**class**);

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

ModificarClienteController modificarClienteControllerMock = *mock*(ModificarClienteController.**class**);

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarConfirmacion(*any*(), *any*(), *any*())).thenReturn(ventanaConfirmacionMock);

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(modificarClienteControllerMock);

*doNothing*().when(modificarClienteControllerMock).setClienteEnModificacion(*any*());

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

*cliente* = **new** Cliente()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setInmuebleBuscado(inmueble)

.setTelefono(telefono)

.setCorreo(correo);

inmueble.setCliente(*cliente*);

*when*(coordinadorMock.modificarCliente(*cliente*)).thenReturn(resultadoModificarClienteEsperado);

**if**(excepcion != **null**){

*when*(coordinadorMock.modificarCliente(*cliente*)).thenThrow(excepcion);

}

ArrayList<TipoDocumento> tipos = **new** ArrayList<>();

tipos.add(tipoDocumento);

Mockito.*when*(coordinadorMock.obtenerTiposDeDocumento()).thenReturn(tipos);

ModificarClienteController modificarClienteController = **new** ModificarClienteController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

setScenographyChanger(scenographyChangerMock);

**super**.inicializar(location, resources);

}

@Override

**public** **void** acceptAction() {

**this**.textFieldNombre.setText(nombre);

**this**.textFieldApellido.setText(apellido);

**this**.comboBoxTipoDocumento.getSelectionModel().select(tipoDocumento);

**this**.textFieldNumeroDocumento.setText(numeroDocumento);

**this**.textFieldTelefono.setText(telefono);

**this**.textFieldCorreo.setText(correo);

**super**.acceptAction();

}

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** setClienteEnModificacion(Cliente clienteEnModificacion) {

**this**.clienteEnModificacion = clienteEnModificacion;

}

};

modificarClienteController.setClienteEnModificacion(*cliente*);

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(ModificarClienteController.***URLVista***, modificarClienteController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

modificarClienteController.acceptAction();

Mockito.*verify*(coordinadorMock).obtenerTiposDeDocumento();

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

}

};

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

Código de la baja en AdministrarClienteController.java

/\*\*

\* Acción que se ejecuta al presionar el botón eliminar

\* Se muestra una ventana emergente para confirmar la operación

\*/

@FXML

**protected** ResultadoControlador handleEliminar() {

ArrayList<ErrorControlador> erroresControlador = **new** ArrayList<>();

**if**(tablaClientes.getSelectionModel().getSelectedItem() == **null**){

**return** **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***);

}

//solicito confirmación al usuario

VentanaConfirmacion ventana = presentador.presentarConfirmacion("Eliminar cliente", "Está a punto de eliminar al cliente.\n ¿Está seguro que desea hacerlo?", **this**.stage);

**if**(!ventana.acepta()){

**return** **new** ResultadoControlador();//si no acepta

}

**try**{

ResultadoEliminarCliente resultado = coordinador.eliminarCliente(tablaClientes.getSelectionModel().getSelectedItem());

**if**(resultado.hayErrores()){

// si hay errores lo muestro al usuario

StringBuilder stringErrores = **new** StringBuilder();

**for**(ErrorEliminarCliente err: resultado.getErrores()){

**switch**(err) {

}

}

presentador.presentarError("No se pudo eliminar el cliente", stringErrores.toString(), stage);

}

**else**{ //si no hay errores muestro notificación

presentador.presentarToast("Se ha eliminado el cliente con éxito", stage);

}

//actualizo la tabla

tablaClientes.getItems().clear();

tablaClientes.getItems().addAll(coordinador.obtenerClientes());

**return** **new** ResultadoControlador(erroresControlador.toArray(**new** ErrorControlador[0]));

} **catch**(PersistenciaException e){ //falla en la capa de persistencia

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

} **catch**(Exception e){// alguna otra excepción inesperada

presentador.presentarExcepcionInesperada(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

}

Código del test de la baja en AdministrarClienteControllerTest.java

**protected** Object[] parametersForTestEliminarCliente() {

Cliente cliente = **new** Cliente();

Boolean acepta = **true**;

ResultadoControlador resultadoControladorCorrecto = **new** ResultadoControlador();

ResultadoControlador resultadoControladorErrorPersistencia = **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

ResultadoControlador resultadoControladorErrorDesconocido = **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

ResultadoEliminarCliente resultadoLogicaCorrecto = **new** ResultadoEliminarCliente();

Throwable excepcionPersistencia = **new** ObjNotFoundException("", **new** Exception());

Throwable excepcionInesperada = **new** Exception();

**return** **new** Object[] {

//cliente,acepta,resultadoControlador,resultadoLogica,excepcion

/\*0\*/ **new** Object[] { cliente, acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario acepta y el cliente se elimina correctamente

/\*1\*/ **new** Object[] { cliente, !acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario no acepta, pero de haber aceptado, se hubiese eliminado el cliente correctamente

/\*2\*/ **new** Object[] { cliente, acepta, resultadoControladorErrorPersistencia, **null**, excepcionPersistencia }, //test donde el controlador tira una excepción de persistencia

/\*3\*/ **new** Object[] { cliente, acepta, resultadoControladorErrorDesconocido, **null**, excepcionInesperada } //test donde el controlador tira unaexcepción inesperada

};

}

@Test

@Parameters

/\*\*

\* Test para probar la baja de un cliente en el controlador de administrar clientes

\*

\* **@param** cliente

\* cliente a eliminar

\* **@param** acepta

\* si usuario acepta confirmación de eliminar

\* **@param** resultadoControlador

\* resultado que se espera que retorne el método a probar

\* **@param** resultadoLogica

\* resultado que retornará el mock de la capa lógica

\* **@param** excepcion

\* excepción que se simula lanzar desde la capa lógica

\*/

**public** **void** testEliminarCliente(Cliente cliente, Boolean acepta, ResultadoControlador resultadoControlador, ResultadoEliminarCliente resultadoLogica, Throwable excepcion) **throws** Exception {

CoordinadorJavaFX coordinadorMock = **new** CoordinadorJavaFX() {

@Override

**public** ResultadoEliminarCliente eliminarCliente(Cliente cliente) **throws** PersistenciaException {

**if**(resultadoLogica != **null**){

**return** resultadoLogica;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

@Override

**public** ArrayList<Cliente> obtenerClientes() **throws** PersistenciaException {

ArrayList<Cliente> clientes = **new** ArrayList<>();

clientes.add(cliente);

**return** clientes;

}

};

PresentadorVentanas presentadorMock = **new** PresentadorVentanasMock(acepta);

AdministrarClienteController administrarClienteController = **new** AdministrarClienteController() {

@Override

**public** ResultadoControlador handleEliminar() {

tablaClientes.getSelectionModel().select(cliente);

**return** **super**.handleEliminar();

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

administrarClienteController.setCoordinador(coordinadorMock);

administrarClienteController.setPresentador(presentadorMock);

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarClienteController.***URLVista***, administrarClienteController);

administrarClienteController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

*assertEquals*(resultadoControlador, administrarClienteController.handleEliminar());

}

};

**try**{

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

Código de la pantalla de cargar inmueble buscado para alta y modificación de cliente en InmuebleBuscadoController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos y los carga al inmueble buscado.

\* Si el cliente no posee inmueble buscado se crea uno, sino, se modifica el existente.

\*

\* Al finalizar regresa a la pantalla correspondiente, ya sea alta cliente o modificar cliente.

\*/

@FXML

**protected** **void** acceptAction() {

Double supeficieMinima = **null**;

Integer antiguedadMaxima = **null**;

Integer dormitoriosMinimos = **null**;

Integer bañosMinimos = **null**;

Double precioMaximo = **null**;

StringBuffer errores = **new** StringBuffer("");

//obtengo datos introducidos por el usuario

**try**{

supeficieMinima = Double.*valueOf*(textFieldSuperficie.getText().trim());

} **catch**(Exception e){

errores.append("Superficie incorrecta. Introduzca solo números y un punto para decimales.\n");

}

**try**{

precioMaximo = Double.*valueOf*(textFieldPrecio.getText().trim());

} **catch**(Exception e){

errores.append("Precio incorrecto. Introduzca solo números y un punto para decimales.\n");

}

**try**{

antiguedadMaxima = Integer.*valueOf*(textFieldAntiguedad.getText().trim());

} **catch**(Exception e){

errores.append("Antigüedad incorrecta. Introduzca solo números\n");

}

**try**{

dormitoriosMinimos = Integer.*valueOf*(textFieldDormitorios.getText().trim());

} **catch**(Exception e){

errores.append("Dormitorios incorrecto. Introduzca solo números\n");

}

**try**{

bañosMinimos = Integer.*valueOf*(textFieldBaños.getText().trim());

} **catch**(Exception e){

errores.append("Baños incorrecto. Introduzca solo números\n");

}

**if**(!errores.toString().isEmpty()){ //si hay algún error lo muestro al usuario

presentador.presentarError("Revise sus campos", errores.toString(), stage);

}

**else**{

//Si no hay errores se terminan de obtener los datos que no se verifican

Boolean propiedadHorizontal = checkBoxPropiedadHorizontal.isSelected();

Boolean garage = checkBoxGarage.isSelected();

Boolean patio = checkBoxPatio.isSelected();

Boolean piscina = checkBoxPiscina.isSelected();

Boolean aguaCorriente = checkBoxAguaCorriente.isSelected();

Boolean cloaca = checkBoxCloaca.isSelected();

Boolean gasNatural = checkBoxGasNatural.isSelected();

Boolean aguaCaliente = checkBoxAguaCaliente.isSelected();

Boolean telefono = checkBoxTelefono.isSelected();

Boolean lavadero = checkBoxLavadero.isSelected();

Boolean pavimento = checkBoxPavimento.isSelected();

Boolean casa = checkBoxCasa.isSelected();

Boolean departamento = checkBoxDepartamento.isSelected();

Boolean local = checkBoxLocal.isSelected();

Boolean galpon = checkBoxGalpon.isSelected();

Boolean terreno = checkBoxTerreno.isSelected();

Boolean quinta = checkBoxQuinta.isSelected();

InmuebleBuscado inmuebleBuscado = cliente.getInmuebleBuscado();

**if**(alta){ //si se está dando de alta el cliente, se crea un nuevo inmueble

inmuebleBuscado = **new** InmuebleBuscado();

cliente.setInmuebleBuscado(inmuebleBuscado);

inmuebleBuscado.setCliente(cliente);

}

//cargo los datos al inmueble

inmuebleBuscado.setSuperficieMin(supeficieMinima)

.setAntiguedadMax(antiguedadMaxima)

.setDormitoriosMin(dormitoriosMinimos)

.setBañosMin(bañosMinimos)

.setPrecioMax(precioMaximo)

.setPropiedadHorizontal(propiedadHorizontal)

.setGaraje(garage)

.setPatio(patio)

.setPiscina(piscina)

.setAguaCaliente(aguaCaliente)

.setAguaCorriente(aguaCorriente)

.setCloacas(cloaca)

.setGasNatural(gasNatural)

.setTelefono(telefono)

.setLavadero(lavadero)

.setPavimento(pavimento);

inmuebleBuscado.getLocalidades().clear();

inmuebleBuscado.getLocalidades().addAll(listaLocalidadesSeleccionadas);

inmuebleBuscado.getBarrios().clear();

inmuebleBuscado.getBarrios().addAll(listaBarriosSeleccionados);

**try**{

**for**(TipoInmueble tipo: coordinador.obtenerTiposInmueble()){

**switch**(tipo.getTipo()) {

**case** ***CASA***:

**if**(casa){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***DEPARTAMENTO***:

**if**(departamento){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***GALPON***:

**if**(galpon){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***LOCAL***:

**if**(local){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***QUINTA***:

**if**(quinta){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***TERRENO***:

**if**(terreno){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

}

}

} **catch**(PersistenciaException e){ //excepción originada en la capa de persistencia

presentador.presentarExcepcion(e, stage);

}

**if**(alta){ //si se está dando de alta vuelvo a la vista de alta cliente

AltaClienteController controlador = (AltaClienteController) cambiarmeAScene(AltaClienteController.***URLVista***);

controlador.setCliente(cliente);

}

**else**{ //si se está modificando vuelvo a la vista de modificar cliente

ModificarClienteController controlador = (ModificarClienteController) cambiarmeAScene(ModificarClienteController.***URLVista***);

controlador.setClienteEnModificacion(cliente);

}

}

}

Código del test de la pantalla de cargar inmueble buscado para alta y modificación de cliente en InmuebleBuscadoControllerTest.java

**protected** Object[] parametersForTestCargarInmuebleBuscado() {

ArrayList<Localidad> localidades = **new** ArrayList<>();

localidades.add(**new** Localidad("Federal", **new** Provincia("Entre ríos", **new** Pais("Argentina"))));

ArrayList<Barrio> barrios = **new** ArrayList<>();

Cliente clienteEnModificacion = **new** Cliente();

clienteEnModificacion.setInmuebleBuscado(**new** InmuebleBuscado());

clienteEnModificacion.getInmuebleBuscado().setCliente(clienteEnModificacion);

**return** **new** Object[] {

//String superficie,antiguedad,dormitorios,baños,precio,localidades,barrios,local,casa,departamento,terreno,galpon,quinta,propiedadHorizontal,garage,patio,piscina,aguaCorriente,cloaca,gasNatural,aguaCaliente,telefono,lavadero,pavimento,clienteNuevo,clienteEnModificacion,camposCorrectos,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,excepcion

//prueba correcta de inmueble nuevo

/\*0\*/ **new** Object[] { "30.0", "10", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **new** Cliente(), **null**, 1, 0, 0, **null** },

//prueba campo de texto incorrecto de inmueble nuevo

/\*1\*/ **new** Object[] { "30.0", "abc", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **new** Cliente(), **null**, 0, 1, 0, **null** },

//prueba campo texto vacio de inmueble nuevo

/\*2\*/ **new** Object[] { "30.0", "10", "", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **new** Cliente(), **null**, 0, 1, 0, **null** },

//prueba PersistenciaException de inmueble nuevo

/\*3\*/ **new** Object[] { "30.0", "10", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **new** Cliente(), **null**, 1, 0, 1, **new** SaveUpdateException(**new** Throwable()) },

//prueba correcta de inmueble en modificación

/\*4\*/ **new** Object[] { "30.0", "10", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **null**, clienteEnModificacion, 1, 0, 0, **null** },

//prueba campo de texto incorrecto de inmueble en modificación

/\*5\*/ **new** Object[] { "30.0", "abc", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **null**, clienteEnModificacion, 0, 1, 0, **null** },

//prueba campo texto vacio de inmueble en modificación

/\*6\*/ **new** Object[] { "30.0", "10", "", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **null**, clienteEnModificacion, 0, 1, 0, **null** },

//prueba PersistenciaException de inmueble en modificación

/\*7\*/ **new** Object[] { "30.0", "10", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **null**, clienteEnModificacion, 1, 0, 1, **new** SaveUpdateException(**new** Throwable()) }

};

}

/\*\*

\* Test para probar cargar un inmueble buscado cuando se está creando un cliente y cuando se está modificando un cliente

\* al momento en que el usuario presiona el botón de aceptar

\*

\* **@param** superficie

\* introducida por el usuario

\* **@param** antiguedad

\* introducida por el usuario

\* **@param** dormitorios

\* introducidos por el usuario

\* **@param** baños

\* introducidos por el usuario

\* **@param** precio

\* introducido por el usuario

\* **@param** localidades

\* introducidas por el usuario

\* **@param** barrios

\* introducidos por el usuario

\* **@param** local

\* si es seleccionado el checkbox por el usuario

\* **@param** casa

\* si es seleccionado el checkbox por el usuario

\* **@param** departamento

\* si es seleccionado el checkbox por el usuario

\* **@param** terreno

\* si es seleccionado el checkbox por el usuario

\* **@param** galpon

\* si es seleccionado el checkbox por el usuario

\* **@param** quinta

\* si es seleccionado el checkbox por el usuario

\* **@param** propiedadHorizontal

\* si es seleccionado el checkbox por el usuario

\* **@param** garage

\* si es seleccionado el checkbox por el usuario

\* **@param** patio

\* si es seleccionado el checkbox por el usuario

\* **@param** piscina

\* si es seleccionado el checkbox por el usuario

\* **@param** aguaCorriente

\* si es seleccionado el checkbox por el usuario

\* **@param** cloaca

\* si es seleccionado el checkbox por el usuario

\* **@param** gasNatural

\* si es seleccionado el checkbox por el usuario

\* **@param** aguaCaliente

\* si es seleccionado el checkbox por el usuario

\* **@param** telefono

\* si es seleccionado el checkbox por el usuario

\* **@param** lavadero

\* si es seleccionado el checkbox por el usuario

\* **@param** pavimento

\* si es seleccionado el checkbox por el usuario

\* **@param** clienteNuevo

\* cliente que se está creando

\* **@param** clienteEnModificacion

\* cliente que se está modificando

\* **@param** camposCorrectos

\* 1 si los campos son correctos, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarError

\* 1 si llama al método presentar error del presentador de ventanas, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* 1 si llama al método presentar excepción del presentador de ventanas, 0 si no

\* **@param** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCargarInmuebleBuscado(String superficie,

String antiguedad,

String dormitorios,

String baños,

String precio,

ArrayList<Localidad> localidades,

ArrayList<Barrio> barrios,

Boolean local,

Boolean casa,

Boolean departamento,

Boolean terreno,

Boolean galpon,

Boolean quinta,

Boolean propiedadHorizontal,

Boolean garage,

Boolean patio,

Boolean piscina,

Boolean aguaCorriente,

Boolean cloaca,

Boolean gasNatural,

Boolean aguaCaliente,

Boolean telefono,

Boolean lavadero,

Boolean pavimento,

Cliente clienteNuevo,

Cliente clienteEnModificacion,

Integer camposCorrectos,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Exception excepcion) **throws** Throwable {

CoordinadorJavaFX coordinadorMock = Mockito.*mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

AltaClienteController altaClienteControllerMock = *mock*(AltaClienteController.**class**);

ModificarClienteController modificarClienteControllerMock = *mock*(ModificarClienteController.**class**);

*when*(coordinadorMock.obtenerPaises()).thenReturn(**new** ArrayList<Pais>());

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

**if**(clienteNuevo != **null**){

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(altaClienteControllerMock);

}

**if**(clienteEnModificacion != **null**){

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(modificarClienteControllerMock);

}

*doNothing*().when(altaClienteControllerMock).setCliente(*any*());

*doNothing*().when(modificarClienteControllerMock).setClienteEnModificacion(*any*());

**if**(excepcion == **null**){

ArrayList<TipoInmueble> tipos = **new** ArrayList<>();

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***CASA***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***DEPARTAMENTO***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***GALPON***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***LOCAL***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***QUINTA***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***TERRENO***));

*when*(coordinadorMock.obtenerTiposInmueble()).thenReturn(tipos);

}

**else**{

*when*(coordinadorMock.obtenerTiposInmueble()).thenThrow(excepcion);

}

InmuebleBuscadoController inmuebleBuscadoController = **new** InmuebleBuscadoController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

setScenographyChanger(scenographyChangerMock);

**super**.inicializar(location, resources);

}

@Override

**public** **void** acceptAction() {

**this**.textFieldAntiguedad.setText(antiguedad);

**this**.textFieldBaños.setText(baños);

**this**.textFieldDormitorios.setText(dormitorios);

**this**.textFieldPrecio.setText(precio);

**this**.textFieldSuperficie.setText(superficie);

**this**.listaBarriosSeleccionados.clear();

**this**.listaBarriosSeleccionados.addAll(barrios);

**this**.listaLocalidadesSeleccionadas.clear();

**this**.listaLocalidadesSeleccionadas.addAll(localidades);

**this**.checkBoxAguaCaliente.setSelected(aguaCaliente);

**this**.checkBoxAguaCorriente.setSelected(aguaCorriente);

**this**.checkBoxCasa.setSelected(casa);

**this**.checkBoxCloaca.setSelected(cloaca);

**this**.checkBoxDepartamento.setSelected(departamento);

**this**.checkBoxGalpon.setSelected(galpon);

**this**.checkBoxGarage.setSelected(garage);

**this**.checkBoxGasNatural.setSelected(gasNatural);

**this**.checkBoxLavadero.setSelected(lavadero);

**this**.checkBoxLocal.setSelected(local);

**this**.checkBoxPatio.setSelected(patio);

**this**.checkBoxPavimento.setSelected(pavimento);

**this**.checkBoxPiscina.setSelected(piscina);

**this**.checkBoxPropiedadHorizontal.setSelected(propiedadHorizontal);

**this**.checkBoxQuinta.setSelected(quinta);

**this**.checkBoxTelefono.setSelected(telefono);

**this**.checkBoxTerreno.setSelected(terreno);

**super**.acceptAction();

}

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** setCliente(Cliente cliente) {

**this**.cliente = cliente;

InmuebleBuscado inmueble = cliente.getInmuebleBuscado();

**if**(inmueble != **null**){

**this**.alta = **false**;

}

**else**{

**this**.alta = **true**;

}

}

};

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(InmuebleBuscadoController.***URLVista***, inmuebleBuscadoController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

**if**(clienteNuevo != **null**){

inmuebleBuscadoController.setCliente(clienteNuevo);

}

**if**(clienteEnModificacion != **null**){

inmuebleBuscadoController.setCliente(clienteEnModificacion);

}

inmuebleBuscadoController.acceptAction();

**if**(clienteNuevo != **null** && excepcion == **null**){

Mockito.*verify*(altaClienteControllerMock, *times*(camposCorrectos)).setCliente(clienteNuevo);

}

**if**(clienteEnModificacion != **null** && excepcion == **null**){

Mockito.*verify*(modificarClienteControllerMock, *times*(camposCorrectos)).setClienteEnModificacion(clienteEnModificacion);

}

Mockito.*verify*(coordinadorMock, *times*(camposCorrectos)).obtenerTiposInmueble();

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

}

};

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

### Taskcard 18 Lógica alta, modificar y baja cliente

Código del archivo en GestorCliente.java

/\*\*

\* Se encarga de validar los datos de un cliente a crear y, en caso de que no haya errores,

\* delegar el guardado del objeto a la capa de acceso a datos.

