# Código Fuente y Pruebas de Unidad

## Iteración 1:

#### Taskcard 1 Vista alta, modificar y baja

Código del archivo AdministrarVendedorController.java

/\*\*

\* Acción que se ejecuta al presionar el botón agregar

\* Se pasa a la pantalla alta vendedor

\*/

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

//Se llama a la pantalla alta vendedor

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

}

/\*\*

\* Acción que se ejecuta al presionar el botón modificar

\* Se pasa a la pantalla modificar vendedor

\*/

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

//Se comprueba que efectivamente haya un vendedor seleccionado

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

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

**return**;

}

//Se llama a la pantalla de modificar vendedor

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

controlador.setVendedor(vendedor);

}

/\*\*

\* Acción que se ejecuta al presionar el botón eliminar

\* Se muestra una ventana emergente para confirmar la operación

\*/

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

//Se comprueba que efectivamente haya un vendedor seleccionado

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

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

**return**;

}

//Se pregunta al usuario si quiere eliminar el vendedor

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

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

**try**{

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

ResultadoEliminarVendedor resultado = coordinador.eliminarVendedor(vendedor);

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

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

//Procesamiento de errores de la lógica

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

**switch**(err) {

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

}

}

//Se muestran los errores devueltos por la vista

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

}

**else**{

//Si no hubo errores

//Se quita el inmueble de la vista

tablaVendedores.getItems().remove(vendedor);

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

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

}

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

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e,stage);

}

}

}

Prueba de unidad del archivo AdministrarInmuebleControllerTest.java

/\*\*

\* Prueba el método agregarVendedor()

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

\*

\* **@throws** Throwable

\*/

@Test

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

//Se crean los mocks necesarios

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

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

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

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

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

//Controlador a probar;

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

@Override

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

**this**.coordinador = coordinadorMock;

**this**.setScenographyChanger(scenographyChangerMock);

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

}

@Override

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

}

};

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

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

administrarVendedorController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

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

@Override

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

//Método a probar

administrarVendedorController.agregarAction(**null**);

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

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

}

};

//Se corre el test en el hilo de JavaFX

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

;

}

//Casos de prueba

//listaVendedores, llamaAModificar

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

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

/\*\*

\* Prueba el método Modificar Vendedor()

\* **@param** listaVendedores

\* vendedores inicialmente en la tabla de vendedores

\* **@param** llamaAModificar

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

\* **@throws** Throwable

\*/

@Test

@Parameters

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

//Se crean los mocks necesarios

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

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

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

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

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

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

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

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

@Override

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

**this**.coordinador = coordinadorMock;

**this**.setScenographyChanger(scenographyChangerMock);

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

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

}

@Override

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

}

};

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

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

administrarVendedorController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

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

@Override

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

//Método a probar

administrarVendedorController.modificarAction(**null**);

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

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

}

};

//Se corre el test en el hilo de JavaFX

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

}

//Casos de prueba

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

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

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

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

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

/\*\*

\* Prueba el método eliminar Vendedor()

\* **@param** listaVendedores

\* vendedores inicialmente en la tabla de vendedores

\* **@param** llamaAPresentadorVentanasPresentarConfirmacion

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

\* **@param** aceptarVentanaConfirmacion

\* indica si el usuario acepta la ventana de confirmación

\* **@param** resultadoEliminarVendedorEsperado

\* resultado devuelto por la capa lógica

\* **@param** llamaAPresentadorVentanasPresentarError

\* indica si se debe presentar un error

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* indica si se debe presentar una excepcion

\* **@param** excepcion

\* excepcion lanzada por la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

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

Integer llamaAPresentadorVentanasPresentarConfirmacion,

Boolean aceptarVentanaConfirmacion,

ResultadoEliminarVendedor resultadoEliminarVendedorEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

PersistenciaException excepcion)

