# Código Fuente y Pruebas de Unidad

## Iteración 1:

### ABM Vendedor

#### Taskcard 1 Vista y lógica del Login

#### Taskcard 4 Vista principal para la administración

#### Taskcard 11 Entidad inmueble

#### 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

#### Taskcard 16 Entidad cliente

#### Taskcard 19 Persistidor cliente