\*

\* Modificada en TaskCard 27 de la iteración 2

\*

\* **@param** cliente

\* cliente a crear

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\* **@throws** GestionException

\* se lanza una excepción EntidadExistenteConEstadoBaja cuando se encuentra que ya existe un vendedor con la misma identificación pero tiene estado BAJA

\*/

**public** ResultadoCrearCliente crearCliente(Cliente cliente) **throws** PersistenciaException, GestionException {

ArrayList<ErrorCrearCliente> errores = **new** ArrayList<>();

// valida formato de datos

**if**(!validador.validarNombre(cliente.getNombre())){

errores.add(ErrorCrearCliente.***Formato\_Nombre\_Incorrecto***);

}

**if**(!validador.validarApellido(cliente.getApellido())){

errores.add(ErrorCrearCliente.***Formato\_Apellido\_Incorrecto***);

}

**if**(!validador.validarTelefono(cliente.getTelefono())){

errores.add(ErrorCrearCliente.***Formato\_Telefono\_Incorrecto***);

}

**if**(!validador.validarDocumento(cliente.getTipoDocumento(), cliente.getNumeroDocumento())){

errores.add(ErrorCrearCliente.***Formato\_Documento\_Incorrecto***);

}

**if**(!validador.validarEmail(cliente.getCorreo())){

errores.add(ErrorCrearCliente.***Formato\_Correo\_Incorrecto***);

}

//valida si existe un cliente con ese tipo y número de documento

Cliente clienteAuxiliar = persistidorCliente.obtenerCliente(**new** FiltroCliente(cliente.getTipoDocumento().getTipo(),

cliente.getNumeroDocumento()));

**if**(**null** != clienteAuxiliar){

**if**(clienteAuxiliar.getEstado().getEstado().equals(EstadoStr.***ALTA***)){

errores.add(ErrorCrearCliente.***Ya\_Existe\_Cliente***);// si existe y tiene estado alta

}

**else**{// si existe y tiene estado baja

**throw** **new** EntidadExistenteConEstadoBajaException();

}

}

**if**(errores.isEmpty()){//si no hay errores

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***ALTA***)){

cliente.setEstado(e);//seteo el estado en alta

}

}

persistidorCliente.guardarCliente(cliente);

}

**return** **new** ResultadoCrearCliente(errores.toArray(**new** ErrorCrearCliente[0]));

}

/\*\*

\* Se encarga de validar los datos de un cliente a modificar y, en caso de que no haya errores,

\* delegar el guardado del objeto a la capa de acceso a datos.

\*

\* Modificada en TaskCard 27 de la iteración 2

\*

\* **@param** cliente

\* cliente a modificar

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ResultadoModificarCliente modificarCliente(Cliente cliente) **throws** PersistenciaException {

ArrayList<ErrorModificarCliente> errores = **new** ArrayList<>();

// valida formato de datos

**if**(!validador.validarNombre(cliente.getNombre())){

errores.add(ErrorModificarCliente.***Formato\_Nombre\_Incorrecto***);

}

**if**(!validador.validarApellido(cliente.getApellido())){

errores.add(ErrorModificarCliente.***Formato\_Apellido\_Incorrecto***);

}

**if**(!validador.validarTelefono(cliente.getTelefono())){

errores.add(ErrorModificarCliente.***Formato\_Telefono\_Incorrecto***);

}

**if**(!validador.validarDocumento(cliente.getTipoDocumento(), cliente.getNumeroDocumento())){

errores.add(ErrorModificarCliente.***Formato\_Documento\_Incorrecto***);

}

**if**(!validador.validarEmail(cliente.getCorreo())){

errores.add(ErrorModificarCliente.***Formato\_Correo\_Incorrecto***);

}

//verifica si existe otro cliente con los nuevos tipo y número de documento

Cliente clienteAuxiliar = persistidorCliente.obtenerCliente(**new** FiltroCliente(cliente.getTipoDocumento().getTipo(),

cliente.getNumeroDocumento()));

**if**(clienteAuxiliar != **null** && !cliente.equals(clienteAuxiliar)){

errores.add(ErrorModificarCliente.***Otro\_Cliente\_Posee\_Mismo\_Documento\_Y\_Tipo***);

}

**if**(errores.isEmpty()){//si no hay errores

**if**(cliente.getEstado().getEstado().equals(EstadoStr.***BAJA***)){

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***ALTA***)){

cliente.setEstado(e); //si el estado es baja, se setea en alta

}

}

}

persistidorCliente.modificarCliente(cliente);

}

**return** **new** ResultadoModificarCliente(errores.toArray(**new** ErrorModificarCliente[0]));

}

/\*\*

\* Se encarga de validar que exista el cliente a eliminar, se setea el estado en BAJA y,

\* en caso de que no haya errores, delegar el guardado del objeto a la capa de acceso a datos.

\*

\* **@param** cliente

\* cliente a eliminar

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ResultadoEliminarCliente eliminarCliente(Cliente cliente) **throws** PersistenciaException {

//se setea el estado en baja y se manda a guardar

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

**for**(Estado e: estados){

**if**(e.getEstado().equals(EstadoStr.***BAJA***)){

cliente.setEstado(e);

}

}

persistidorCliente.modificarCliente(cliente);

**return** **new** ResultadoEliminarCliente();

}

Código de los test en GestorCliente.java

**protected** Object[] parametersForTestCrearCliente() {

//Se carga cliente con datos básicos solo para evitar punteros nulos

//Las validaciones se hacen en el validador, por lo que se setean luego en los mocks los valores esperados

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

*cliente* = **new** Cliente();

InmuebleBuscado inm = **new** InmuebleBuscado(*cliente*, 4567.2, 345.9, 10, 2, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

*cliente*.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setTelefono("1234-1234")

.setCorreo("asdf@asf.com")

.setInmuebleBuscado(inm);

//Parámetros de JUnitParams

**return** **new** Object[] {

//resValNombre,resValApellido,resValDocumento,resValTelefono,resValCorreo,resObtenerCliente,guardar,resultadoCrearClienteEsperado

/\*0\*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, **null**, 1, ***resultadoCorrecto*** },

/\*1\*/ **new** Object[] { **false**, **true**, **true**, **true**, **true**, **null**, 0, ***resultadoCrearNombreIncorrecto*** },

/\*2\*/ **new** Object[] { **true**, **false**, **true**, **true**, **true**, **null**, 0, ***resultadoCrearApellidoIncorrecto*** },

/\*3\*/ **new** Object[] { **true**, **true**, **false**, **true**, **true**, **null**, 0, ***resultadoCrearDocumentoIncorrecto*** },

/\*4\*/ **new** Object[] { **true**, **true**, **true**, **false**, **true**, **null**, 0, ***resultadoCrearTelefonoIncorrecto*** },

/\*5\*/ **new** Object[] { **true**, **true**, **true**, **true**, **false**, **null**, 0, ***resultadoCrearCorreoIncorrecto*** },

/\*6\*/ **new** Object[] { **false**, **false**, **true**, **true**, **true**, **null**, 0, **new** ResultadoCrearCliente(ErrorCrearCliente.***Formato\_Nombre\_Incorrecto***, ErrorCrearCliente.***Formato\_Apellido\_Incorrecto***) },

/\*7\*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, *cliente*, 0, ***resultadoCrearYaExiste*** }

};

}

/\*\*

\* Test para probar el método crearCliente

\*

\* **@param** resValNombre

\* resultado devuelto por el mock validador de formato al validar nombre

\* **@param** resValApellido

\* resultado devuelto por el mock validador de formato al validar apellido

\* **@param** resValDocumento

\* resultado devuelto por el mock validador de formato al validar documento

\* **@param** resValTelefono

\* resultado devuelto por el mock validador de formato al validar teléfono

\* **@param** resValCorreo

\* resultado devuelto por el mock validador de formato al validar correo

\* **@param** resObtenerCliente

\* resultado devuelto por el mock persistidor al obtener cliente

\* **@param** guardar

\* 1 si se espera que se ejecute el guardar hacia capa de persistencia, 0 si no

\* **@param** resultadoCrearClienteEsperado

\* resultado que se espera que retorne el método a probar

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testCrearCliente(Boolean resValNombre, Boolean resValApellido, Boolean resValDocumento, Boolean resValTelefono, Boolean resValCorreo, Cliente resObtenerCliente, Integer guardar, ResultadoCrearCliente resultadoCrearClienteEsperado) **throws** Exception {

//Inicialización de los mocks

ClienteService clienteServiceMock = *mock*(ClienteService.**class**);

ValidadorFormato validadorFormatoMock = *mock*(ValidadorFormato.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

GestorCliente gestorCliente = **new** GestorCliente() {

{

**this**.persistidorCliente = clienteServiceMock;

**this**.validador = validadorFormatoMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***ALTA***));

//Setear valores esperados a los mocks

*when*(validadorFormatoMock.validarNombre(*cliente*.getNombre())).thenReturn(resValNombre);

*when*(validadorFormatoMock.validarApellido(*cliente*.getApellido())).thenReturn(resValApellido);

*when*(validadorFormatoMock.validarDocumento(*cliente*.getTipoDocumento(), *cliente*.getNumeroDocumento())).thenReturn(resValDocumento);

*when*(validadorFormatoMock.validarTelefono(*cliente*.getTelefono())).thenReturn(resValTelefono);

*when*(validadorFormatoMock.validarEmail(*cliente*.getCorreo())).thenReturn(resValCorreo);

*when*(clienteServiceMock.obtenerCliente(*any*())).thenReturn(resObtenerCliente);

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

*doNothing*().when(clienteServiceMock).guardarCliente(*cliente*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoCrearCliente resultadoCrearCliente = gestorCliente.crearCliente(*cliente*);

//Comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

*assertEquals*(resultadoCrearClienteEsperado, resultadoCrearCliente);

**if**(guardar.equals(1)){

*assertEquals*(EstadoStr.***ALTA***, *cliente*.getEstado().getEstado());

}

*verify*(validadorFormatoMock).validarNombre(*cliente*.getNombre());

*verify*(validadorFormatoMock).validarApellido(*cliente*.getApellido());

*verify*(validadorFormatoMock).validarDocumento(*cliente*.getTipoDocumento(), *cliente*.getNumeroDocumento());

*verify*(validadorFormatoMock).validarTelefono(*cliente*.getTelefono());

*verify*(validadorFormatoMock).validarEmail(*cliente*.getCorreo());

*verify*(gestorDatosMock, *times*(guardar)).obtenerEstados();

*verify*(clienteServiceMock, *times*(guardar)).guardarCliente(*cliente*);

}

**protected** Object[] parametersForTestModificarCliente() {

//Se carga cliente con datos básicos solo para evitar punteros nulos

//Las validaciones se hacen en el validador, por lo que se setean luego en los mocks los valores esperados

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

Estado est = **new** Estado(EstadoStr.***ALTA***);

*clienteM* = **new** Cliente() {

**public** Cliente setId(Integer id) {

**this**.id = id;

**return** **this**;

}

{

**this**.setId(1)

.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setEstado(est);

}

};

InmuebleBuscado inm = **new** InmuebleBuscado(*clienteM*, 4567.2, 345.9, 10, 2, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

*clienteM*.setInmuebleBuscado(inm);

*clienteM2* = **new** Cliente() {

**public** Cliente setId(Integer id) {

**this**.id = id;

**return** **this**;

}

{

**this**.setId(2)

.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc);

}

};

InmuebleBuscado inm2 = **new** InmuebleBuscado(*clienteM2*, 4567.2, 345.9, 10, 2, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

*clienteM2*.setInmuebleBuscado(inm2);

//Parámetros de JUnitParams

**return** **new** Object[] {

//resValNombre,resValApellido,resValDocumento,resValTelefono,resValCorreo,resObtenerCliente,modificar,resultadoModificarClienteEsperado

/\*0\*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, *clienteM*, 1, ***resultadoCorrectoModificar*** },

/\*1\*/ **new** Object[] { **false**, **true**, **true**, **true**, **true**, *clienteM*, 0, ***resultadoModificarNombreIncorrecto*** },

/\*2\*/ **new** Object[] { **true**, **false**, **true**, **true**, **true**, *clienteM*, 0, ***resultadoModificarApellidoIncorrecto*** },

/\*3\*/ **new** Object[] { **true**, **true**, **false**, **true**, **true**, *clienteM*, 0, ***resultadoModificarDocumentoIncorrecto*** },

/\*4\*/ **new** Object[] { **true**, **true**, **true**, **false**, **true**, *clienteM*, 0, ***resultadoModificarTelefonoIncorrecto*** },

/\*5\*/ **new** Object[] { **true**, **true**, **true**, **true**, **false**, *clienteM*, 0, ***resultadoModificarCorreoIncorrecto*** },

/\*6\*/ **new** Object[] { **false**, **false**, **true**, **true**, **true**, *clienteM*, 0, **new** ResultadoModificarCliente(ErrorModificarCliente.***Formato\_Nombre\_Incorrecto***, ErrorModificarCliente.***Formato\_Apellido\_Incorrecto***) },

/\*7\*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, *clienteM2*, 0, ***resultadoModificarYaSePoseeMismoDocumento*** }

};

}

/\*\*

\* Test para probar el método modificarCliente

\*

\* **@param** resValNombre

\* resultado devuelto por el mock validador de formato al validar nombre

\* **@param** resValApellido

\* resultado devuelto por el mock validador de formato al validar apellido

\* **@param** resValDocumento

\* resultado devuelto por el mock validador de formato al validar documento

\* **@param** resValTelefono

\* resultado devuelto por el mock validador de formato al validar teléfono

\* **@param** resValCorreo

\* resultado devuelto por el mock validador de formato al validar correo

\* **@param** resObtenerCliente

\* resultado devuelto por el mock persistidor al obtener cliente

\* **@param** modificar

\* 1 si se espera que se ejecute el modificar hacia capa de persistencia, 0 si no

\* **@param** resultadoModificarClienteEsperado

\* resultado que se espera que retorne el método a probar

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testModificarCliente(Boolean resValNombre, Boolean resValApellido, Boolean resValDocumento, Boolean resValTelefono, Boolean resValCorreo, Cliente resObtenerCliente, Integer modificar, ResultadoModificarCliente resultadoModificarClienteEsperado) **throws** Exception {

//Inicialización de los mocks

ClienteService clienteServiceMock = *mock*(ClienteService.**class**);

ValidadorFormato validadorFormatoMock = *mock*(ValidadorFormato.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

//Clase anónima necesaria para inyectar dependencias hasta que funcione Spring

GestorCliente gestorCliente = **new** GestorCliente() {

{

**this**.persistidorCliente = clienteServiceMock;

**this**.validador = validadorFormatoMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***ALTA***));

//Setear valores esperados a los mocks

*when*(validadorFormatoMock.validarNombre(*clienteM*.getNombre())).thenReturn(resValNombre);

*when*(validadorFormatoMock.validarApellido(*clienteM*.getApellido())).thenReturn(resValApellido);

*when*(validadorFormatoMock.validarDocumento(*clienteM*.getTipoDocumento(), *cliente*.getNumeroDocumento())).thenReturn(resValDocumento);

*when*(validadorFormatoMock.validarTelefono(*clienteM*.getTelefono())).thenReturn(resValTelefono);

*when*(validadorFormatoMock.validarEmail(*clienteM*.getCorreo())).thenReturn(resValCorreo);

*when*(clienteServiceMock.obtenerCliente(*any*())).thenReturn(resObtenerCliente);

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

*doNothing*().when(clienteServiceMock).modificarCliente(*clienteM*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoModificarCliente resultadoModificarCliente = gestorCliente.modificarCliente(*clienteM*);

//Comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

*assertEquals*(resultadoModificarClienteEsperado, resultadoModificarCliente);

**if**(modificar.equals(1)){

*assertEquals*(EstadoStr.***ALTA***, *clienteM*.getEstado().getEstado());

}

*verify*(validadorFormatoMock).validarNombre(*clienteM*.getNombre());

*verify*(validadorFormatoMock).validarApellido(*clienteM*.getApellido());

*verify*(validadorFormatoMock).validarDocumento(*clienteM*.getTipoDocumento(), *clienteM*.getNumeroDocumento());

*verify*(validadorFormatoMock).validarTelefono(*clienteM*.getTelefono());

*verify*(validadorFormatoMock).validarEmail(*clienteM*.getCorreo());

*verify*(clienteServiceMock, *times*(modificar)).modificarCliente(*clienteM*);

}

**protected** Object[] parametersForTestEliminarCliente() {

//Se carga cliente con datos básicos solo para evitar punteros nulos

//Las validaciones se hacen en el validador, por lo que se setean luego en los mocks los valores esperados

TipoDocumento doc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

Estado est = **new** Estado(EstadoStr.***ALTA***);

*clienteE* = **new** Cliente();

InmuebleBuscado inm = **new** InmuebleBuscado(*cliente*, 4567.2, 345.9, 10, 2, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

*clienteE*.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setEstado(est)

.setInmuebleBuscado(inm);

//Parámetros de JUnitParams

**return** **new** Object[] {

//resObtenerCliente,eliminar,resultadoEliminarClienteEsperado

/\*0\*/ **new** Object[] { *clienteE*, 1, ***resultadoCorrectoEliminar*** },

};

}

/\*\*

\* Test para probar el método eliminarCliente

\*

\* **@param** resObtenerCliente

\* resultado devuelto por el mock persistidor al obtener cliente

\* **@param** eliminar

\* 1 si se espera que se ejecute el modificar hacia capa de persistencia (ya que es baja lógica), 0 si no

\* **@param** resultadoEliminarClienteEsperado

\* resultado que se espera retorne el método a probar

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testEliminarCliente(Cliente resObtenerCliente, Integer eliminar, ResultadoEliminarCliente resultadoEliminarClienteEsperado) **throws** Exception {

//Inicialización de los mocks

ClienteService clienteServiceMock = *mock*(ClienteService.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

//Clase anónima necesaria para inyectar dependencias hasta que funcione Spring

GestorCliente gestorCliente = **new** GestorCliente() {

{

**this**.persistidorCliente = clienteServiceMock;

**this**.gestorDatos = gestorDatosMock;

}

};

ArrayList<Estado> estados = **new** ArrayList<>();

estados.add(**new** Estado(EstadoStr.***BAJA***));

//Setear valores esperados a los mocks

*when*(clienteServiceMock.obtenerCliente(*any*())).thenReturn(resObtenerCliente);

*when*(gestorDatosMock.obtenerEstados()).thenReturn(estados);

*doNothing*().when(clienteServiceMock).modificarCliente(*clienteE*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoEliminarCliente resultadoEliminarCliente = gestorCliente.eliminarCliente(*clienteE*);

//Comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

*assertEquals*(resultadoEliminarClienteEsperado, resultadoEliminarCliente);

**if**(eliminar.equals(1)){

*assertEquals*(EstadoStr.***BAJA***, *clienteE*.getEstado().getEstado());

}

*verify*(gestorDatosMock, *times*(eliminar)).obtenerEstados();

*verify*(clienteServiceMock, *times*(eliminar)).modificarCliente(*clienteE*);

}

### Taskcard 19 Persistidor cliente

Código del archivo ClienteService.java

/\*\*

\* Interface que define los métodos de persistencia de un cliente

\* Pertenece a la taskcard 19 de la iteración 1 y a la historia 6

\*/

**public** **interface** ClienteService {

**public** **void** guardarCliente(Cliente cliente) **throws** PersistenciaException;

**public** **void** modificarCliente(Cliente cliente) **throws** PersistenciaException;

**public** Cliente obtenerCliente(FiltroCliente filtro) **throws** PersistenciaException;

**public** ArrayList<Cliente> listarClientes() **throws** PersistenciaException;

}

Código del archivo AdministrarInmuebleController.java

@Override

@Transactional(rollbackFor = PersistenciaException.**class**)

**public** **void** guardarCliente(Cliente cliente) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.save(cliente);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

@Override

@Transactional(rollbackFor = PersistenciaException.**class**)

**public** **void** modificarCliente(Cliente cliente) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.update(cliente);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

@Override

@Transactional(readOnly = **true**, rollbackFor = PersistenciaException.**class**)

**public** Cliente obtenerCliente(FiltroCliente filtro) **throws** PersistenciaException {

Cliente cliente;

Session session = getSessionFactory().getCurrentSession();

**try**{//named query ubicada en entidad Cliente

cliente = (Cliente) session.getNamedQuery("obtenerCliente").setParameter("tipoDocumento", filtro.getTipoDocumento()).setParameter("documento", filtro.getDocumento()).uniqueResult();

} **catch**(NoResultException e){

**return** **null**;

} **catch**(NonUniqueResultException e){

**return** **null**;

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** cliente;

}

@Override

@Transactional(readOnly = **true**, rollbackFor = PersistenciaException.**class**)

**public** ArrayList<Cliente> listarClientes() **throws** PersistenciaException {

ArrayList<Cliente> clientes = **new** ArrayList<>();

Session session = getSessionFactory().getCurrentSession();

