# 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 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 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 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 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 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 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 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 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***);

}

(…)