**try**{//named query ubicada en entidad Cliente

**for**(Object o: session.getNamedQuery("obtenerClientes").list()){

**if**(o **instanceof** Cliente){

clientes.add((Cliente) o);

}

}

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** clientes;

}

## Iteración 2:

### Taskcard 20 Vista de consulta inmueble

Código en AdministrarInmuebleController.java

/\*\*

\* Método que permite realizar una consulta de inmuebles

\* Pertenece a la taskcard 20 de la iteración 2 y a la historia 4

\*/

@FXML

**private** **void** buscarAction() {

StringBuilder errores = **new** StringBuilder("");

//Se obtienen los datos ingresados por el usuario y se verifican errores

Pais pais = comboBoxPais.getValue();

Provincia provincia = comboBoxProvincia.getValue();

Localidad localidad = comboBoxLocalidad.getValue();

Barrio barrio = comboBoxBarrio.getValue();

EstadoInmueble estadoInmueble = comboBoxEstadoInmueble.getValue();

TipoInmueble tipoInmueble = comboBoxTipoInmueble.getValue();

Integer cantidadDormitorios = **null**;

Double precioMinimo = **null**;

Double precioMaximo = **null**;

**boolean** vacioCD = **false**, vacioPMa = **false**, vacioPMi = **false**;

**if**(!textFieldCantidadDormitorios.getText().trim().isEmpty()){

**try**{

cantidadDormitorios = Integer.*valueOf*(textFieldCantidadDormitorios.getText().trim());

} **catch**(Exception e){

errores.append("Cantidad de dormitorios incorrecta. Introduzca solo números.\n");

}

}

**else**{

vacioCD = **true**;

}

**if**(!textFieldPrecioMinimo.getText().trim().isEmpty()){

**try**{

precioMinimo = Double.*valueOf*(textFieldPrecioMinimo.getText().trim());

} **catch**(Exception e){

errores.append("Precio mínimo incorrecto. Introduzca solo números y un punto para decimales.\n");

}

}

**else**{

vacioPMa = **true**;

}

**if**(!textFieldPrecioMaximo.getText().trim().isEmpty()){

**try**{

precioMaximo = Double.*valueOf*(textFieldPrecioMaximo.getText().trim());

} **catch**(Exception e){

errores.append("Precio máximo incorrecto. Introduzca solo números y un punto para decimales.\n");

}

}

**else**{

vacioPMi = **true**;

}

**if**(precioMaximo != **null** && precioMinimo != **null** && precioMaximo < precioMinimo){

errores.append("El precio máximo es menor al precio mínimo.\n");

}

//si hay errores se muestra al usuario

**if**(!errores.toString().isEmpty()){

presentador.presentarError("Revise sus campos", errores.toString(), stage);

}

**else**{

tablaInmuebles.getItems().clear();

// si no se introdujo ningún filtro de búsqueda

**if**(pais == **null** && provincia == **null** && localidad == **null** && barrio == **null** && estadoInmueble == **null** && tipoInmueble == **null** && vacioCD && vacioPMa && vacioPMi){

**try**{

tablaInmuebles.getItems().addAll(coordinador.obtenerInmuebles());

} **catch**(PersistenciaException e){ //fallo en la capa de persistencia

presentador.presentarExcepcion(e, stage);

}

}

**else**{ // si se introdujo algún filtro de búsqueda

**try**{

FiltroInmueble filtro = **new** FiltroInmueble.Builder()

.barrio(barrio)

.cantidadDormitorios(cantidadDormitorios)

.estadoInmueble(((estadoInmueble != **null**) ? estadoInmueble.getEstado() : (**null**)))

.localidad(localidad)

.pais(pais)

.precioMaximo(precioMaximo)

.precioMinimo(precioMinimo)

.provincia(provincia)

.tipoInmueble(((tipoInmueble != **null**) ? tipoInmueble.getTipo() : (**null**)))

.build();

tablaInmuebles.getItems().addAll(coordinador.obtenerInmuebles(filtro));

} **catch**(PersistenciaException e) { // fallo en la capa de persistencia

presentador.presentarExcepcion(e, stage);

} **catch**(Exception e){ //alguna otra excepción inesperada

presentador.presentarExcepcionInesperada(e, stage);

}

}

tablaInmuebles.getItems().removeAll(inmueblesNoMostrar);

}

}

### Taskcard 21 Lógica y persistidor de consulta inmuebles

Código del archivo FiltroInmueble.java

/\*\*

\* Genera la consulta hql para hibernate con los parámetros con los que lo construyeron

\*/

**private** **void** setConsulta() {

consulta = **this**.getSelect() + **this**.getFrom() + **this**.getWhere() + **this**.getOrderBy();

}

/\*\*

\* Genera el select de la consulta hql para hibernate con los parámetros con los que lo construyeron

\*/

**private** String getSelect() {

String select = "SELECT " + nombreEntidad;

**return** select;

}

/\*\*

\* Genera el from de la consulta hql para hibernate con los parámetros con los que lo construyeron

\*/

**private** String getFrom() {

String from = " FROM Inmueble " + nombreEntidad;

**return** from;

}

/\*\*

\* Genera el where de la consulta hql para hibernate con los parámetros con los que lo construyeron

\*/

**private** String getWhere() {

String where =

((pais != **null**) ? (nombreEntidad + ".direccion.localidad.provincia.pais.nombre LIKE :pai AND ") : ("")) +

((provincia != **null**) ? (nombreEntidad + ".direccion.localidad.provincia.nombre LIKE :pro AND ") : ("")) +

((localidad != **null**) ? (nombreEntidad + ".direccion.localidad.nombre LIKE :loc AND ") : ("")) +

((barrio != **null**) ? (nombreEntidad + ".direccion.barrio.nombre LIKE :bar AND ") : ("")) +

((tipoInmueble != **null**) ? (nombreEntidad + ".tipo.tipo = :tii AND ") : ("")) +

((cantidadDormitorios != **null**) ? (nombreEntidad + ".datosEdificio.dormitorios >= :cad AND ") : ("")) +

((precioMaximo != **null**) ? (nombreEntidad + ".precio <= :pma AND ") : ("")) +

((precioMinimo != **null**) ? (nombreEntidad + ".precio >= :pmi AND ") : ("")) +

((estadoInmueble != **null**) ? (nombreEntidad + ".estadoInmueble.estado = :esi AND ") : (""));

where = " WHERE " + where + nombreEntidad + ".estado.estado = 'ALTA'";

**return** where;

}

/\*\*

\* Genera el order by de la consulta hql para hibernate con los parámetros con los que lo construyeron

\*/

**private** String getOrderBy() {

String orderBy = " ORDER BY " + nombreEntidad + ".fechaCarga ASC";

**return** orderBy;

}

/\*\*

\* Setea los parámetros con los que lo construyeron a la consulta hql de hibernate

\*/

**public** Query setParametros(Query query) {

**if**(pais != **null**){

query.setParameter("pai", "%" + pais + "%");

}

**if**(provincia != **null**){

query.setParameter("pro", "%" + provincia + "%");

}

**if**(localidad != **null**){

query.setParameter("loc", "%" + localidad + "%");

}

**if**(barrio != **null**){

query.setParameter("bar", "%" + barrio + "%");

}

**if**(tipoInmueble != **null**){

query.setParameter("tii", tipoInmueble);

}

**if**(cantidadDormitorios != **null**){

query.setParameter("cad", cantidadDormitorios);

}

**if**(precioMaximo != **null**){

query.setParameter("pma", precioMaximo);

}

**if**(precioMinimo != **null**){

query.setParameter("pmi", precioMinimo);

}

**if**(estadoInmueble != **null**){

query.setParameter("esi", estadoInmueble);

}

**return** query;

}

/\*\*

\* Devuelve la consulta hql de hibernate generada al construirse

\*/

**public** String getConsultaDinamica() {

**return** consulta;

}

Código del archivo InmuebleServiceImpl.java

@Override

@Transactional(readOnly = **true**, rollbackFor = PersistenciaException.**class**)

**public** ArrayList<Inmueble> listarInmuebles(FiltroInmueble filtro) **throws** PersistenciaException {

ArrayList<Inmueble> inmuebles = **new** ArrayList<>();

Session session = getSessionFactory().getCurrentSession();

**try**{

Query query = session.createQuery(filtro.getConsultaDinamica());

filtro.setParametros(query);

**for**(Object o: query.list()){

**if**(o **instanceof** Inmueble){

inmuebles.add((Inmueble) o);

}

}

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** inmuebles;

}

Código del archivo GestorInmueble.java

/\*\*

\* Obtiene el listado de inmuebles solicitándola a la capa de acceso a datos con el criterio del filtro pasado

\*

\* Pertenece a la TaskCard 21 de la iteración 2 y a la historia 4

\*

\* **@param** filtro

\* Criterio de búsqueda de los inmuebles

\*

\* **@return** el listado de inmuebles solicitados

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ArrayList<Inmueble> obtenerInmuebles(FiltroInmueble filtro) **throws** PersistenciaException {

**return** persistidorInmueble.listarInmuebles(filtro);

}

Prueba de unidad del archivo InmuebleServiceImplTest.java

//Casos de prueba

//filtro

/\* 0 \*/**new** Object[] { filtroCompleto },

/\* 1 \*/**new** Object[] { filtroSinEstadoInmueble },

/\* 2 \*/**new** Object[] { filtroVacio }

/\*\*

\* Prueba las querys que genera el filtro que se le pasa a listarInmuebles(FiltroInmueble filtro) para verificar que sean válidas.

\* Corresponde con la taskcard 21 de la iteración 2 y a la historia 4

\*

\* **@param** filtro

\* filtro a probar sus querys

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testListarInmuebles(FiltroInmueble filtro) **throws** Exception {

System.***out***.println("Inicio test");

**try**{

***persistidorInmuble***.listarInmuebles(filtro);

} **catch**(Exception e){

e.printStackTrace();

System.***out***.println("Fin test");

Assert.*fail*("No debería haber fallado");

}

System.***out***.println("Fin test");

}

### Taskcard 22 Vista alta catálogo de inmueble

Código del archivo AltaCatalogoController.java

/\*\*

\* Método que se ejecuta al presionar el botón para agregar inmuebles

\*/

@FXML

**public** **void** agregarInmueble() {

**final** ArrayList<Inmueble> inmueblesNuevos = **new** ArrayList<>();

//Se va a la vista de consulta inmueble y se agregan a la vista al volver

AdministrarInmuebleController vistaInmuebles = (AdministrarInmuebleController) **this**.cambiarScene(fondo, AdministrarInmuebleController.***URLVista***, (Pane) fondo.getChildren().get(0));

vistaInmuebles.formatearObtenerInmueblesNoVendidos(inmuebles, inmueblesNuevos, () -> {

agregarInmuebles(inmueblesNuevos);

}, **true**);

}

/\*\*

\* Método que agrega inmuebles a la vista

\*

\* **@param** inmueblesNuevos

\* inmuebles a agregar a la vista

\*/

**private** **void** agregarInmuebles(ArrayList<Inmueble> inmueblesNuevos) {

**try**{

**for**(Inmueble inmueble: inmueblesNuevos){

//Se agrega un renglón a la lista de inmuebles

RenglonInmuebleController renglonController = **new** RenglonInmuebleController(inmueble);

//Se setea lo que va a hacer el botón eliminarInmueble del renglón

renglonController.setEliminarInmueble(() -> {

//Se quita al inmueble de la vista

listaInmuebles.getChildren().remove(renglonController.getRoot());

renglones.remove(renglonController.getRoot());

inmuebles.remove(inmueble);

});

//Se agrega el inmueble a la vista

listaInmuebles.getChildren().add(renglonController.getRoot());

renglones.put(renglonController.getRoot(), renglonController);

inmuebles.add(inmueble);

}

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e);

}

}

/\*\*

\* Acción que se ejecuta al apretar el botón generar catalogo.

\*

\* Toma datos de la vista, los carga al catálogo y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, se muestran al usuario.

\*

\* **@return** ResultadoControlador que resume lo que hizo el controlador

\*/

@FXML

**public** ResultadoControlador generarCatalogo() {

//Inicialización de variables

ResultadoCrearCatalogo resultado;

StringBuffer erroresBfr = **new** StringBuffer();

CatalogoVista catalogo = **null**;

//Toma de datos de la vista

Map<Inmueble, Imagen> fotos = **new** HashMap<>();

**for**(Node n: listaInmuebles.getChildren()){

**if**(renglones.get(n) != **null**){

RenglonInmuebleController renglon = renglones.get(n);

fotos.put(renglon.getInmueble(), renglon.getFotoSeleccionada());

}

}

//Se cargan los datos de la vista al catálogo a crear

catalogo = **new** CatalogoVista(cbCliente.getValue(), fotos);

//Inicio transacciones al gestor

**try**{

//Se llama a la lógica para crear el catálogo y se recibe el resultado de las validaciones y datos extras de ser necesarios

resultado = coordinador.crearCatalogo(catalogo);

} **catch**(PersistenciaException | GestionException e){

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

//Procesamiento de errores de la lógica

**if**(resultado.hayErrores()){

**for**(ErrorCrearCatalogo e: resultado.getErrores()){

**switch**(e) {

**case** ***Barrio\_Inmueble\_Inexistente***:

erroresBfr.append("El barrio del inmueble está vacío.\n");

**break**;

**case** ***Cliente\_inexistente***:

erroresBfr.append("No se seleccionó un cliente.\n");

**break**;

**case** ***Codigo\_Inmueble\_Inexistente***:

erroresBfr.append("El código del inmueble está vacío.\n");

**break**;

**case** ***Direccion\_Inmueble\_Inexistente***:

erroresBfr.append("La dirección del inmueble está vacío.\n");

**break**;

**case** ***Localidad\_Inmueble\_Inexistente***:

erroresBfr.append("La localidad del inmueble está vacío.\n");

**break**;

**case** ***Precio\_Inmueble\_Inexistente***:

erroresBfr.append("El precio del inmueble está vacío.\n");

**break**;

**case** ***Tipo\_Inmueble\_Inexistente***:

erroresBfr.append("El tipo del inmueble está vacío.\n");

**break**;

}

}

String errores = erroresBfr.toString();

**if**(!errores.isEmpty()){

//Se muestran los errores

presentador.presentarError("Error al crear el catálogo", errores, stage);

}

//Se retorna error

**return** **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***);

}

**else**{

//Se muestra una notificación de que se creó correctamente el catálogo

presentador.presentarToast("Se ha creado el catálogo con éxito", stage);

//Se muestra el catálogo

mostrarPDF(resultado.getCatalogoPDF());

//Se retorna que no hubo errores

**return** **new** ResultadoControlador();

}

}

### Taskcard 23 Lógica alta catálogo de inmueble y generar PDF

Código del archivo GestorCatalogo.java

/\*\*

\* Se encarga de validar los datos de un catalogoVista a crear y, en caso de que no haya errores,

\* delegar la generación del archivo pdf al gestorPDF

\* \*

\* **@param** catalogoVista

\* clase con los datos necesarios para generar un catálogo

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** GestionException

\* se lanza una excepción GenerarPDFException si ocurre un error al generar el PDF

\*/

**public** ResultadoCrearCatalogo crearCatalogo(CatalogoVista catalogoVista) **throws** GestionException {

ArrayList<ErrorCrearCatalogo> errores = **new** ArrayList<>();

PDF catalogoPDF = **null**;

**if**(catalogoVista.getCliente() == **null**){

errores.add(ErrorCrearCatalogo.***Cliente\_inexistente***);

}

catalogoVista.getFotos().forEach((i, f) -> {

**if**(i.getId() == **null**){

errores.add(ErrorCrearCatalogo.***Codigo\_Inmueble\_Inexistente***);

}

**if**(i.getTipo() == **null**){

errores.add(ErrorCrearCatalogo.***Tipo\_Inmueble\_Inexistente***);

}

**if**(i.getDireccion() == **null**){

errores.add(ErrorCrearCatalogo.***Direccion\_Inmueble\_Inexistente***);

}

**if**(i.getDireccion() != **null** && i.getDireccion().getLocalidad() == **null**){

errores.add(ErrorCrearCatalogo.***Localidad\_Inmueble\_Inexistente***);

}

**if**(i.getDireccion() != **null** && i.getDireccion().getBarrio() == **null**){

errores.add(ErrorCrearCatalogo.***Barrio\_Inmueble\_Inexistente***);

}

**if**(i.getPrecio() == **null**){

errores.add(ErrorCrearCatalogo.***Precio\_Inmueble\_Inexistente***);

}

});

**if**(errores.isEmpty()){

catalogoPDF = gestorPDF.generarPDF(catalogoVista);

}

**return** **new** ResultadoCrearCatalogo(catalogoPDF, errores.toArray(**new** ErrorCrearCatalogo[0]));

}

Prueba de unidad del archivo GestorCatalogo.java

//Casos de prueba

//cliente, idSet, tipoInmueble, direccion, localidad, barrio, precio, resultadoCrearCatalogoEsperado, excepcion, excepcionEsperada, llamaACrearPDF

/\* 0 \*/ **new** Object[] { **new** Cliente(), 1, **new** TipoInmueble(TipoInmuebleStr.***CASA***), **new** Direccion(), **new** Localidad(), **new** Barrio(), 1.0, ***resultadoCrearCorrecto***, **null**, **null**, 1 }, //Test correcto

/\* 1 \*/ **new** Object[] { **null**, 1, **new** TipoInmueble(TipoInmuebleStr.***CASA***), **new** Direccion(), **new** Localidad(), **new** Barrio(), 1.0, ***resultadoCrearCliente\_inexistente***, **null**, **null**, 0 }, //Cliente vacío

/\* 2 \*/ **new** Object[] { **new** Cliente(), **null**, **new** TipoInmueble(TipoInmuebleStr.***CASA***), **new** Direccion(), **new** Localidad(), **new** Barrio(), 1.0, ***resultadoCrearCodigo\_Inmueble\_Inexistente***, **null**, **null**, 0 }, //Código de inmueble vacío

/\* 3 \*/ **new** Object[] { **new** Cliente(), 1, **null**, **new** Direccion(), **new** Localidad(), **new** Barrio(), 1.0, ***resultadoCrearTipo\_Inmueble\_Inexistente***, **null**, **null**, 0 }, //Tipo de inmueble vacío

/\* 4 \*/ **new** Object[] { **new** Cliente(), 1, **new** TipoInmueble(TipoInmuebleStr.***CASA***), **null**, **new** Localidad(), **new** Barrio(), 1.0, ***resultadoCrearDireccion\_Inmueble\_Inexistente***, **null**, **null**, 0 }, //Dirección de inmueble vacía

/\* 5 \*/ **new** Object[] { **new** Cliente(), 1, **new** TipoInmueble(TipoInmuebleStr.***CASA***), **new** Direccion(), **null**, **new** Barrio(), 1.0, ***resultadoCrearLocalidad\_Inmueble\_Inexistente***, **null**, **null**, 0 }, //Localidad de inmueble vacía

/\* 6 \*/ **new** Object[] { **new** Cliente(), 1, **new** TipoInmueble(TipoInmuebleStr.***CASA***), **new** Direccion(), **new** Localidad(), **null**, 1.0, ***resultadoCrearBarrio\_Inmueble\_Inexistente***, **null**, **null**, 0 }, //Barrio de inmueble vacío

/\* 7 \*/ **new** Object[] { **new** Cliente(), 1, **new** TipoInmueble(TipoInmuebleStr.***CASA***), **new** Direccion(), **new** Localidad(), **new** Barrio(), **null**, ***resultadoCrearPrecio\_Inmueble\_Inexistente***, **null**, **null**, 0 }, //Barrio de inmueble vacío

/\* 8 \*/ **new** Object[] { **new** Cliente(), 1, **new** TipoInmueble(TipoInmuebleStr.***CASA***), **new** Direccion(), **new** Localidad(), **new** Barrio(), 1.0, **null**, **new** GenerarPDFException(**new** Throwable()), **new** GenerarPDFException(**new** Throwable()), 1 } //El gestorPDF tira una excepción

/\*\*

\* Prueba el método crearCatalogo(), el cual corresponde con la taskcard 23 de la iteración 2

\*

\* **@param** cliente

\* cliente al que generarle el catálogo

\* **@param** idSet

\* código del inmueble para el catálogo

\* **@param** tipoInmueble

\* tipo del inmueble para el catálogo

\* **@param** direccion

\* dirección del inmueble para el catálogo

\* **@param** localidad

\* localidad del inmueble para el catálogo

\* **@param** barrio

\* barrio del inmueble para el catálogo

\* **@param** precio

\* precio del inmueble para el catálogo

\* **@param** resultadoCrearCatalogoEsperado

\* resultado esperado del test

\* **@param** excepcion

\* excepcion arrojada por el gestorPDF

\* **@param** excepcionEsperada

\* excepción esperada que sea lanzada por el método

\* **@param** llamaACrearPDF

\* indica si se debe llamar a crear el pdf

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testCrearCatalogo(Cliente cliente, Integer idSet, TipoInmueble tipoInmueble, Direccion direccion, Localidad localidad, Barrio barrio, Double precio, ResultadoCrearCatalogo resultadoCrearCatalogoEsperado, GestionException excepcion, GestionException excepcionEsperada, Integer llamaACrearPDF) **throws** Exception {

//Inicialización de los mocks y el catálogo que se usará para probar

GestorPDF gestorPDFMock = *mock*(GestorPDF.**class**);

**if**(direccion != **null**){

direccion.setLocalidad(localidad).setBarrio(barrio);

}

Inmueble inmueble = **new** Inmueble() {

@Override

**public** Integer getId() {

**return** idSet;

}

};

inmueble.setTipo(tipoInmueble)

.setDireccion(direccion)

.setPrecio(precio);

HashMap<Inmueble, Imagen> inmuebles = **new** HashMap<>();

inmuebles.put(inmueble, **null**);

CatalogoVista catalogoVista = **new** CatalogoVista(cliente, inmuebles);

//Clase anónima necesaria para inyectar dependencias

GestorCatalogo gestorCatalogo = **new** GestorCatalogo() {

{

**this**.gestorPDF = gestorPDFMock;

}

};

//Setear valores esperados a los mocks

**if**(excepcion != **null**){

*when*(gestorPDFMock.generarPDF(*any*(CatalogoVista.**class**))).thenThrow(excepcion);

}

**else**{

*when*(gestorPDFMock.generarPDF(*any*(CatalogoVista.**class**))).thenReturn(**new** PDF());

}

//Llamar al método a testear y comprobar resultados obtenidos, que se llaman a los métodos deseados y con los parámetros correctos

**if**(excepcionEsperada == **null**){

*assertEquals*(resultadoCrearCatalogoEsperado.getErrores(), gestorCatalogo.crearCatalogo(catalogoVista).getErrores());

}

**else**{

**try**{

gestorCatalogo.crearCatalogo(catalogoVista);

Assert.*fail*("Debería haber fallado!");

} **catch**(Exception e){

*assertEquals*(excepcionEsperada.getClass(), e.getClass());

}

}

*verify*(gestorPDFMock, *times*(llamaACrearPDF)).generarPDF(catalogoVista);

}

### Taskcard 24 Vista alta, baja y listar reserva

Código del archivo AdministrarReservaController.java

/\*\*

\* Acción que se ejecuta al presionar el botón nuevo

\* Se pasa a la pantalla alta reserva

\*/

**public** **void** nuevoAction(ActionEvent event) {

//Se verifica que se haya seleccionado un inmueble y que este no esté vendido

**if**(inmueble != **null** && inmueble.getEstadoInmueble().getEstado().equals(EstadoInmuebleStr.***VENDIDO***)){

presentador.presentarError("Error al crear reserva", "El inmueble ya fue vendido", stage);

**return**;

}

//Se llama a la pantalla alta reserva

AltaReservaController controlador = (AltaReservaController) cambiarmeAScene(AltaReservaController.***URLVista***, ***URLVista***);

//Se le pasan a la pantalla los datos del cliente o inmueble según corresponda

**if**(cliente != **null**){

controlador.setCliente(cliente);

controlador.setVendedorLogueado(vendedorLogueado);

}

**else** **if**(inmueble != **null**){

controlador.setInmueble(inmueble);

controlador.setVendedorLogueado(vendedorLogueado);

}

}

/\*\*

\* Acción que se ejecuta al presionar el botón ver

\* Se pasa a la pantalla de ver pdf

\*/

**public** **void** verAction(ActionEvent event) {

Reserva reserva = tablaReservas.getSelectionModel().getSelectedItem();

//Se comprueba que se haya seleccionado una reserva

**if**(reserva == **null**){

**return**;

}

//Se llama a la pantalla que muestra los PDFs

VerPDFController visorPDF = (VerPDFController) cambiarScene(fondo, VerPDFController.***URLVista***, (Pane) fondo.getChildren().get(0));

visorPDF.cargarPDF(reserva.getArchivoPDF());

}

/\*\*

\* Acción que se ejecuta al presionar el botón eliminar

\* Se muestra una ventana emergente para confirmar la operación

\* Si acepta se da de baja la reserva

\*/

**public** **void** eliminarAction(ActionEvent event) {

Reserva reserva = tablaReservas.getSelectionModel().getSelectedItem();

//Se comprueba que se haya seleccionado una reserva

**if**(reserva == **null**){

**return**;

}

//Se muestra una ventana para que el usuario confirme la operación

VentanaConfirmacion ventana = presentador.presentarConfirmacion("Eliminar reserva", "Está a punto de eliminar una reserva.\n¿Desea continuar?", **this**.stage);

**if**(ventana.acepta()){

ResultadoEliminarReserva resultado = **new** ResultadoEliminarReserva();

**try**{

//Se da de baja la reserva

resultado = coordinador.eliminarReserva(reserva);

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

}

//Se muestran los errores devueltos por el método de la lógica

**if**(resultado.hayErrores()){

StringBuilder stringErrores = **new** StringBuilder();

**for**(ErrorEliminarReserva err: resultado.getErrores()){

**switch**(err) {

//Por el momento no hay errores

}

}

presentador.presentarError("Ha ocurrido un error", stringErrores.toString(), stage);

}

**else**{

//Si no hay errores se muestra una notificación y se remueve la reserva de la lista

tablaReservas.getItems().remove(reserva);

presentador.presentarToast("Se ha eliminado la reserva con éxito", stage);

}

}

}

Prueba de unidad del archivo AdministrarReservaControllerTest.java

/\*\*

\* Se prueba que se llame a la pantalla altaReserva si se presiona el botón alta

\*

\* **@throws** Throwable

\*/

@Test

**public** **void** testAltaReserva() **throws** Throwable {

//Se crean los mocks necesarios

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

AltaReservaController altaReservaControllerMock = *mock*(AltaReservaController.**class**);

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

//Se setea lo que debe devolver el mock cuando es invocado por la clase a probar

*when*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(altaReservaControllerMock);

//Controlador a probar;

AdministrarReservaController administrarReservaController = **new** AdministrarReservaController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.setScenographyChanger(scenographyChangerMock);

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarReservaController.***URLVista***, administrarReservaController);

administrarReservaController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Método a probar

administrarReservaController.nuevoAction(**null**);

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(scenographyChangerMock, *times*(1)).cambiarScenography(AltaReservaController.***URLVista***, **false**);

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

;

}

//Casos de prueba

// listaReservas, llamaAPresentadorVentanasPresentarConfirmacion, aceptarVentanaConfirmacion, resultadoEliminarReservaEsperado, llamaAPresentadorVentanasPresentarError, llamaAPresentadorVentanasPresentarExcepcion, excepcion

/\* 0 \*/ **new** Object[] { listaReservas, 1, **true**, resultadoSinError, 0, 0, **null** }, //Reserva correcta y se selecciona eliminar

/\* 1 \*/ **new** Object[] { listaReservas, 1, **true**, resultadoSinError, 0, 1, exception }, //La base de datos devuelve una excepción

/\* 2 \*/ **new** Object[] { listaReservas, 1, **false**, resultadoSinError, 0, 0, **null** }, //Reserva correcta y no se acepta la ventana de confirmación

/\* 3 \*/ **new** Object[] { listaReservasVacia, 0, **false**, resultadoSinError, 0, 0, **null** } //La lista de reservas es vacía

/\*\*

\* Se prueba que se elimine una reserva si se presiona el botón eliminar

\*

\* **@param** listaReservas

\* lista de las reservas en la tabla

\* **@param** llamaAPresentadorVentanasPresentarConfirmacion

\* indica si se debe llamar a presentar el cuadro de confirmación

\* **@param** aceptarVentanaConfirmacion

\* indica si el usuario acepta o no la confirmación

\* **@param** resultadoEliminarReservaEsperado

\* resultado esperado por el test

\* **@param** llamaAPresentadorVentanasPresentarError

\* indica si se debe llamar a mostrar una ventana de error

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* indica si se debe llamar a mostrar una ventana de excepción

\* **@param** excepcion

\* excepción devuelta por la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testEliminarReserva(ArrayList<Reserva> listaReservas,

Integer llamaAPresentadorVentanasPresentarConfirmacion,

Boolean aceptarVentanaConfirmacion,

ResultadoEliminarReserva resultadoEliminarReservaEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

PersistenciaException excepcion)

**throws** Throwable {

//Se crean los mocks necesarios

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaConfirmacion ventanaConfirmacionMock = *mock*(VentanaConfirmacion.**class**);

//Se setea lo que deben devolver los mocks cuando son invocados por la clase a probar

*when*(coordinadorMock.obtenerReservas(*any*(Cliente.**class**))).thenReturn(listaReservas);

*when*(coordinadorMock.obtenerReservas(*any*(Inmueble.**class**))).thenReturn(listaReservas);

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarConfirmacion(*any*(), *any*(), *any*())).thenReturn(ventanaConfirmacionMock);

*when*(ventanaConfirmacionMock.acepta()).thenReturn(aceptarVentanaConfirmacion);

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

*when*(coordinadorMock.eliminarReserva(*any*(Reserva.**class**))).thenReturn(resultadoEliminarReservaEsperado);

**if**(excepcion != **null**){

*when*(coordinadorMock.eliminarReserva(*any*(Reserva.**class**))).thenThrow(excepcion);

}

//Controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos en la tabla

AdministrarReservaController administrarReservaController = **new** AdministrarReservaController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

**this**.setCliente(**new** Cliente());

**this**.tablaReservas.getSelectionModel().select(0); //se selecciona la primer reserva de la lista

}

@Override

**protected** **void** setTitulo(String titulo) {

}

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarReservaController.***URLVista***, administrarReservaController);

administrarReservaController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

Integer cantidadReservasAntesDeEliminar = administrarReservaController.tablaReservas.getItems().size();

//Método a probar

administrarReservaController.eliminarAction(**null**);

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarConfirmacion)).presentarConfirmacion(*any*(), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

**if**(llamaAPresentadorVentanasPresentarExcepcion == 0){

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*any*(), *any*(), *any*());

}

Integer cantidadReservasDespuesDeEliminar = administrarReservaController.tablaReservas.getItems().size();

**if**(llamaAPresentadorVentanasPresentarError != 1 && llamaAPresentadorVentanasPresentarExcepcion != 1 && aceptarVentanaConfirmacion){

*assertNotEquals*(cantidadReservasAntesDeEliminar, cantidadReservasDespuesDeEliminar);

}

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

Código del archivo AltaReservaController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga a una reserva y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, éstos se muestran al usuario.

\*/

**public** **void** acceptAction() {

StringBuilder error = **new** StringBuilder("");

String importe = textFieldImporte.getText().trim();

Cliente cliente = comboBoxCliente.getValue();

Inmueble inmueble = comboBoxInmueble.getValue();

//Se validan los campos ingresados por el usuario

**if**(cliente == **null**){

error.append("Seleccione un cliente").append("\r\n");

}

**if**(inmueble == **null**){

error.append("Seleccione un inmueble").append("\r\n");

}

**if**(importe.equals("")){

error.append("Ingrese un importe").append("\r\n");

}

**if**(datePickerInicio.getValue() == **null**){

error.append("Ingrese una fecha de inicio").append("\r\n");

}

**if**(datePickerFin.getValue() == **null**){

error.append("Ingrese una fecha de fin").append("\r\n");

}

//Si hay algún error se muestra una ventana informando de ello

**if**(!error.toString().isEmpty()){

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

Date fechaInicio = Date.*from*(datePickerInicio.getValue().atStartOfDay(ZoneId.*systemDefault*()).toInstant());

Date fechaFin = Date.*from*(datePickerFin.getValue().atStartOfDay(ZoneId.*systemDefault*()).toInstant());

//Se crea la reserva para pasársela a la capa lógica

Reserva reserva = **new** Reserva()

.setImporte(Double.*valueOf*(importe))

.setFechaFin(fechaFin)

.setFechaInicio(fechaInicio)

.setCliente(cliente)

.setInmueble(inmueble);

ResultadoCrearReserva resultadoCrearReserva = **null**;

**try**{

//Se llama a la capa lógica para que de de alta la reserva

resultadoCrearReserva = coordinador.crearReserva(reserva);

error.delete(0, error.length());

//Se obtienen los errores y se genera un mensaje para mostrarle al usuario con ellos

**if**(resultadoCrearReserva.hayErrores()){

**for**(ErrorCrearReserva e: resultadoCrearReserva.getErrores()){

**switch**(e) {

**case** ***Cliente\_Vacío***:

error.append("No se ha seleccionado un cliente\r\n");

**break**;

**case** ***Nombre\_Cliente\_Vacío***:

error.append("El cliente seleccionado no tiene nombre\r\n");

**break**;

**case** ***Apellido\_Cliente\_Vacío***:

error.append("El cliente seleccionado no tiene un apellido\r\n");

**break**;

**case** ***NúmeroDocumento\_Cliente\_Vacío***:

error.append("El cliente seleccionado no tiene número de documento\r\n");

**break**;

**case** ***TipoDocumento\_Cliente\_Vacío***:

error.append("El cliente seleccionado no tiene un tipo de documento\r\n");

**break**;

**case** ***Inmueble\_Vacío***:

error.append("No se ha seleccionado un inmueble\r\n");

**break**;

**case** ***Dirección\_Inmueble\_Vacía***:

error.append("El inmueble seleccionado no tiene una dirección\r\n");

**break**;

**case** ***Barrio\_Inmueble\_Vacío***:

error.append("El inmueble seleccionado no tiene un barrio\r\n");

**break**;

**case** ***Calle\_Inmueble\_Vacía***:

error.append("El inmueble seleccionado no tiene una calle\r\n");

**break**;

**case** ***Localidad\_Inmueble\_Vacía***:

error.append("El inmueble seleccionado no tiene una localidad\r\n");

**break**;

**case** ***Altura\_Inmueble\_Vacía***:

error.append("El inmueble seleccionado no tiene una altura en su dirección\r\n");

**break**;

**case** ***Tipo\_Inmueble\_Vacío***:

error.append("El inmueble seleccionado no tiene un tipo de inmueble\r\n");

**break**;

**case** ***Propietario\_Vacío***:

error.append("El inmueble seleccionado no tiene propietario\r\n");

**break**;

**case** ***Nombre\_Propietario\_Vacío***:

error.append("El propietario del inmueble ingresado no tiene nombre\r\n");

**break**;

**case** ***Apellido\_Propietario\_Vacío***:

error.append("El propietario del inmueble seleccionado no tiene apellido\r\n");

**break**;

**case** ***Importe\_Vacío***:

error.append("No se ha ingresado un importe\r\n");

**break**;

**case** ***Importe\_Menor\_O\_Igual\_A\_Cero***:

error.append("El formato del importe es incorrecto\r\n");

**break**;

**case** ***FechaFin\_vacía***:

error.append("No se ha seleccionado una fecha de fin\r\n");

**break**;

**case** ***FechaInicio\_vacía***:

error.append("No se ha seleccionado una fecha de inicio\r\n");

**break**;

**case** ***Fecha\_Inicio\_Posterior\_A\_Fecha\_Fin***:

error.append("La fecha de inicio de la reserva no puede ser posterior a la fecha de vencimiento");

**break**;

**case** ***Existe\_Otra\_Reserva\_Activa***:

error.append("Ya existe una reserva para este inmueble en ese periodo de tiempo\r\n");

**break**;

**case** ***Inmueble\_Vendido***:

error.append("El inmueble ya fue vendido\r\n");

**break**;

}

}

//En caso de haber errores, se le muestran al usuario

presentador.presentarError("Revise sus campos", error.toString(), stage);

}

**else**{

//Si todo es correcto se muestra una notificación y se muestra el pdf con la reserva

presentador.presentarToast("Se ha realizado la reserva con éxito", stage);

mostrarPDF(resultadoCrearReserva.getPdfReserva());

}

} **catch**(PersistenciaException e){

//Si la capa lógica lanza una excepción se presenta una ventana de excepción indicando el error

presentador.presentarExcepcion(e, stage);

} **catch**(Exception e){

//Si ocurre una excepción inesperada, se informa de ello

presentador.presentarExcepcionInesperada(e, stage);

}

}

}

Prueba de unidad del archivo AltaReservaControllerTest.java

//Casos de prueba

// inmuebleAReservar, clienteSeleccionado, importe, fechaInicio, fechaFin, resultadoCrearReservaEsperado, llamaAPresentadorVentanasPresentarError, llamaAPresentadorVentanasPresentarExcepcion, llamaAPresentadorVentanasPresentarExcepcionInesperada, llamaACrearReserva, excepcion, reservaExitosa

/\* 0 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCorrecto***, 0, 0, 0, 1, **null**, 1 }, //prueba correcta

/\* 1 \*/ **new** Object[] { inmueble, **null**, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCorrecto***, 1, 0, 0, 0, **null**, 0 }, //cliente no seleccionado

/\* 2 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearCliente\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //cliente null

/\* 3 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearNombre\_Cliente\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //cliente sin nombre

/\* 4 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearApellido\_Cliente\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //cliente sin apellido

/\* 5 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearTipoDocumento\_Cliente\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //cliente sin tipo documento

/\* 6 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearNúmeroDocumento\_Cliente\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //cliente sin número de documento

/\* 7 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearPropietario\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //propietario del inmueble null

/\* 8 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearNombre\_Propietario\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //propietario del inmueble sin nombre

/\* 9 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearApellido\_Propietario\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //propietario del inmueble sin apellido

/\* 10 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearInmueble\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //inmueble null

/\* 11 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearTipo\_Inmueble\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //inmueble sin tipo inmueble

/\* 12 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearDirección\_Inmueble\_Vacía***, 1, 0, 0, 1, **null**, 0 }, //inmueble sin dirección

/\* 13 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearLocalidad\_Inmueble\_Vacía***, 1, 0, 0, 1, **null**, 0 }, //inmueble sin localidad

/\* 14 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearBarrio\_Inmueble\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //inmueble sin barrio

/\* 15 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearCalle\_Inmueble\_Vacía***, 1, 0, 0, 1, **null**, 0 }, //inmueble sin calle

/\* 16 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearAltura\_Inmueble\_Vacía***, 1, 0, 0, 1, **null**, 0 }, //inmueble sin altura

/\* 17 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearFechaInicio\_vacía***, 1, 0, 0, 1, **null**, 0 }, //sin fecha inicio

/\* 18 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearFechaFin\_vacía***, 1, 0, 0, 1, **null**, 0 }, //sin fecha fin

/\* 19 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearFecha\_Inicio\_Posterior\_A\_Fecha\_Fin***, 1, 0, 0, 1, **null**, 0 }, //fecha inicio posterior a fecha fin

/\* 20 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearExiste\_Otra\_Reserva\_Activa***, 1, 0, 0, 1, **null**, 0 }, //ya existe otra reserva sobre el inmueble en ese período

/\* 21 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearImporte\_Vacío***, 1, 0, 0, 1, **null**, 0 }, //importe null

/\* 22 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearImporte\_Menor\_O\_Igual\_A\_Cero***, 1, 0, 0, 1, **null**, 0 }, //importe menor o igual a cero

/\* 23 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCrearInmuebleVendido***, 1, 0, 0, 1, **null**, 0 }, //importe menor o igual a cero

/\* 24 \*/ **new** Object[] { inmueble, cliente, "10", fechaInicioCorrecta, fechaFinCorrecta, ***resultadoCorrecto***, 0, 1, 0, 1, **new** SaveUpdateException(**new** Throwable()), 0 }, //excepcion al guardar en base de datos

/\*\*

\* Test para probar la función de alta reserva del controlador

\*

\* **@param** inmuebleAReservar

\* inmueble de la reserva

\* **@param** clienteSeleccionado

\* cliente asociado a la reserva

\* **@param** importe

\* importe por el cual se genera la reserva

\* **@param** fechaInicio

\* fecha de inicio de la reserva

\* **@param** fechaFin

\* fecha de finalización del período de vigencia de la reserva

\* **@param** resultadoCrearReserva

\* resultado devuelto por el mock de la capa lógica

\* **@param** llamaAPresentadorVentanasPresentarError

\* indica si se debe presentar una ventana de error

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* indica si se debe presentar una ventana de excepción

\* **@param** llamaAPresentadorVentanasPresentarExcepcionInesperada

\* indica si se debe presentar una ventana de exepción inesperada

\* **@param** llamaACrearReserva

\* indica si se debe llamar a crear la reserva en la lógica

\* **@param** excepcion

\* excepción devuelta por la capa lógica

\* **@param** reservaExitosa

\* indica se la generación de la reserva debe ser exitosa

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCrearReserva(Inmueble inmuebleAReservar, Cliente clienteSeleccionado, String importe, LocalDate fechaInicio, LocalDate fechaFin, ResultadoCrearReserva resultadoCrearReserva, Integer llamaAPresentadorVentanasPresentarError, Integer llamaAPresentadorVentanasPresentarExcepcion, Integer llamaAPresentadorVentanasPresentarExcepcionInesperada, Integer llamaACrearReserva, Exception excepcion, Integer reservaExitosa) **throws** Throwable {

//Se crean los mocks necesarios

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaErrorExcepcionInesperada ventanaErrorExcepcionInesperadaMock = *mock*(VentanaErrorExcepcionInesperada.**class**);

ScenographyChanger scenographyChangerMock = *mock*(ScenographyChanger.**class**);

VerPDFController verPDFControllerMock = *mock*(VerPDFController.**class**);

//Se setea lo que deben devolver los mocks cuando son invocados por la clase a probar

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarExcepcionInesperada(*any*(), *any*())).thenReturn(ventanaErrorExcepcionInesperadaMock);

*when*(scenographyChangerMock.cambiarScenography(*eq*(VerPDFController.***URLVista***), *any*(Boolean.**class**))).thenReturn(verPDFControllerMock);

*doNothing*().when(verPDFControllerMock).setVendedorLogueado(*any*(Vendedor.**class**));

*doNothing*().when(verPDFControllerMock).cargarPDF(*any*());

*doNothing*().when(presentadorVentanasMock).presentarToast(*any*(), *any*());

Reserva reserva = **new** Reserva()

.setImporte(Double.*valueOf*(importe))

.setFechaFin(Date.*from*(fechaFin.atStartOfDay(ZoneId.*systemDefault*()).toInstant()))

.setFechaInicio(Date.*from*(fechaInicio.atStartOfDay(ZoneId.*systemDefault*()).toInstant()))

.setCliente(clienteSeleccionado)

.setInmueble(inmuebleAReservar);

*when*(coordinadorMock.crearReserva(reserva)).thenReturn(resultadoCrearReserva);

**if**(excepcion != **null**){

*when*(coordinadorMock.crearReserva(reserva)).thenThrow(excepcion);

}

ArrayList<Cliente> clientes = **new** ArrayList<>();

clientes.add(clienteSeleccionado);

*when*(coordinadorMock.obtenerClientes()).thenReturn(clientes);

ArrayList<Inmueble> inmuebles = **new** ArrayList<>();

inmuebles.add(inmuebleAReservar);

*when*(coordinadorMock.obtenerInmuebles()).thenReturn(inmuebles);

//Controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos que ingresaría el usuario en la vista

AltaReservaController altaReservaController = **new** AltaReservaController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.setScenographyChanger(scenographyChangerMock);

**this**.presentador = presentadorVentanasMock;

**super**.inicializar(location, resources);

}

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** acceptAction() {

**this**.textFieldImporte.setText(importe);

**this**.datePickerInicio.setValue(fechaInicio);

**this**.datePickerFin.setValue(fechaFin);

**this**.comboBoxCliente.getSelectionModel().select(clienteSeleccionado);

**this**.comboBoxInmueble.getSelectionModel().select(inmuebleAReservar);

**this**.inmueble = inmuebleAReservar;

**super**.acceptAction();

};

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AltaReservaController.***URLVista***, altaReservaController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Método a probar

altaReservaController.acceptAction();

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(coordinadorMock).obtenerClientes();

Mockito.*verify*(coordinadorMock, *times*(llamaACrearReserva)).crearReserva(*any*());

Mockito.*verify*(scenographyChangerMock, *times*(reservaExitosa)).cambiarScenography(*eq*(VerPDFController.***URLVista***), *any*(Boolean.**class**));

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcion)).presentarExcepcion(*eq*(excepcion), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcionInesperada)).presentarExcepcionInesperada(*eq*(excepcion), *any*());

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

### Taskcard 25 Lógica alta, baja reserva y generar PDF

Código del archivo GestorReserva.java

/\*\*

\* Método para crear una reserva. Primero se validan las reglas de negocio y luego se persiste.

\* Pertenece a la taskcard 25 de la iteración 2 y a la historia 7

\*

\* **@param** reserva

\* a guardar

\* **@return** resultado de la operación

\* **@throws** PersistenciaException

\* si falló al persistir

\* **@throws** GestionException

\* si falló al procesar la reserva

\*/

**public** ResultadoCrearReserva crearReserva(Reserva reserva) **throws** PersistenciaException, GestionException {

//Inicialización de variables

Set<ErrorCrearReserva> errores = **new** HashSet<>();

**boolean** reservaEnConflictoEncontrada = **false**;

**boolean** fechaInicioVacia = **false**;

**boolean** fechaFinVacia = **false**;

Reserva reservaEnConflicto = **null**;

//Validaciones de lógica

**if**(reserva.getCliente() == **null**){

errores.add(ErrorCrearReserva.***Cliente\_Vacío***);

}

**else**{

**if**(reserva.getCliente().getNombre() == **null**){

errores.add(ErrorCrearReserva.***Nombre\_Cliente\_Vacío***);

}

**if**(reserva.getCliente().getApellido() == **null**){

errores.add(ErrorCrearReserva.***Apellido\_Cliente\_Vacío***);

}

**if**(reserva.getCliente().getTipoDocumento() == **null** || reserva.getCliente().getTipoDocumento().getTipo() == **null**){

errores.add(ErrorCrearReserva.***TipoDocumento\_Cliente\_Vacío***);

}

**if**(reserva.getCliente().getNumeroDocumento() == **null**){

errores.add(ErrorCrearReserva.***NúmeroDocumento\_Cliente\_Vacío***);

}

}

**if**(reserva.getInmueble() == **null**){

errores.add(ErrorCrearReserva.***Inmueble\_Vacío***);

}

**else**{

**if**(reserva.getInmueble().getEstadoInmueble().getEstado().equals(EstadoInmuebleStr.***VENDIDO***)){

errores.add(ErrorCrearReserva.***Inmueble\_Vendido***);

}

**if**(reserva.getInmueble().getPropietario() == **null**){

errores.add(ErrorCrearReserva.***Propietario\_Vacío***);

}

**else**{

**if**(reserva.getInmueble().getPropietario().getNombre() == **null**){

errores.add(ErrorCrearReserva.***Nombre\_Propietario\_Vacío***);

}

**if**(reserva.getInmueble().getPropietario().getApellido() == **null**){

errores.add(ErrorCrearReserva.***Apellido\_Propietario\_Vacío***);

}

}

**if**(reserva.getInmueble().getTipo() == **null** || reserva.getInmueble().getTipo().getTipo() == **null**){

errores.add(ErrorCrearReserva.***Tipo\_Inmueble\_Vacío***);

}

**if**(reserva.getInmueble().getDireccion() == **null**){

errores.add(ErrorCrearReserva.***Dirección\_Inmueble\_Vacía***);

}

**else**{

**if**(reserva.getInmueble().getDireccion().getLocalidad() == **null**){

errores.add(ErrorCrearReserva.***Localidad\_Inmueble\_Vacía***);

}

**if**(reserva.getInmueble().getDireccion().getBarrio() == **null**){

errores.add(ErrorCrearReserva.***Barrio\_Inmueble\_Vacío***);

}

**if**(reserva.getInmueble().getDireccion().getCalle() == **null**){

errores.add(ErrorCrearReserva.***Calle\_Inmueble\_Vacía***);

}

**if**(reserva.getInmueble().getDireccion().getNumero() == **null**){

errores.add(ErrorCrearReserva.***Altura\_Inmueble\_Vacía***);

}

}

}

**if**(reserva.getFechaInicio() == **null**){

errores.add(ErrorCrearReserva.***FechaInicio\_vacía***);

fechaInicioVacia = **true**;

}

**if**(reserva.getFechaFin() == **null**){

errores.add(ErrorCrearReserva.***FechaFin\_vacía***);

fechaFinVacia = **true**;

}

**if**(!fechaInicioVacia && !fechaFinVacia){

//Validamos que el rango de fechas no coincida con el de otra reserva

**if**(reserva.getFechaInicio().compareTo(reserva.getFechaFin()) > 0){

errores.add(ErrorCrearReserva.***Fecha\_Inicio\_Posterior\_A\_Fecha\_Fin***);

}

**else** **if**(reserva.getInmueble() != **null**){

Set<Reserva> reservasDelInmueble = reserva.getInmueble().getReservas();

Iterator<Reserva> itRes = reservasDelInmueble.iterator();

Reserva res = **null**;

**while**(!reservaEnConflictoEncontrada && itRes.hasNext()){

res = itRes.next();

**if**(res.getEstado().getEstado().equals(EstadoStr.***ALTA***)

&& res.getFechaFin().compareTo(reserva.getFechaInicio()) > 0

&& res.getFechaInicio().compareTo(reserva.getFechaFin()) < 0){

reservaEnConflictoEncontrada = **true**;

}

}

**if**(reservaEnConflictoEncontrada){

reservaEnConflicto = res;

errores.add(ErrorCrearReserva.***Existe\_Otra\_Reserva\_Activa***);

}

}

}

**if**(reserva.getImporte() == **null**){

errores.add(ErrorCrearReserva.***Importe\_Vacío***);

}

**else** **if**(reserva.getImporte() <= 0){

errores.add(ErrorCrearReserva.***Importe\_Menor\_O\_Igual\_A\_Cero***);

}

**if**(errores.isEmpty()){

//Si no hay errores generamos el pdf de la reserva

**final** PDF pdfReserva = gestorPDF.generarPDF(reserva);

//se lo seteamos a reserva

reserva.setArchivoPDF(pdfReserva);

//también le seteamos el estado

Estado estadoAlta = **null**;

**for**(Estado estado: gestorDatos.obtenerEstados()){

**if**(estado.getEstado().equals(EstadoStr.***ALTA***)){

estadoAlta = estado;

}

}

reserva.setEstado(estadoAlta);

//Mandamos un mail con la reserva (de forma asincrónica)

**new** Thread(() -> {

**try**{

**if**(reserva.getCliente().getCorreo() != **null**){

gestorEmail.enviarEmail(reserva.getCliente().getCorreo(), ***ASUNTO\_RESERVA\_CREADA***, ***MENSAJE\_RESERVA\_CREADA***, pdfReserva);

}

} **catch**(IOException | MessagingException e){

e.printStackTrace();

}

}).start();

//Persistimos la reserva

persistidorReserva.guardarReserva(reserva);

//Retornamos el PDF generado

**return** **new** ResultadoCrearReserva(pdfReserva);

}

//Retornamos los errores

**return** **new** ResultadoCrearReserva(reservaEnConflicto, errores.toArray(**new** ErrorCrearReserva[0]));

}

/\*\*

\* Método para eliminar una reserva. Se hace una baja lógica.

\* Pertenece a la taskcard 25 de la iteración 2 y a la historia 7

\*

\* **@param** reserva

\* a eliminar

\* **@return** resultado de la operación

\* **@throws** PersistenciaException

\* si falló al persistir

\*/

**public** ResultadoEliminarReserva eliminarReserva(Reserva reserva) **throws** PersistenciaException {

//Seteamos el estado de la reserva

ArrayList<Estado> estados = gestorDatos.obtenerEstados();

Estado estadoBaja = **null**;

**for**(Estado estado: estados){

**if**(estado.getEstado().equals(EstadoStr.***BAJA***)){

estadoBaja = estado;

}

}

reserva.setEstado(estadoBaja);

//Persistimos el cambio

persistidorReserva.modificarReserva(reserva);

**return** **new** ResultadoEliminarReserva();

}

Prueba de unidad del archivo GestorReservaTest.java

//Casos de prueba

//reserva, resultado, excepcion

/\* 0 \*/**new** Object[] { reservaCorrecta, **new** ResultadoCrearReserva(**null**), **null** }, //reserva correcta

/\* 1 \*/**new** Object[] { reservaSinCliente, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Cliente\_Vacío***), **null** }, //reserva sin cliente

/\* 2 \*/**new** Object[] { reservaSinNombreCliente, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Nombre\_Cliente\_Vacío***), **null** }, //reserva sin nombre cliente

/\* 3 \*/**new** Object[] { reservaSinApellidoCliente, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Apellido\_Cliente\_Vacío***), **null** }, //reserva sin apellido cliente

/\* 4 \*/**new** Object[] { reservaSinTipoDocumentoCliente, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***TipoDocumento\_Cliente\_Vacío***), **null** }, //reserva sin tipo documento en el cliente

/\* 5 \*/**new** Object[] { reservaSinNumeroDocumentoCliente, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***NúmeroDocumento\_Cliente\_Vacío***), **null** }, //reserva sin numero de documento en el cliente

/\* 6 \*/**new** Object[] { reservaSinInmueble, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Inmueble\_Vacío***), **null** }, //reserva sin inmueble

/\* 7 \*/**new** Object[] { reservaSinPropietario, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Propietario\_Vacío***), **null** }, //reserva sin Propietario

/\* 8 \*/**new** Object[] { reservaSinNombrePropietario, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Nombre\_Propietario\_Vacío***), **null** }, //reserva sin nombre de Propietario

/\* 9 \*/**new** Object[] { reservaSinApellidoPropietario, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Apellido\_Propietario\_Vacío***), **null** }, //reserva sin apellido de Propietario

/\* 10 \*/**new** Object[] { reservaSinTipoInmueble, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Tipo\_Inmueble\_Vacío***), **null** }, //reserva sin tipo de Inmueble

/\* 11 \*/**new** Object[] { reservaSinDireccionInmueble, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Dirección\_Inmueble\_Vacía***), **null** }, //reserva sin direccion de inmueble

/\* 12 \*/**new** Object[] { reservaSinLocalidadInmueble, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Localidad\_Inmueble\_Vacía***), **null** }, //reserva sin localidad de inmueble

/\* 13 \*/**new** Object[] { reservaSinBarrioInmueble, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Barrio\_Inmueble\_Vacío***), **null** }, //reserva sin Barrio de inmueble

/\* 14 \*/**new** Object[] { reservaSinCalleInmueble, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Calle\_Inmueble\_Vacía***), **null** }, //reserva sin calle de inmueble

/\* 15 \*/**new** Object[] { reservaSinAlturaInmueble, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Altura\_Inmueble\_Vacía***), **null** }, //reserva sin altura de calle de inmueble

/\* 16 \*/**new** Object[] { reservaSinFechaInicio, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***FechaInicio\_vacía***), **null** }, //reserva sin fecha de inicio vacía

/\* 17 \*/**new** Object[] { reservaSinFechaFin, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***FechaFin\_vacía***), **null** }, //reserva sin fecha de fin vacía

/\* 18 \*/**new** Object[] { reservaFechaInicioPosteriorAFechaFin, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Fecha\_Inicio\_Posterior\_A\_Fecha\_Fin***), **null** }, //reserva con fecha inicio posterior a fecha fin

/\* 19 \*/**new** Object[] { reservaConflictiva, **new** ResultadoCrearReserva(reservaConflictiva, ErrorCrearReserva.***Existe\_Otra\_Reserva\_Activa***), **null** }, //existe otra reserva en el mismo rango de fechas

/\* 20 \*/**new** Object[] { reservaSinImporte, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Importe\_Vacío***), **null** }, //reserva sin Importe

/\* 21 \*/**new** Object[] { reservaImporteMenorIgualCero, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Importe\_Menor\_O\_Igual\_A\_Cero***), **null** }, //reserva con importe menor o igual a cero

/\* 22 \*/**new** Object[] { reservaInmuebleVendido, **new** ResultadoCrearReserva(**null**, ErrorCrearReserva.***Inmueble\_Vendido***), **null** }, //reserva con inmueble vendido

/\* 23 \*/**new** Object[] { reservaCorrecta, **null**, **new** GenerarPDFException(**new** Exception()) }, //el gestorPDF tira una excepción

/\* 24 \*/**new** Object[] { reservaCorrecta, **null**, **new** ObjNotFoundException("", **new** Exception()) }, //el persistidor tira una excepción

/\* 25 \*/**new** Object[] { reservaCorrecta, **null**, **new** Exception() } //el persistidor tira una excepción inesperada

/\*\*

\* Prueba el método crearReserva(), el cual corresponde con la taskcard 25 de la iteración 2 y a la historia 7

\*

\* **@param** reserva

\* reserva a crear

\* **@param** resultado

\* resultado que se espera que devuelva el gestor

\* **@param** excepcion

\* es la excepcion que debe lanzar el mock del persistidor o del gestorPDF, si la prueba involucra procesar una excepcion de dicho persistidor/gestor, debe ser nulo propietario para que

\* se use

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testCrearReserva(Reserva reserva, ResultadoCrearReserva resultado, Throwable excepcion) **throws** Exception {

//Se crean los mocks de la prueba

GestorDatos gestorDatosMock = **new** GestorDatos() {

@Override

**public** ArrayList<Estado> obtenerEstados() **throws** PersistenciaException {

**return** **new** ArrayList<>();

}

};

GestorPDF gestorPDFMock = **new** GestorPDF() {

@Override

**public** PDF generarPDF(Reserva reserva) **throws** GestionException {

**if**(excepcion != **null** && excepcion **instanceof** GestionException){

**throw** (GestionException) excepcion;

}

**return** **new** PDF();

}

};

GestorEmail gestorEmailMock = **new** GestorEmail(**true**) {

@Override

**public** **void** enviarEmail(String destinatario, String asunto, String mensaje, Archivo archivo)

**throws** IOException, MessagingException {

}

};

ReservaService persistidorReservaMock = **new** ReservaService() {

@Override

**public** **void** guardarReserva(Reserva reserva) **throws** PersistenciaException {

**if**(excepcion != **null**){

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**else** **if**(!(excepcion **instanceof** GestionException)){

**new** Integer("asd");

}

}

}

@Override

**public** **void** modificarReserva(Reserva reserva) **throws** PersistenciaException {

}

@Override

**public** ArrayList<Reserva> obtenerReservas(Cliente cliente) **throws** PersistenciaException {

**return** **null**;

}

@Override

**public** ArrayList<Reserva> obtenerReservas(Inmueble inmueble) **throws** PersistenciaException {

**return** **null**;

}

};

//Se crea el gestor a probar, se sobreescriben algunos métodos para setear los mocks

GestorReserva gestorReserva = **new** GestorReserva() {

{

**this**.gestorPDF = gestorPDFMock;

**this**.persistidorReserva = persistidorReservaMock;

**this**.gestorDatos = gestorDatosMock;

**this**.gestorEmail = gestorEmailMock;

}

};

//Creamos la prueba

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

**if**(resultado != **null**){

//Se hacen las verificaciones pertinentes para comprobar que el gestor se comporte adecuadamente

*assertEquals*(resultado, gestorReserva.crearReserva(reserva));

}

**else**{

**try**{

gestorReserva.crearReserva(reserva);

Assert.*fail*("Debería haber fallado!");

} **catch**(PersistenciaException | GestionException e){

Assert.*assertEquals*((excepcion), e);

} **catch**(Exception e){

**if**(excepcion **instanceof** PersistenciaException){

Assert.*fail*("Debería haber tirado una PersistenciaException y tiro otra Exception!");

}

}

}

}

};

//Ejecutamos la prueba

**try**{

test.evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

//Casos de prueba

//reserva, resultado, excepcion

/\* 0 \*/**new** Object[] { reservaCorrecta, **new** ResultadoEliminarReserva(), **null** }, //reserva correcta

/\* 1 \*/**new** Object[] { reservaCorrecta, **null**, **new** ObjNotFoundException("", **new** Exception()) }, //el persistidor tira una excepción

/\* 2 \*/**new** Object[] { reservaCorrecta, **null**, **new** Exception() } //el persistidor tira una excepción inesperada

/\*\*

\* Prueba el método eliminarReserva(), el cual corresponde con la taskcard 25 de la iteración 2 y a la historia 7

\*

\* **@param** reserva

\* reserva a eliminar

\* **@param** resultado

\* resultado que se espera que devuelva el gestor

\* **@param** excepcion

\* es la excepcion que debe lanzar el mock del persistidor, si la prueba involucra procesar una excepcion de dicho persistidor, debe ser nulo propietario para que

\* se use

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testEliminarReserva(Reserva reserva, ResultadoEliminarReserva resultado, Throwable excepcion) **throws** Exception {

//Se crean los mocks de la prueba

GestorDatos gestorDatosMock = **new** GestorDatos() {

@Override

**public** ArrayList<Estado> obtenerEstados() **throws** PersistenciaException {

**return** **new** ArrayList<>();

}

};

ReservaService persistidorReservaMock = **new** ReservaService() {

@Override

**public** **void** guardarReserva(Reserva reserva) **throws** PersistenciaException {

}

@Override

**public** **void** modificarReserva(Reserva reserva) **throws** PersistenciaException {

**if**(excepcion != **null**){

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**else** **if**(!(excepcion **instanceof** GestionException)){

**new** Integer("asd");

}

}

}

@Override

**public** ArrayList<Reserva> obtenerReservas(Cliente cliente) **throws** PersistenciaException {

**return** **null**;

}

@Override

**public** ArrayList<Reserva> obtenerReservas(Inmueble inmueble) **throws** PersistenciaException {

**return** **null**;

}

};

//Se crea el gestor a probar, se sobreescriben algunos métodos para setear los mocks

GestorReserva gestorReserva = **new** GestorReserva() {

{

**this**.persistidorReserva = persistidorReservaMock;

**this**.gestorDatos = gestorDatosMock;

}

};

//Creamos la prueba

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

**if**(resultado != **null**){

//Se hacen las verificaciones pertinentes para comprobar que el gestor se comporte adecuadamente

*assertEquals*(resultado, gestorReserva.eliminarReserva(reserva));

}

**else**{

**try**{

gestorReserva.eliminarReserva(reserva);

Assert.*fail*("Debería haber fallado!");

} **catch**(PersistenciaException e){

Assert.*assertEquals*((excepcion), e);

} **catch**(Exception e){

**if**(excepcion **instanceof** PersistenciaException){

Assert.*fail*("Debería haber tirado una PersistenciaException y tiro otra Exception!");

}

}

}

}

};

//Ejecutamos la prueba

**try**{

test.evaluate();

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

Código del archivo GestorPDF.java

/\*\*

\* Método para crear un PDF de una reserva a partir de los datos de una ReservaVista.

\* Pertenece a la taskcard 25 de la iteración 2 y a la historia 7

\*

\* **@param** reserva

\* datos que se utilizaran para generar el PDF de una reserva

\* **@return** reserva en PDF

\*/

**public** PDF generarPDF(Reserva reserva) **throws** GestionException {

**try**{

//Inicialización de parámetros

pdf = **null**;

//Cargar pantalla que representa al PDF

FXMLLoader loader = **new** FXMLLoader();

loader.setLocation(getClass().getResource(***URLDocumentoReserva***));

Pane documentoReserva = (Pane) loader.load();

FutureTask<Throwable> future = **new** FutureTask<>(() -> {

**try**{

//Cargar datos a la pantalla que representa al PDF

Label label = (Label) documentoReserva.lookup("#lblNombreOferente");

label.setText(formateador.nombrePropio(reserva.getCliente().getNombre()));

label = (Label) documentoReserva.lookup("#lblApellidoOferente");

label.setText(formateador.nombrePropio(reserva.getCliente().getApellido()));

label = (Label) documentoReserva.lookup("#lblDocumentoOferente");

label.setText(reserva.getCliente().getTipoDocumento() + " - " + reserva.getCliente().getNumeroDocumento());

label = (Label) documentoReserva.lookup("#lblNombrePropietario");

label.setText(formateador.nombrePropio(reserva.getInmueble().getPropietario().getNombre()));

label = (Label) documentoReserva.lookup("#lblApellidoPropietario");

label.setText(formateador.nombrePropio(reserva.getInmueble().getPropietario().getApellido()));

label = (Label) documentoReserva.lookup("#lblCodigoInmueble");

label.setText(Integer.*toString*(reserva.getInmueble().getId()));

label = (Label) documentoReserva.lookup("#lblTipoInmueble");

label.setText(reserva.getInmueble().getTipo().getTipo().toString());

label = (Label) documentoReserva.lookup("#lblLocalidadInmueble");

label.setText(reserva.getInmueble().getDireccion().getLocalidad().toString());

label = (Label) documentoReserva.lookup("#lblBarrioInmueble");

label.setText(reserva.getInmueble().getDireccion().getBarrio().toString());

label = (Label) documentoReserva.lookup("#lblCalleInmueble");

label.setText(reserva.getInmueble().getDireccion().getCalle().toString());

label = (Label) documentoReserva.lookup("#lblAlturaInmueble");

label.setText(reserva.getInmueble().getDireccion().getNumero());

label = (Label) documentoReserva.lookup("#lblPisoInmueble");

label.setText(reserva.getInmueble().getDireccion().getPiso());

label = (Label) documentoReserva.lookup("#lblDepartamentoInmueble");

label.setText(reserva.getInmueble().getDireccion().getDepartamento());

label = (Label) documentoReserva.lookup("#lblOtrosInmueble");

label.setText(reserva.getInmueble().getDireccion().getOtros());

label = (Label) documentoReserva.lookup("#lblImporte");

label.setText(String.*format*("$ %10.2f", reserva.getImporte()));

label = (Label) documentoReserva.lookup("#lblFechaRealizacion");

label.setText(conversorFechas.diaMesYAnioToString(reserva.getFechaInicio()));

label = (Label) documentoReserva.lookup("#lblFechaVencimiento");

label.setText(conversorFechas.diaMesYAnioToString(reserva.getFechaFin()));

label = (Label) documentoReserva.lookup("#lblHoraGenerado");

Date ahora = **new** Date();

label.setText(String.*format*(label.getText(), conversorFechas.horaYMinutosToString(ahora), conversorFechas.diaMesYAnioToString(ahora)));

//Generación del archivo

pdf = generarPDF(documentoReserva);

} **catch**(Throwable e){

**return** e;

}

**return** **null**;

});

//Asegurarse de que lo anterior se corra en el hilo de javaFX

**if**(!Platform.*isFxApplicationThread*()){

Platform.*runLater*(future);

}

**else**{

future.run();

}

//Si hubo error se lanza excepción

Throwable excepcion = future.get();

**if**(excepcion != **null**){

**throw** excepcion;

}

**if**(pdf == **null**){

**throw** **new** NullPointerException("Error al generar PDF");

}

} **catch**(Throwable e){

**throw** **new** GenerarPDFException(e);

}

**return** pdf;

}

/\*\*

\* Método para crear un PDF a partir de una pantalla.

\*

\* **@param** pantallaAPDF

\* pantalla que se imprimirá en PDF

\* **@return** PDF de una captura de la pantalla pasada

\*/

**private** PDF generarPDF(Node pantallaAPDF) **throws** Exception {

//Se imprime la pantalla en una imagen

**new** Scene((Parent) pantallaAPDF);

WritableImage image = pantallaAPDF.snapshot(**new** SnapshotParameters(), **null**);

ByteArrayOutputStream baos = **new** ByteArrayOutputStream();

ImageIO.*write*(SwingFXUtils.*fromFXImage*(image, **null**), "png", baos);

**byte**[] imageInByte = baos.toByteArray();

baos.flush();

baos.close();

//Se carga la imagen en un PDF

Image imagen = Image.*getInstance*(imageInByte);

Document document = **new** Document();

ByteArrayOutputStream pdfbaos = **new** ByteArrayOutputStream();

PdfWriter escritor = PdfWriter.*getInstance*(document, pdfbaos);

document.open();

imagen.setAbsolutePosition(0, 0);

imagen.scaleToFit(PageSize.***A4***.getWidth(), PageSize.***A4***.getHeight());

document.add(imagen);

document.close();

//Se obtiene el archivo PDF

**byte**[] pdfBytes = pdfbaos.toByteArray();

pdfbaos.flush();

escritor.close();

pdfbaos.close();

//Se genera un objeto PDF

**return** (PDF) **new** PDF().setArchivo(pdfBytes);

}

Prueba de unidad del archivo GestorPDFTest.java

/\*\*

\* Prueba el método generarPDF(Reserva reserva), el cual corresponde con la taskcard 25 de la iteración 2 y a la historia 7

\* El test es en parte manual ya que genera un archivo pdf que debe comprobarse si es correcto manualmente.

\*

\* **@throws** Exception

\*/

@Test

**public** **void** testGenerarPDFReserva() **throws** Exception {

**new** ControladorTest(LoginController.***URLVista***, **new** LoginController() {

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

}

});

GestorPDF gestor = **new** GestorPDF() {

{

formateador = **new** FormateadorString();

conversorFechas = **new** ConversorFechas();

}

};

Cliente cliente = **new** Cliente().setNombre("Pablo")

.setApellido("Van Derdonckt")

.setTipoDocumento(**new** TipoDocumento().setTipo(TipoDocumentoStr.***DNI***))

.setNumeroDocumento("36696969");

Propietario propietario = **new** Propietario().setNombre("Esteban")

.setApellido("Rebechi");

Localidad localidad = **new** Localidad().setProvincia(**new** Provincia().setPais(**new** Pais())).setNombre("Ceres");

Direccion direccion = **new** Direccion().setCalle(**new** Calle().setLocalidad(localidad).setNombre("Azquenaga")).setLocalidad(localidad).setBarrio(**new** Barrio().setLocalidad(localidad).setNombre("Vicente Zaspe"))

.setNumero("3434")

.setPiso("6")

.setDepartamento("6B")

.setOtros("otros");

Inmueble inmueble = **new** Inmueble() {

@Override

**public** Integer getId() {

**return** 12345;

};

}.setTipo(**new** TipoInmueble(TipoInmuebleStr.***DEPARTAMENTO***))

.setDireccion(direccion)

.setPropietario(propietario);

Date fechahoy = **new** Date();

Reserva reserva = **new** Reserva().setCliente(cliente).setInmueble(inmueble).setImporte(300000.50).setFechaInicio(fechahoy).setFechaFin(fechahoy);

PDF pdf = gestor.generarPDF(reserva);

FileOutputStream fos = **new** FileOutputStream("testReserva.pdf");

fos.write(pdf.getArchivo());

fos.close();

}

### Taskcard 26 Lógica envío de mail

Código del archivo GestorEmail.java

/\*\*

\* Método para el envío de email usando la API de Gmail

\*

\* **@param** destinatario

\* destinatario del email

\* **@param** asunto

\* asunto del email

\* **@param** mensaje

\* cuerpo del mensaje del email

\* **@param** archivo

\* arhivo que será adjuntado al email

\* **@throws** IOException

\* **@throws** MessagingException

\*/

**public** **void** enviarEmail(String destinatario, String asunto, String mensaje, Archivo archivo) **throws** IOException, MessagingException {

// Build a new authorized API client service.

//Se obtiene un servicio de cliente de la API

//Además se solicitan los permisos necesarios al usuario

Gmail service = getGmailService();

//La palabra clave "me" representa al dueño de la cuenta con la que se enviará el email

String user = "me";

//Se crea un archivo temporal para poder enviarlo como adjunto

File archivoTMP = **new** File(***URL\_RESERVA***);

FileOutputStream fos = **new** FileOutputStream(archivoTMP);

fos.write(archivo.getArchivo());

fos.flush();

fos.close();

//Se invoca al método de la API con los parámetros necesarios, se pasa la dirección de email de la inmobiliaria como campo "from"

sendMessage(service, user, createEmailWithAttachment(destinatario, "olimpoagilinmobiliaria2016@gmail.com", asunto, mensaje, archivoTMP));

//Finalmente se borra el archivo temporal

archivoTMP.delete();

}

### Taskcard 27 Cambios ABM Cliente

Código del archivo Cliente.java

@Column(name = "correo", length = 100)

**private** String correo;

Código del archivo AltaClienteController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga al cliente y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, se muestran al usuario.

\*/

@FXML

**public** **void** acceptAction() {

(…)

//obtengo datos introducidos por el usuario

(…)

String correo = textFieldCorreo.getText().trim();

(…)

**if**(correo.isEmpty()){

error.append("Inserte una dirección de correo electrónico").append("\n");

}

(…)

cliente.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setCorreo(correo);

(…)

**case** ***Formato\_Correo\_Incorrecto***:

stringErrores.append("Formato de correo electrónico incorrecto.\n");

**break**;

(…)

Código del archivo ModificarClienteController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga al cliente y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, se muestran al usuario.

\*/

@FXML

**protected** **void** acceptAction() {

(…)

//obtengo datos introducidos por el usuario

(…)

String correo = textFieldCorreo.getText().trim();

(…)

//verifico que no estén vacíos

**if**(correo.isEmpty()){

error.append("Inserte una dirección de correo electrónico").append("\n");

}

(…)

clienteEnModificacion.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setCorreo(correo);

(…)

**case** ***Formato\_Correo\_Incorrecto***:

stringErrores.append("Formato de correo electrónico incorrecto.\n");

**break**;

(…)

Código del archivo GestorCliente.java

/\*\*

\* Se encarga de validar los datos de un cliente a crear y, en caso de que no haya errores,

\* delegar el guardado del objeto a la capa de acceso a datos.

\*

\* Modificada en TaskCard 27 de la iteración 2

\*

\* **@param** cliente

\* cliente a crear

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\* **@throws** GestionException

\* se lanza una excepción EntidadExistenteConEstadoBaja cuando se encuentra que ya existe un vendedor con la misma identificación pero tiene estado BAJA

\*/

**public** ResultadoCrearCliente crearCliente(Cliente cliente) **throws** PersistenciaException, GestionException {

(…)

**if**(!validador.validarEmail(cliente.getCorreo())){

errores.add(ErrorCrearCliente.***Formato\_Correo\_Incorrecto***);

}

(…)

/\*\*

\* Se encarga de validar los datos de un cliente a modificar y, en caso de que no haya errores,

\* delegar el guardado del objeto a la capa de acceso a datos.

\*

\* Modificada en TaskCard 27 de la iteración 2

\*

\* **@param** cliente

\* cliente a modificar

\* **@return** un resultado informando errores correspondientes en caso de que los haya

\*

\* **@throws** PersistenciaException

\* se lanza esta excepción al ocurrir un error interactuando con la capa de acceso a datos

\*/

**public** ResultadoModificarCliente modificarCliente(Cliente cliente) **throws** PersistenciaException {

(…)

**if**(!validador.validarEmail(cliente.getCorreo())){

errores.add(ErrorModificarCliente.***Formato\_Correo\_Incorrecto***);

}

(…)

### Taskcard 28 Persistidor y entidad reserva

Código del archivo Reserva.java

/\*\*

\* Entidad que modela una reserva

\* Pertenece a la taskcard 28 de la iteración 2 y a la historia 7

\*/

**public** **class** Reserva {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** Integer id; //ID

@Column(name = "importe", nullable = **false**)

**private** Double importe;

@Column(name = "fecha\_inicio", nullable = **false**)

**private** Date fechaInicio;

@Column(name = "fecha\_fin", nullable = **false**)

**private** Date fechaFin;

@OneToOne(cascade = CascadeType.***ALL***, orphanRemoval = **true**, optional = **false**)

@JoinColumn(name = "idarchivo", foreignKey = @ForeignKey(name = "reserva\_idarchivo\_fk"), nullable = **false**)

**private** PDF archivoPDF;

//Relaciones

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idcliente", referencedColumnName = "id", foreignKey = @ForeignKey(name = "reserva\_idcliente\_fk"), nullable = **false**)

**private** Cliente cliente;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idinmueble", referencedColumnName = "id", foreignKey = @ForeignKey(name = "reserva\_idinmueble\_fk"), nullable = **false**)

**private** Inmueble inmueble;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idestado", referencedColumnName = "id", foreignKey = @ForeignKey(name = "reserva\_idestado\_fk"), nullable = **false**)

**private** Estado estado;

Código del archivo ReservaServiceImpl.java

/\*

\* Método para guardar en la base de datos una reserva

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** **void** guardarReserva(Reserva reserva) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.save(reserva);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

/\*

\* Método para modificar en la base de datos una reserva

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** **void** modificarReserva(Reserva reserva) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.update(reserva);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

/\*

\* Método para obtener todas las reservas de un cliente

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** ArrayList<Reserva> obtenerReservas(Cliente cliente) **throws** PersistenciaException {

ArrayList<Reserva> reservas = **new** ArrayList<>();

Session session = getSessionFactory().getCurrentSession();

**try**{

//named query ubicada en entidad reserva

**for**(Object o: session.getNamedQuery("obtenerReservasCliente").setParameter("cliente", cliente).list()){

**if**(o **instanceof** Reserva){

reservas.add((Reserva) o);

}

}

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** reservas;

}

/\*

\* Método para obtener todas las reservas de un inmueble

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**) //si falla hace rollback de la transacción

**public** ArrayList<Reserva> obtenerReservas(Inmueble inmueble) **throws** PersistenciaException {

ArrayList<Reserva> reservas = **new** ArrayList<>();

Session session = getSessionFactory().getCurrentSession();

**try**{

//named query ubicada en entidad reserva

**for**(Object o: session.getNamedQuery("obtenerReservasInmueble").setParameter("inmueble", inmueble).list()){

**if**(o **instanceof** Reserva){

reservas.add((Reserva) o);

}

}

} **catch**(Exception e){

**throw** **new** ConsultaException(e);

}

**return** reservas;

}

### Taskcard 29 Vista alta y listar venta

Código del alta en AltaVentaController.java

/\*\*

\* Acción que se ejecuta al apretar el botón aceptar.

\*

\* Valida que se hayan insertado datos, los carga a la venta y deriva la operación a capa lógica.

\* Si la capa lógica retorna errores, se muestran al usuario.

\*/

@FXML

**public** **void** acceptAction() {

StringBuilder errores = **new** StringBuilder("");

//obtengo y verifico los datos introducidos

Cliente cliente = comboBoxCliente.getValue();

Double importe = **null**;

String medioDePago = textFieldMedioDePago.getText().trim();

**if**(cliente == **null**){

errores.append("Elija un cliente").append("\n");

}

**try**{

importe = Double.*valueOf*(textFieldImporte.getText().trim());

} **catch**(Exception e){

errores.append("Importe incorrecto. Introduzca solo números y un punto para decimales.\n");

}

**if**(medioDePago.isEmpty()){

errores.append("Ingrese un medio de pago").append("\n");

}

**if**(inmueble == **null**){

errores.append("No hay inmueble seleccionado").append("\n");

}

**else** **if**(inmueble.getPropietario() == **null**){

errores.append("No se encuentra el propietario del inmueble").append("\n");

}

//si hay errores se muestra al usuario

**if**(!errores.toString().isEmpty()){

presentador.presentarError("Revise sus campos", errores.toString(), stage);

}

**else**{ //si no hay errores se muestra un Dialog emergente para que el vendedor confirme su contraseña

**boolean** contraseñaCorrecta = showConfirmarContraseñaDialog();

// si la contraseña es correcta se crea la venta

**if**(contraseñaCorrecta){

Venta venta = **new** Venta();

venta.setCliente(cliente);

venta.setFecha(**new** Date(System.*currentTimeMillis*()));

venta.setImporte(importe);

venta.setInmueble(inmueble);

venta.setMedioDePago(medioDePago);

venta.setPropietario(inmueble.getPropietario());

venta.setVendedor(vendedorLogueado);

**try**{ //se delega la operación a capa lógica

ResultadoCrearVenta resultado = coordinador.crearVenta(venta);

**if**(resultado.hayErrores()){

//si hay errores los muestro al usuario

StringBuilder stringErrores = **new** StringBuilder();

**for**(ErrorCrearVenta e: resultado.getErrores()){

**switch**(e) {

**case** ***Cliente\_Igual\_A\_Propietario***:

stringErrores.append("El cliente seleccionado es el actual propietario del inmueble.\n");

**break**;

**case** ***Cliente\_Vacío***:

stringErrores.append("No se ha seleccionado ningún cliente.\n");

**break**;

**case** ***Formato\_Medio\_De\_Pago\_Incorrecto***:

stringErrores.append("Formato de medio de pago incorrecto.\n");

**break**;

**case** ***Importe\_vacío***:

stringErrores.append("No se ha introducido importe.\n");

**break**;

**case** ***Inmueble\_Reservado\_Por\_Otro\_Cliente***:

stringErrores.append("El inmueble está reservado por otro cliente.\n");

**break**;

**case** ***Inmueble\_Vacío***:

stringErrores.append("No se ha seleccionado ningún inmueble.\n");

**break**;

**case** ***Inmueble\_Ya\_Vendido***:

stringErrores.append("El inmueble ya se encuentra vendido.\n");

**break**;

**case** ***Medio\_De\_Pago\_Vacío***:

stringErrores.append("No se ha introducido medio de pago.\n");

**break**;

**case** ***Propietario\_Vacío***:

stringErrores.append("El inmueble no posee propietario.\n");

**break**;

**case** ***Vendedor\_Vacío***:

stringErrores.append("No se ha confirmado el vendedor en esta operación.\n");

**break**;

}

}

presentador.presentarError("Revise sus campos", stringErrores.toString(), stage);

}

**else**{

//si no hay errores muestro una notificación

//y pregunto al usuario si quiere imprimir el documento generado

presentador.presentarToast("Se ha realizado la venta con éxito", stage);

VentanaConfirmacion ventana = presentador.presentarConfirmacion("Venta realizada correctamente", "¿Desea imprimir el documento generado?", stage);

**if**(ventana.acepta()){

//si acepta mando a imprimir

**try**{

impresora.imprimirPDF(venta.getArchivoPDF());

} **catch**(ImprimirPDFException ex){

presentador.presentarExcepcion(ex, stage);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e);

}

}

//vuelvo a la vista de listar inmuebles

cambiarmeAScene(AdministrarInmuebleController.***URLVista***);

}

} **catch**(GestionException e){

presentador.presentarExcepcion(e, stage);

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e);

}

}

}

}

Código del test del alta en AltaVentaControllerTest.java

**protected** Object[] parametersForTestCrearVenta() {

TipoDocumento tipoDoc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

Cliente cliente = **new** Cliente();

cliente.setTipoDocumento(tipoDoc).setNumeroDocumento("11111111");

Cliente clienteProp = **new** Cliente();

clienteProp.setTipoDocumento(tipoDoc).setNumeroDocumento("44444444");

Vendedor vendedor = **new** Vendedor();

vendedor.setTipoDocumento(tipoDoc).setNumeroDocumento("99999999");

Propietario propietario = **new** Propietario();

propietario.setTipoDocumento(tipoDoc).setNumeroDocumento("44444444");

Inmueble inmueble = **new** Inmueble().setDireccion(**new** Direccion()).setPropietario(propietario);

Inmueble inmuebleP = **new** Inmueble().setDireccion(**new** Direccion());

**return** **new** Object[] {

//inmuebleAVender,clienteSeleccionado,vendedorLogueado,importe,medioDePago,resultadoCrearVentaEsperado,llamaAPresentadorVentanasPresentarConfirmacion,usuarioAceptaConfirmacion,llamaACoordinadorImprimirPDF,contraseñaCorrecta,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcionLogica,llamaAPresentadorVentanasPresentarExcepcionImprimir,llamaACrearVenta,excepcionCapaLogica,excepcionAlImprimir

/\*0\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCorrecto***, 1, **true**, 1, **true**, 0, 0, 0, 1, **null**, **null** }, //prueba correcta, usuario acepta imprimir pdf

/\*1\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCorrecto***, 1, **false**, 0, **true**, 0, 0, 0, 1, **null**, **null** }, //prueba correcta, usuario no acepta imprimir pdf

/\*2\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", **null**, 0, **false**, 0, **false**, 0, 0, 0, 0, **null**, **null** }, //Datos correctos pero contraseña incorrecta

/\*3\*/**new** Object[] { inmueble, **null**, vendedor, "1000000","contado", **null**, 0, **false**, 0, **false**, 1, 0, 0, 0, **null**, **null** }, //no se elije cliente

/\*4\*/**new** Object[] { inmueble, cliente, vendedor, "abcd","contado", **null**, 0, **false**, 0, **false**, 1, 0, 0, 0, **null**, **null** }, //importe incorrecto

/\*5\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","", **null**, 0, **false**, 0, **false**, 1, 0, 0, 0, **null**, **null** }, //no se ingresa medio de pago

/\*6\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCrearCliente\_Vacío***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //capa lógica retorna cliente vacío

/\*7\*/**new** Object[] { inmueble, cliente, **null**, "1000000","contado", ***resultadoCrearVendedor\_Vacio***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //capa lógica retorna vendedor vacío

/\*8\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCrearImporte\_Vacio***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //capa lógica retorna importe vacío

/\*9\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCrearMedio\_De\_Pago\_Vacio***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //capa lógica retorna medio de pago vacío

/\*10\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado#%&/", ***resultadoCrearMedio\_De\_Pago\_Incorrecto***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //formato de medio de pago incorrecto

/\*11\*/**new** Object[] { inmuebleP, cliente, vendedor, "1000000","contado", **null**, 0, **false**, 0, **true**, 1, 0, 0, 0, **null**, **null** }, //inmueble sin propietario

/\*12\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCrearPropietario\_Vacío***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //capa lógica retorna inmueble sin propietario

/\*13\*/**new** Object[] { **null**, cliente, vendedor, "1000000","contado", **null**, 0, **false**, 0, **true**, 1, 0, 0, 0, **null**, **null** }, //inmueble vacío

/\*14\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCrearInmueble\_Vacío***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //capa lógica retorna inmueble vacío

/\*15\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCrearInmueble\_Ya\_Vendido***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //el inmueble ya está vendido

/\*16\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCrearInmueble\_Reservado\_Por\_Otro\_Cliente***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //el inmueble ya está reservado

/\*17\*/**new** Object[] { inmueble, clienteProp, vendedor, "1000000","contado", ***resultadoCrearCliente\_Igual\_A\_Propietario***, 0, **false**, 0, **true**, 1, 0, 0, 1, **null**, **null** }, //el cliente ya es el propietario del inmueble

/\*18\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", **null**, 0, **false**, 0, **true**, 0, 1, 0, 1, **new** GenerarPDFException(**new** Throwable()), **null** }, //excepción al generar pdf

/\*19\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", **null**, 0, **false**, 0, **true**, 0, 1, 0, 1, **new** SaveUpdateException(**new** Throwable()), **null** }, //excepción al persistir

/\*20\*/**new** Object[] { inmueble, cliente, vendedor, "1000000","contado", ***resultadoCorrecto***, 1, **true**, 1, **true**, 0, 0, 1, 1, **null**, **new** ImprimirPDFException(**new** Throwable()) }, //excepción al imprimir pdf

};

}

/\*\*

\* Test para el alta venta cuando el usuario presiona el botón aceptar

\*

\* **@param** inmuebleAVender

\* inmueble que se desea vender

\* **@param** clienteSeleccionado

\* cliente seleccionado por el usuario como comprador

\* **@param** vendedorLogueado

\* vendedor logueado en ese momento

\* **@param** importe

\* importe de la venta insertado por el usuario

\* **@param** medioDePago

\* medio de pago de la venta insertado por el usuario

\* **@param** resultadoCrearVentaEsperado

\* resultado que retornará el mock de capa lógica

\* **@param** llamaAPresentadorVentanasPresentarConfirmacion

\* 1 si llama al método presentar confirmación del presentador de ventanas, 0 si no

\* **@param** usuarioAceptaConfirmacion

\* si el usuario acepta la confirmación mostrada

\* **@param** llamaACoordinadorImprimirPDF

\* 1 si llama al método imprimirPDF del coordinador

\* **@param** contraseñaCorrecta

\* si la contraseña ingresada al confirmar contraseña es correcta

\* **@param** llamaAPresentadorVentanasPresentarError

\* 1 si llama al método presentar error del presentador de ventanas, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarExcepcionLogica

\* 1 si llama al método presentar excepción del presentador de ventanas para una excepción de capa lógica, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarExcepcionImprimir

\* 1 si llama al método presentar excepción del presentador de ventanas para una excepción al imprimir, 0 si no

\* **@param** llamaACrearVenta

\* 1 si llama al método crear venta hacia la capa lógica, 0 si no

\* **@param** excepcionCapaLogica

\* excepción de capa lógica que simula ser lanzada

\* **@param** excepcionAlImprimir

\* excepción al imprimir que simula ser lanzada

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCrearVenta(Inmueble inmuebleAVender,

Cliente clienteSeleccionado,

Vendedor vendedorLogueado,

String importe,

String medioDePago,

ResultadoCrearVenta resultadoCrearVentaEsperado,

Integer llamaAPresentadorVentanasPresentarConfirmacion,

**boolean** usuarioAceptaConfirmacion,

Integer llamaACoordinadorImprimirPDF,

**boolean** contraseñaCorrecta,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcionLogica,

Integer llamaAPresentadorVentanasPresentarExcepcionImprimir,

Integer llamaACrearVenta,

Exception excepcionCapaLogica,

Exception excepcionAlImprimir)

**throws** Throwable {

//Se crean los mocks necesarios

CoordinadorJavaFX coordinadorMock = *mock*(CoordinadorJavaFX.**class**);

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VentanaErrorExcepcion ventanaErrorExcepcionMock = *mock*(VentanaErrorExcepcion.**class**);

VentanaErrorExcepcionInesperada ventanaErrorExcepcionInesperadaMock = *mock*(VentanaErrorExcepcionInesperada.**class**);

VentanaConfirmacion ventanaConfirmacionMock = *mock*(VentanaConfirmacion.**class**);

ImpresoraPDF impresoraMock = *mock*(ImpresoraPDF.**class**);

//Se setea lo que deben devolver los mocks cuando son invocados por la clase a probar

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

*when*(presentadorVentanasMock.presentarExcepcion(*any*(), *any*())).thenReturn(ventanaErrorExcepcionMock);

*when*(presentadorVentanasMock.presentarExcepcionInesperada(*any*(), *any*())).thenReturn(ventanaErrorExcepcionInesperadaMock);

*when*(ventanaConfirmacionMock.acepta()).thenReturn(usuarioAceptaConfirmacion);

*when*(presentadorVentanasMock.presentarConfirmacion(*any*(), *any*(), *any*())).thenReturn(ventanaConfirmacionMock);

**if**(excepcionCapaLogica != **null**){

*when*(coordinadorMock.crearVenta(*any*())).thenThrow(excepcionCapaLogica);

} **else** {

*when*(coordinadorMock.crearVenta(*any*())).thenReturn(resultadoCrearVentaEsperado);

}

**if**(excepcionAlImprimir != **null**){

Mockito.*doThrow*(excepcionAlImprimir).when(impresoraMock).imprimirPDF(*any*());

} **else** {

*doNothing*().when(impresoraMock).imprimirPDF(*any*());

}

ArrayList<Cliente> clientes = **new** ArrayList<>();

clientes.add(clienteSeleccionado);

*when*(coordinadorMock.obtenerClientes()).thenReturn(clientes);

//Controlador a probar, se sobreescriben algunos métodos para setear los mocks y setear los datos que ingresaría el usuario en la vista

AltaVentaController altaVentaController = **new** AltaVentaController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

**this**.impresora = impresoraMock;

**super**.inicializar(location, resources);

}

@Override

**public** **void** setTitulo(String titulo) {

}

@Override

**public** **void** setInmueble(Inmueble inmueble) {

**this**.inmueble = inmueble;

}

@Override

**protected** **boolean** showConfirmarContraseñaDialog() {

**return** contraseñaCorrecta;

}

@Override

**public** **void** acceptAction() {

**this**.textFieldImporte.setText(importe);

**this**.textFieldMedioDePago.setText(medioDePago);

**this**.comboBoxCliente.getSelectionModel().select(clienteSeleccionado);

**super**.acceptAction();

};

};

altaVentaController.setInmueble(inmuebleAVender);

altaVentaController.setVendedorLogueado(vendedorLogueado);

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AltaVentaController.***URLVista***, altaVentaController);

Statement test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

//Método a probar

altaVentaController.acceptAction();

//Se hacen las verificaciones pertinentes para comprobar que el controlador se comporte adecuadamente

Mockito.*verify*(coordinadorMock).obtenerClientes();

Mockito.*verify*(coordinadorMock, *times*(llamaACrearVenta)).crearVenta(*any*());

Mockito.*verify*(impresoraMock,*times*(llamaACoordinadorImprimirPDF)).imprimirPDF(*any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Revise sus campos"), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcionLogica)).presentarExcepcion(*eq*(excepcionCapaLogica), *any*());

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarExcepcionImprimir)).presentarExcepcion(*eq*(excepcionAlImprimir), *any*());

Mockito.*verify*(presentadorVentanasMock,*times*(llamaAPresentadorVentanasPresentarConfirmacion)).presentarConfirmacion(*any*(), *any*(), *any*());

Mockito.*verify*(presentadorVentanasMock,*times*(llamaAPresentadorVentanasPresentarConfirmacion)).presentarToast(*any*(), *any*());

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(test, **null**).evaluate();

}

Código del listar en AdministrarVentaController.java

**public** **void** setCliente(Cliente persona) {

Platform.*runLater*(() -> {

**if**(persona != **null**){

labelPersona.setText("Cliente: " + persona.getApellido() + ", " + persona.getNombre());

tablaVentas.getItems().addAll(persona.getVentas());

}

columnaCliente.setVisible(**false**);

tipoPersona = TipoPersona.***Cliente***;

});

}

**public** **void** setPropietario(Propietario persona) {

Platform.*runLater*(() -> {

**if**(persona != **null**){

labelPersona.setText("Propietario: " + persona.getApellido() + ", " + persona.getNombre());

tablaVentas.getItems().addAll(persona.getVentas());

}

columnaPropietario.setVisible(**false**);

tipoPersona = TipoPersona.***Propietario***;

});

}

**public** **void** setVendedor(Vendedor persona) {

Platform.*runLater*(() -> {

**if**(persona != **null**){

labelPersona.setText("Vendedor: " + persona.getApellido() + ", " + persona.getNombre());

tablaVentas.getItems().addAll(persona.getVentas());

}

columnaVendedor.setVisible(**false**);

tipoPersona = TipoPersona.***Vendedor***;

});

}

/\*\*

\* Acción que se ejecuta al presionar el botón ver inmueble

\*/

@FXML

**protected** **void** handleVerInmueble() {

**if**(tablaVentas.getSelectionModel().getSelectedItem() != **null**){

VerBasicosInmuebleController controlador = (VerBasicosInmuebleController) cambiarmeAScene(VerBasicosInmuebleController.***URLVista***);

controlador.setInmueble(tablaVentas.getSelectionModel().getSelectedItem().getInmueble());

**switch**(tipoPersona) {

**case** ***Cliente***:

controlador.setCliente(tablaVentas.getSelectionModel().getSelectedItem().getCliente());

**break**;

**case** ***Propietario***:

controlador.setPropietario(tablaVentas.getSelectionModel().getSelectedItem().getPropietario());

**break**;

**case** ***Vendedor***:

controlador.setVendedor(tablaVentas.getSelectionModel().getSelectedItem().getVendedor());

**break**;

}

}

}

/\*\*

\* Acción que se ejecuta al presionar el botón ver documento

\*/

@FXML

**protected** **void** handleVerDocumento() {

Venta venta = tablaVentas.getSelectionModel().getSelectedItem();

**if**(venta == **null**){

**return**;

}

VerPDFController visorPDF = (VerPDFController) cambiarScene(fondo, VerPDFController.***URLVista***, (Pane) fondo.getChildren().get(0));

visorPDF.cargarPDF(venta.getArchivoPDF());

}

@FXML

**protected** **void** handleSalir() {

**switch**(tipoPersona) {

**case** ***Cliente***:

cambiarmeAScene(AdministrarClienteController.***URLVista***);

**break**;

**case** ***Propietario***:

cambiarmeAScene(AdministrarPropietarioController.***URLVista***);

**break**;

**case** ***Vendedor***:

**if**(vendedorLogueado.getRoot() == **true**) {

cambiarmeAScene(AdministrarVendedorController.***URLVista***);

} **else** {

ModificarVendedorController controlador = (ModificarVendedorController) cambiarmeAScene(ModificarVendedorController.***URLVista***);

controlador.setVendedor(vendedorLogueado);

controlador.mostrarBotonVerVentas();

}

**break**;

}

}

Código del test del listar en AdministrarVentaControllerTest.java

**protected** Object[] parametersForTestRolesDeIngresoYEgreso() {

Cliente cliente = **new** Cliente();

cliente.setNombre("Juan").setApellido("Pérez");

Vendedor vendedor = **new** Vendedor();

vendedor.setNombre("Juan").setApellido("Pérez");

Propietario propietario = **new** Propietario();

propietario.setNombre("Juan").setApellido("Pérez");

Venta venta = **new** Venta();

venta.setCliente(cliente).setPropietario(propietario).setVendedor(vendedor);

venta.setFecha(**new** Date()).setInmueble(**new** Inmueble());

cliente.getVentas().add(venta);

propietario.getVentas().add(venta);

vendedor.getVentas().add(venta);

**return** **new** Object[] {

//cliente,propietario,vendedor,presionaVerInmueble,presionaVerDocumento,llamaAPresentadorVentanasPresentarError,excepcion

/\* 0 \*/**new** Object[] { cliente, **null**, **null**, **true**, **false**, 0, **null** }, // se accede para un cliente, se presiona ver inmueble

/\* 1 \*/**new** Object[] { cliente, **null**, **null**, **false**, **true**, 0, **null** }, // se accede para un cliente, se presiona ver documento

/\* 2 \*/**new** Object[] { cliente, **null**, **null**, **false**, **true**, 1, **new** IOException() }, // se accede para un cliente, se presiona ver documento y ocurre excepcion al abrir pdf

/\* 3 \*/**new** Object[] { **null**, propietario, **null**, **true**, **false**, 0, **null** }, // se accede para un propietario, se presiona ver inmueble

/\* 4 \*/**new** Object[] { **null**, propietario, **null**, **false**, **true**, 0, **null** }, // se accede para un propietario, se presiona ver documento

/\* 5 \*/**new** Object[] { **null**, propietario, **null**, **false**, **true**, 1, **new** IOException() }, // se accede para un propietario, se presiona ver documento y ocurre excepcion al abrir pdf

/\* 6 \*/**new** Object[] { **null**, **null**, vendedor, **true**, **false**, 0, **null** }, // se accede para un vendedor, se presiona ver inmueble

/\* 7 \*/**new** Object[] { **null**, **null**, vendedor, **false**, **true**, 0, **null** }, // se accede para un vendedor, se presiona ver documento

/\* 8 \*/**new** Object[] { **null**, **null**, vendedor, **false**, **true**, 1, **new** IOException() }, // se accede para un vendedor, se presiona ver documento y ocurre excepcion al abrir pdf

};

}

/\*\*

\* Test para probar las transiciones correctas desde y hacia la pantalla y la muestra de las

\* columnas correctas según el rol para el cual se desean conocer las ventas

\*

\* **@param** cliente

\* cliente para el cual se desean listar las ventas

\* **@param** propietario

\* propietario para el cual se desean listar las ventas

\* **@param** vendedor

\* vendedor para el cual se desean listar las ventas

\* **@param** presionaVerInmueble

\* si el usuario presiona ver inmueble

\* **@param** presionaVerDocumento

\* si el usuario presiona ver documento

\* **@param** llamaAPresentadorVentanasPresentarError

\* 1 si llama al método presentar error del presentador de ventanas

\* **@param** excepcion

\* excepción que se simula lanzar desde capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testRolesDeIngresoYEgreso(Cliente cliente,

Propietario propietario,

Vendedor vendedor,

**boolean** presionaVerInmueble,

**boolean** presionaVerDocumento,

Integer llamaAPresentadorVentanasPresentarError,

Exception excepcion) **throws** Throwable {

PresentadorVentanas presentadorVentanasMock = *mock*(PresentadorVentanas.**class**);

VentanaError ventanaErrorMock = *mock*(VentanaError.**class**);

VerBasicosInmuebleController controladorMock = *mock*(VerBasicosInmuebleController.**class**);

*when*(presentadorVentanasMock.presentarError(*any*(), *any*(), *any*())).thenReturn(ventanaErrorMock);

**if**(cliente != **null**){

Mockito.*doNothing*().when(controladorMock).setCliente(cliente);

}

**if**(propietario != **null**){

Mockito.*doNothing*().when(controladorMock).setPropietario(propietario);

}

**if**(vendedor != **null**){

Mockito.*doNothing*().when(controladorMock).setVendedor(vendedor);

}

Mockito.*doNothing*().when(controladorMock).setVendedorLogueado(*any*());

Mockito.*doNothing*().when(controladorMock).setInmueble(*any*());

AdministrarVentaController administrarVentaController = **new** AdministrarVentaController() {

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

**this**.presentador = presentadorVentanasMock;

**super**.inicializar(location, resources);

}

@Override

**public** **void** setTitulo(String titulo) {

}

@Override

**protected** **void** handleVerDocumento() {

tablaVentas.getSelectionModel().selectFirst();

**if**(tablaVentas.getSelectionModel().getSelectedItem() != **null**){

**try**{

//se intenta abrir el pdf

**if**(excepcion != **null**){

**throw** excepcion;

}

} **catch**(Exception ex){

presentador.presentarError("Error", "No se pudo abrir el archivo pdf", stage);

}

}

}

@Override

**protected** **void** handleVerInmueble() {

tablaVentas.getSelectionModel().selectFirst();

**super**.handleVerInmueble();

}

@Override

**protected** OlimpoController cambiarmeAScene(String url) {

controladorMock.setVendedorLogueado(vendedorLogueado);

**return** controladorMock;

}

};

//Los controladores de las vistas deben correrse en un thread de JavaFX

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarVentaController.***URLVista***, administrarVentaController);

Statement correr\_test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

**if**(cliente != **null**){

administrarVentaController.setCliente(cliente);

}

**if**(propietario != **null**){

administrarVentaController.setPropietario(propietario);

}

**if**(vendedor != **null**){

administrarVentaController.setVendedor(vendedor);

}

}

};

Statement evaluar\_test = **new** Statement() {

@Override

**public** **void** evaluate() **throws** Throwable {

**if**(cliente != **null**){

*assertEquals*(**false**, administrarVentaController.columnaCliente.isVisible());

*assertEquals*(**true**, administrarVentaController.columnaPropietario.isVisible());

*assertEquals*(**true**, administrarVentaController.columnaVendedor.isVisible());

}

**if**(propietario != **null**){

*assertEquals*(**true**, administrarVentaController.columnaCliente.isVisible());

*assertEquals*(**false**, administrarVentaController.columnaPropietario.isVisible());

*assertEquals*(**true**, administrarVentaController.columnaVendedor.isVisible());

}

**if**(vendedor != **null**){

*assertEquals*(**true**, administrarVentaController.columnaCliente.isVisible());

*assertEquals*(**true**, administrarVentaController.columnaPropietario.isVisible());

*assertEquals*(**false**, administrarVentaController.columnaVendedor.isVisible());

}

**if**(presionaVerDocumento){

administrarVentaController.handleVerDocumento();

}

**if**(presionaVerInmueble){

administrarVentaController.handleVerInmueble();

Mockito.*verify*(controladorMock).setInmueble(*any*());

Mockito.*verify*(controladorMock).setVendedorLogueado(*any*());

**if**(cliente != **null**){

Mockito.*verify*(controladorMock).setCliente(cliente);

}

**if**(propietario != **null**){

Mockito.*verify*(controladorMock).setPropietario(propietario);

}

**if**(vendedor != **null**){

Mockito.*verify*(controladorMock).setVendedor(vendedor);

}

}

Mockito.*verify*(presentadorVentanasMock, *times*(llamaAPresentadorVentanasPresentarError)).presentarError(*eq*("Error"), *eq*("No se pudo abrir el archivo pdf"), *any*());

}

};

//Se corre el test en el hilo de JavaFX

corredorTestEnJavaFXThread.apply(correr\_test, **null**).evaluate();

corredorTestEnJavaFXThread.apply(evaluar\_test, **null**).evaluate();

}

### Taskcard 30 Lógica alta venta y generar PDF

Código de la lógica en GestorVenta.java

/\*\*

\* Método para crear una venta. Primero se validan las reglas de negocio y luego se persiste.

\* Pertenece a la taskcard 30 de la iteración 2 y a la historia 8

\*

\* **@param** venta

\* venta que se quiere crear

\* **@return** resultado de la operación

\* **@throws** PersistenciaException

\* si falló al persistir

\*/

**public** ResultadoCrearVenta crearVenta(Venta venta) **throws** PersistenciaException, GestionException {

ArrayList<ErrorCrearVenta> errores = **new** ArrayList<>();

//se validan los datos

**if**(venta.getCliente() == **null**){

errores.add(ErrorCrearVenta.***Cliente\_Vacío***);

}

**if**(venta.getVendedor() == **null**){

errores.add(ErrorCrearVenta.***Vendedor\_Vacío***);

}

**if**(venta.getInmueble() == **null**){

errores.add(ErrorCrearVenta.***Inmueble\_Vacío***);

}

**if**(venta.getPropietario() == **null**){

errores.add(ErrorCrearVenta.***Propietario\_Vacío***);

}

**if**(venta.getImporte() == **null**){

errores.add(ErrorCrearVenta.***Importe\_vacío***);

}

**if**(venta.getMedioDePago().isEmpty() || venta.getMedioDePago() == **null**){

errores.add(ErrorCrearVenta.***Medio\_De\_Pago\_Vacío***);

} **else** **if**(!validador.validarMedioDePago(venta.getMedioDePago())) {

errores.add(ErrorCrearVenta.***Formato\_Medio\_De\_Pago\_Incorrecto***);

}

//verifico si el cliente tiene mismo tipo y número de documento que el propietario. Entonces ya es el propietario del inmueble

**if**(venta.getCliente() != **null** && venta.getPropietario() != **null** && venta.getCliente().getTipoDocumento().equals(venta.getPropietario().getTipoDocumento()) && venta.getCliente().getNumeroDocumento().equals(venta.getPropietario().getNumeroDocumento())) {

errores.add(ErrorCrearVenta.***Cliente\_Igual\_A\_Propietario***);

}

//verifico si el inmueble ya está vendido

**if**(venta.getInmueble() != **null** && venta.getInmueble().getEstadoInmueble().getEstado().equals(EstadoInmuebleStr.***VENDIDO***)) {

errores.add(ErrorCrearVenta.***Inmueble\_Ya\_Vendido***);

}

//verifico si el inmueble está reserado por otro cliente que no sea el que lo quiere comprar

Date fechaHoy = **null**;

**if**(venta.getInmueble() != **null**) {

fechaHoy = **new** Date(System.*currentTimeMillis*());

**for**(Reserva r: venta.getInmueble().getReservas()) {

**if** (fechaHoy.after(r.getFechaInicio()) && fechaHoy.before(r.getFechaFin()) && !r.getCliente().equals(venta.getCliente())) {

errores.add(ErrorCrearVenta.***Inmueble\_Reservado\_Por\_Otro\_Cliente***);

**break**;

}

}

}

//si no hay errores marco al inmueble como vendido, seteo fecha, pdf y mando a persistir

**if**(errores.isEmpty()){

ArrayList<EstadoInmueble> estadosInm = gestorDatos.obtenerEstadosInmueble();

**for**(EstadoInmueble ei: estadosInm){

**if**(ei.getEstado().equals(EstadoInmuebleStr.***VENDIDO***)){

venta.getInmueble().setEstadoInmueble(ei);

**break**;

}

}

gestorInmueble.modificarInmueble(venta.getInmueble());

venta.setFecha(fechaHoy);

venta.setArchivoPDF(gestorPDF.generarPDF(venta));

persistidorVenta.guardarVenta(venta);

}

**return** **new** ResultadoCrearVenta(errores.toArray(**new** ErrorCrearVenta[0]));

}

Código del test de la lógica en GestorVentaTest.java

**protected** Object[] parametersForTestCrearVenta() {

TipoDocumento tipoDoc = **new** TipoDocumento(TipoDocumentoStr.***DNI***);

Propietario p = **new** Propietario();

p.setTipoDocumento(tipoDoc).setNumeroDocumento("12345678");

Vendedor v = **new** Vendedor();

Cliente c = **new** Cliente();

c.setTipoDocumento(tipoDoc).setNumeroDocumento("87654321");

Cliente cp = **new** Cliente();

cp.setTipoDocumento(tipoDoc).setNumeroDocumento("12345678");

Inmueble isi = **new** Inmueble();

isi.setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***VENDIDO***));

Reserva rOtro = **new** Reserva();

rOtro.setCliente(cp).setFechaInicio(**new** Date(System.*currentTimeMillis*() - 1000000000)).setFechaFin(**new** Date(System.*currentTimeMillis*() + 100000000));

Reserva rMismo = **new** Reserva();

rMismo.setCliente(c).setFechaInicio(**new** Date(System.*currentTimeMillis*() - 1000000000)).setFechaFin(**new** Date(System.*currentTimeMillis*() + 100000000));

HashSet<Reserva> hROtro = **new** HashSet<>();

hROtro.add(rOtro);

HashSet<Reserva> hRMismo = **new** HashSet<>();

hROtro.add(rMismo);

Inmueble inoMismo = **new** Inmueble();

inoMismo.setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***)).setReservas(hRMismo);

Inmueble inoOtro = **new** Inmueble();

inoOtro.setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***)).setReservas(hROtro);

//Parámetros de JUnitParams

**return** **new** Object[] {

//venta,resultadoEsperado,resultadoCorrecto,medioDePagoValido,excepcionPersistencia,excepcionPDF

/\* 0 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("contado").setPropietario(p).setVendedor(v), ***resultadoCorrecto***, **true**, **true**, **null**, **null** }, //resultado correcto

/\* 1 \*/**new** Object[] { **new** Venta().setCliente(**null**).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("contado").setPropietario(p).setVendedor(v), ***resultadoCrearCliente\_Vacío***, **false**, **true**, **null**, **null** }, //la venta no tiene cliente

/\* 2 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("contado").setPropietario(**null**).setVendedor(v), ***resultadoCrearPropietario\_Vacío***, **false**, **true**, **null**, **null** }, //la venta no tiene propietario

/\* 3 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(**null**).setMedioDePago("contado").setPropietario(p).setVendedor(v), ***resultadoCrearInmueble\_Vacío***, **false**, **true**, **null**, **null** }, //la venta no tiene inmueble

/\* 4 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("contado").setPropietario(p).setVendedor(**null**), ***resultadoCrearVendedor\_Vacio***, **false**, **true**, **null**, **null** }, //la venta no tiene vendedor

/\* 5 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(**null**).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("contado").setPropietario(p).setVendedor(v), ***resultadoCrearImporte\_Vacio***, **false**, **true**, **null**, **null** }, //la venta no tiene importe

/\* 6 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago(**null**).setPropietario(p).setVendedor(v), ***resultadoCrearMedio\_De\_Pago\_Vacio***, **false**, **true**, **null**, **null** }, //la venta no tiene medio de pago

/\* 7 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("#$%&").setPropietario(p).setVendedor(v), ***resultadoCrearMedio\_De\_Pago\_Incorrecto***, **false**, **false**, **null**, **null** }, //formato de medio de pago incorrecto

/\* 8 \*/**new** Object[] { **new** Venta().setCliente(cp).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("contado").setPropietario(p).setVendedor(v), ***resultadoCrearCliente\_Igual\_A\_Propietario***, **false**, **true**, **null**, **null** }, //el cliente ya es el propietario

/\* 9 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(isi).setMedioDePago("contado").setPropietario(p).setVendedor(v), ***resultadoCrearInmueble\_Ya\_Vendido***, **false**, **true**, **null**, **null** }, //el inmueble ya se encuentra vendido

/\* 10 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(inoOtro).setMedioDePago("contado").setPropietario(p).setVendedor(v), ***resultadoCrearInmueble\_Reservado\_Por\_Otro\_Cliente***, **false**, **true**, **null**, **null** }, //el inmueble ya se encuentra reservado por otro cliente

/\* 11 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(inoMismo).setMedioDePago("contado").setPropietario(p).setVendedor(v), ***resultadoCorrecto***, **true**, **true**, **null**, **null** }, //resultado correcto, el inmueble está reservado pero por el mismo cliente que desea comprarlo

/\* 12 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("contado").setPropietario(p).setVendedor(v), **null**, **false**, **true**, **new** SaveUpdateException(**new** Throwable()), **null** }, //excpecion al persistir

/\* 13 \*/**new** Object[] { **new** Venta().setCliente(c).setImporte(10.0).setInmueble(**new** Inmueble().setEstadoInmueble(**new** EstadoInmueble(EstadoInmuebleStr.***NO\_VENDIDO***))).setMedioDePago("contado").setPropietario(p).setVendedor(v), **null**, **false**, **true**, **null**, **new** GenerarPDFException(**new** Throwable()) }, //excpecion al generar PDF

};

}

/\*\*

\* Prueba el método crearVenta, el cual corresponde con la taskcard 30 de la iteración 2 y a la historia 8

\*

\* **@param** venta

\* venta que se quiere crear

\* **@param** resultadoEsperado

\* resultado que se espera retorne el método a probar

\* **@param** resultadoCorrecto

\* si el resultado es correcto y deberían efectuarse los seteos correspondientes y posterior guardado

\* **@param** medioDePagoValido

\* valor que retorna el mock validador al validad medio de pago

\* **@param** excepcionPersistencia

\* excepción originada en capa de persistencia que se simula ser lanzada

\* **@param** excepcionPDF

\* excepción al generar PFD que se simula ser lanzada

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCrearVenta(Venta venta,

ResultadoCrearVenta resultadoEsperado,

**boolean** resultadoCorrecto,

**boolean** medioDePagoValido,

Exception excepcionPersistencia,

Exception excepcionPDF) **throws** Throwable {

GestorInmueble gestorInmuebleMock = *mock*(GestorInmueble.**class**);

GestorDatos gestorDatosMock = *mock*(GestorDatos.**class**);

GestorPDF gestorPDFMock = *mock*(GestorPDF.**class**);

VentaService persistidorVentaMock = *mock*(VentaService.**class**);

ValidadorFormato validadorMock = *mock*(ValidadorFormato.**class**);

**if**(excepcionPDF == **null**){

Mockito.*when*(gestorPDFMock.generarPDF(venta)).thenReturn(**new** PDF());

}

**else**{

Mockito.*when*(gestorPDFMock.generarPDF(venta)).thenThrow(excepcionPDF);

}

**if**(excepcionPersistencia == **null**){

Mockito.*doNothing*().when(persistidorVentaMock).guardarVenta(venta);

}

**else**{

Mockito.*doThrow*(excepcionPersistencia).when(persistidorVentaMock).guardarVenta(venta);

}

Mockito.*when*(validadorMock.validarMedioDePago(venta.getMedioDePago())).thenReturn(medioDePagoValido);

Mockito.*when*(gestorInmuebleMock.modificarInmueble(venta.getInmueble())).thenReturn(**null**);

ArrayList<EstadoInmueble> estadosInm = **new** ArrayList<>();

estadosInm.add(**new** EstadoInmueble(EstadoInmuebleStr.***VENDIDO***));

Mockito.*when*(gestorDatosMock.obtenerEstadosInmueble()).thenReturn(estadosInm);

GestorVenta gestorVenta = **new** GestorVenta() {

{

**this**.gestorInmueble = gestorInmuebleMock;

**this**.persistidorVenta = persistidorVentaMock;

**this**.gestorDatos = gestorDatosMock;

**this**.gestorPDF = gestorPDFMock;

**this**.validador = validadorMock;

}

};

ResultadoCrearVenta resultadoCrearVenta;

**try**{

resultadoCrearVenta = gestorVenta.crearVenta(venta);

*assertEquals*(resultadoEsperado.getErrores(), resultadoCrearVenta.getErrores());

} **catch**(Exception e){

**if**(excepcionPDF != **null**){

*assertEquals*(excepcionPDF.getClass(), e.getClass());

}

**if**(excepcionPersistencia != **null**){

*assertEquals*(excepcionPersistencia.getClass(), e.getClass());

}

}

**if**(venta.getMedioDePago() != **null**) {

Mockito.*verify*(validadorMock).validarMedioDePago(venta.getMedioDePago());

}

**if**(resultadoCorrecto){

Mockito.*verify*(gestorDatosMock).obtenerEstadosInmueble();

Mockito.*verify*(gestorPDFMock).generarPDF(venta);

Mockito.*verify*(gestorInmuebleMock).modificarInmueble(venta.getInmueble());

Mockito.*verify*(persistidorVentaMock).guardarVenta(venta);

*assertEquals*(EstadoInmuebleStr.***VENDIDO***, venta.getInmueble().getEstadoInmueble().getEstado());

*assertEquals*(PDF.**class**, venta.getArchivoPDF().getClass());

*assertEquals*(Date.**class**, venta.getFecha().getClass());

}

}

Código de generar PDF en GestorPDF.java

/\*\*

\* Método para crear un PDF de una venta a partir de los datos de una Venta.

\* Pertenece a la taskcard 30 de la iteración 2 y a la historia 8

\*

\* **@param** venta

\* datos que se utilizaran para generar el PDF de una venta

\* **@return** venta en PDF

\*/

**public** PDF generarPDF(Venta venta) **throws** GestionException {

**try**{

//se carga el fxml

pdf = **null**;

FXMLLoader loader = **new** FXMLLoader();

loader.setLocation(getClass().getResource(***URLDocumentoVenta***));

Pane documentoVenta = (Pane) loader.load();

FutureTask<Throwable> future = **new** FutureTask<>(() -> {

**try**{

//se setean los campos del documento con los datos de la venta

Label label = (Label) documentoVenta.lookup("#lblNombreComprador");

label.setText(formateador.nombrePropio(venta.getCliente().getNombre()));

label = (Label) documentoVenta.lookup("#lblApellidoComprador");

label.setText(formateador.nombrePropio(venta.getCliente().getApellido()));

label = (Label) documentoVenta.lookup("#lblDocumentoComprador");

label.setText(venta.getCliente().getTipoDocumento() + " - " + venta.getCliente().getNumeroDocumento());

label = (Label) documentoVenta.lookup("#lblNombrePropietario");

label.setText(formateador.nombrePropio(venta.getInmueble().getPropietario().getNombre()));

label = (Label) documentoVenta.lookup("#lblApellidoPropietario");

label.setText(formateador.nombrePropio(venta.getInmueble().getPropietario().getApellido()));

label = (Label) documentoVenta.lookup("#lblDocumentoPropietario");

label.setText(venta.getPropietario().getTipoDocumento() + " - " + venta.getPropietario().getNumeroDocumento());

label = (Label) documentoVenta.lookup("#lblCodigoInmueble");

label.setText(Integer.*toString*(venta.getInmueble().getId()));

label = (Label) documentoVenta.lookup("#lblTipoInmueble");

label.setText(venta.getInmueble().getTipo().getTipo().toString());

label = (Label) documentoVenta.lookup("#lblLocalidadInmueble");

label.setText(venta.getInmueble().getDireccion().getLocalidad().toString());

label = (Label) documentoVenta.lookup("#lblBarrioInmueble");

label.setText(venta.getInmueble().getDireccion().getBarrio().toString());

label = (Label) documentoVenta.lookup("#lblCalleInmueble");

label.setText(venta.getInmueble().getDireccion().getCalle().toString());

label = (Label) documentoVenta.lookup("#lblAlturaInmueble");

label.setText(venta.getInmueble().getDireccion().getNumero());

label = (Label) documentoVenta.lookup("#lblPisoInmueble");

label.setText(venta.getInmueble().getDireccion().getPiso());

label = (Label) documentoVenta.lookup("#lblDepartamentoInmueble");

label.setText(venta.getInmueble().getDireccion().getDepartamento());

label = (Label) documentoVenta.lookup("#lblOtrosInmueble");

label.setText(venta.getInmueble().getDireccion().getOtros());

label = (Label) documentoVenta.lookup("#lblImporte");

label.setText(String.*format*("$ %10.2f", venta.getImporte()));

label = (Label) documentoVenta.lookup("#lblMedioDePago");

label.setText(venta.getMedioDePago());

label = (Label) documentoVenta.lookup("#lblFechaVenta");

label.setText(conversorFechas.diaMesYAnioToString(venta.getFecha()));

label = (Label) documentoVenta.lookup("#lblHoraGenerado");

Date ahora = **new** Date();

label.setText(String.*format*(label.getText(), conversorFechas.horaYMinutosToString(ahora), conversorFechas.diaMesYAnioToString(ahora)));

//genera el archivo

pdf = generarPDF(documentoVenta);

} **catch**(Throwable e){

**return** e; //si algo falla

}

**return** **null**; //si no falla nada

});

//se asegura de que se corra en el hilo de javaFX

**if**(!Platform.*isFxApplicationThread*()){

Platform.*runLater*(future);

}

**else**{

future.run();

}

Throwable excepcion = future.get();

//si hubo error se lanza excepción

**if**(excepcion != **null**){

**throw** excepcion;

}

**if**(pdf == **null**){

**throw** **new** NullPointerException("Error al generar PDF");

}

} **catch**(Throwable e){

**throw** **new** GenerarPDFException(e);

}

**return** pdf;

}

Código del test de generar PDF en GestorPDFTest.java

/\*\*

\* Prueba el método generarPDF(Venta venta), el cual corresponde con la taskcard 30 de la iteración 2 y a la historia 8

\* El test es en parte manual ya que genera un archivo pdf que debe comprobarse si es correcto manualmente.

\*

\* **@throws** Exception

\*/

@Test

**public** **void** testGenerarPDFVenta() **throws** Exception {

**new** ControladorTest(LoginController.***URLVista***, **new** LoginController() {

@Override

**protected** **void** setTitulo(String titulo) {

}

@Override

**public** **void** inicializar(URL location, ResourceBundle resources) {

}

});

GestorPDF gestor = **new** GestorPDF() {

{

formateador = **new** FormateadorString();

conversorFechas = **new** ConversorFechas();

}

};

Cliente cliente = **new** Cliente().setNombre("Jaquelina")

.setApellido("Acosta")

.setTipoDocumento(**new** TipoDocumento().setTipo(TipoDocumentoStr.***DNI***))

.setNumeroDocumento("36696969");

Propietario propietario = **new** Propietario().setNombre("Ignacio")

.setApellido("Falchini")

.setTipoDocumento(**new** TipoDocumento().setTipo(TipoDocumentoStr.***DNI***))

.setNumeroDocumento("12345678");

Localidad localidad = **new** Localidad().setProvincia(**new** Provincia().setPais(**new** Pais())).setNombre("Federal");

Direccion direccion = **new** Direccion().setCalle(**new** Calle().setLocalidad(localidad).setNombre("Donovan")).setLocalidad(localidad).setBarrio(**new** Barrio().setLocalidad(localidad).setNombre("Centro"))

.setNumero("635")

.setPiso("6")

.setDepartamento("B")

.setOtros("otros");

Inmueble inmueble = **new** Inmueble() {

@Override

**public** Integer getId() {

**return** 12345;

};

}.setTipo(**new** TipoInmueble(TipoInmuebleStr.***DEPARTAMENTO***))

.setDireccion(direccion)

.setPropietario(propietario);

Date fechahoy = **new** Date();

Venta venta = **new** Venta().setCliente(cliente)

.setPropietario(propietario)

.setInmueble(inmueble)

.setFecha(fechahoy)

.setImporte(1000000.0)

.setMedioDePago("contado");

PDF pdf = gestor.generarPDF(venta);

FileOutputStream fos = **new** FileOutputStream("testVenta.pdf");

fos.write(pdf.getArchivo());

fos.close();

}

### Taskcard 31 Lógica imprimir venta

Código de imprimir PDF (en general) en ImpresoraPDF.java

/\*\*

\* Método que manda a imprimir un PDF

\*

\* **@param** p

\* PDF a imprimir

\* **@throws** ImprimirPDFException

\* si falla al imprimir

\*/

**public** **void** imprimirPDF(PDF p) **throws** ImprimirPDFException {

**try**{

PDDocument document = PDDocument.*load*(p.getArchivo());

PrinterJob printJob = PrinterJob.*getPrinterJob*();

**if**(printJob.printDialog()){

printJob.setPageable(**new** PDFPageable(document));

printJob.print();

document.close();

}

} **catch**(Exception ex){

**throw** **new** ImprimirPDFException(ex);

}

}

### Taskcard 32 Persistidor y entidad venta

Código del archivo Venta.java

/\*\*

\* Entidad que modela una venta

\* Pertenece a la taskcard 32 de la iteración 2

\*/

**public** **class** Venta {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** Integer id; //ID

@Column(name = "importe", nullable = **false**)

**private** Double importe;

@Column(name = "medio\_de\_pago", nullable = **false**)

**private** String medioDePago;

@Column(name = "fecha", nullable = **false**)

**private** Date fecha;

@OneToOne(fetch = FetchType.***EAGER***, cascade = CascadeType.***ALL***, orphanRemoval = **true**, optional = **false**)

@JoinColumn(name = "idarchivo", foreignKey = @ForeignKey(name = "venta\_idarchivo\_fk"), nullable = **false**)

**private** PDF archivoPDF;

//Relaciones

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idvendedor", referencedColumnName = "id", foreignKey = @ForeignKey(name = "venta\_idvendedor\_fk"), nullable = **false**)

**private** Vendedor vendedor;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idcliente", referencedColumnName = "id", foreignKey = @ForeignKey(name = "venta\_idcliente\_fk"), nullable = **false**)

**private** Cliente cliente;

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idpropietario", referencedColumnName = "id", foreignKey = @ForeignKey(name = "venta\_idpropietario\_fk"), nullable = **false**)

**private** Propietario propietario;

@OneToOne(fetch = FetchType.***EAGER***, optional = **false**)

@JoinColumn(name = "idinmueble", referencedColumnName = "id", foreignKey = @ForeignKey(name = "venta\_idinmueble\_fk"), nullable = **false**)

**private** Inmueble inmueble;

Código del archivo VentaServiceImpl.java

/\*

\* Método para guardar en la base de datos una venta

\*/

@Override

@Transactional(rollbackFor = PersistenciaException.**class**)//si falla hace rollback de la transacción

**public** **void** guardarVenta(Venta venta) **throws** PersistenciaException {

Session session = getSessionFactory().getCurrentSession();

**try**{

session.save(venta);

} **catch**(Exception e){

**throw** **new** SaveUpdateException(e);

}

}

### Taskcard 33 Agregar EstadoInmueble a inmueble

Se agregó el siguiente código a la entidad Inmueble en Inmueble.java

@ManyToOne(fetch = FetchType.***EAGER***)

@JoinColumn(name = "idestadoinmueble", referencedColumnName = "id", foreignKey = @ForeignKey(name = "inmueble\_idestadoinmueble\_fk"), nullable = **false**)

**private** EstadoInmueble estadoInmueble; //Establece si el inmueble fue vendido o no

Código de Estado Inmueble en EstadoInmueble.java

/\*\*

\* Entidad que modela los estados por los cuales pueden pasar un inmueble.

\* Pertenece a la taskcard 33 de la iteración 2 y a la historia de usuario 8

\*/

@NamedQuery(name = "obtenerEstadosInmueble", query = "SELECT e FROM EstadoInmueble e")

@Entity

@Table(name = "estadoinmueble")

**public** **class** EstadoInmueble {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id; //ID

@Column(name = "estado", length = 20, unique = **true**)

@Enumerated(EnumType.***STRING***)

**private** EstadoInmuebleStr estado;

//getters, setters, constructores y otros métodos

(…)