**throws** Throwable {

//Se crean los mocks necesarios

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

}

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

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

@Override

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

**this**.coordinador = coordinadorMock;

**this**.setScenographyChanger(scenographyChangerMock);

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

**this**.presentador = presentadorVentanasMock;

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

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

}

@Override

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

}

};

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

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

administrarVendedorController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

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

@Override

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

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

//Método a probar

administrarVendedorController.eliminarAction(**null**);

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

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

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

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

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

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

}

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

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

*assertNotEquals*(cantidadVendedoresAntesDeEliminar, cantidadVendedoresDespuesDeEliminar);

}

}

};

//Se corre el test en el hilo de JavaFX

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

}

Código del archivo AltaVendedorController.java

/\*\*

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

\*

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

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

\*/

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

//Se toman los datos ingresados por el usuario

cargarDatos();

//Se validan los datos ingresados por el usuario

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

validarVistaAltaVendedor(error);

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

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

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

}

**else**{

crearVendedor();

}

}

/\*\*

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

\*/

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

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

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

vendedor.setNombre(nombre)

.setApellido(apellido)

.setNumeroDocumento(numeroDocumento)

.setTipoDocumento(tipoDoc)

.setSalt(encriptador.generarSal())

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

ResultadoCrearVendedor resultadoCrearVendedor = **null**;

**try**{

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

resultadoCrearVendedor = coordinador.crearVendedor(vendedor);

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

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

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

parsearErroresLogica(error, listaErrores);

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

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

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

}

**else**{

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

salir();

}

//Se manejan las excepciones que puede devolver la lógica

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

} **catch**(EntidadExistenteConEstadoBajaException e){

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

manejarVendedorExistenteBaja(vendedor);

} **catch**(Exception e){

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

}

}

/\*\*

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

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

\*

\* **@param** vendedor

\*/

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

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

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

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

**try**{

vendedor = coordinador.obtenerVendedor(vendedor);

} **catch**(PersistenciaException e1){

presentador.presentarExcepcion(e1, stage);

}

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

controlador.setVendedor(vendedor);

controlador.setAltaVendedor();

}

}

/\*\*

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

\*

\* **@param** error

\* Texto a mostrar al usuario

\* **@param** listaErrores

\* Lista de errores de la capa lógica

\*/

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

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

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

}

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

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

}

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

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

}

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

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

}

}

/\*\*

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

\*

\* **@param** error

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

\*/

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

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

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

}

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

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

}

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

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

}

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

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

}

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

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

}

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

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

}

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

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

}

}

/\*\*

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

\*/

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

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

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

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

password1 = passwordFieldContraseña.getText();

password2 = passwordFieldRepiteContraseña.getText();

tipoDoc = comboBoxTipoDocumento.getValue();

}

Prueba de unidad del archivo AltaVendedorControllerTest.java

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

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

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

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

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

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

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

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

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

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

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

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

/\*\*

\* **@param** nombre

\* nombre del vendedor a crear

\* **@param** apellido

\* apellido del vendedor a crear

\* **@param** tipoDocumento

\* tipoDocumento del vendedor a crear

\* **@param** numeroDocumento

\* numeroDocumento del vendedor a crear

\* **@param** contraseña

\* contraseña ingresada

\* **@param** contraseña2

\* segunda contraseña ingresada

\* **@param** resultadoCrearVendedorEsperado

\* apellido del vendedor a crear

\* **@param** llamaAPresentadorVentanasPresentarError

\* indica si se debe presentar una ventana de error

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* indica si se debe presentar una ventana de excepción

\* **@param** llamaAPresentadorVentanasPresentarExcepcionInesperada

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

\* **@param** llamaACrearVendedor

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

\* **@param** excepcion

\* excepción devuelta por la lógica

\* **@param** aceptarVentanaConfirmacion

\* indica si el usuario acepta la ventana de confirmación

\* **@param** llamaACambiarScene

\* indica si debe llamar al cambio de pantalla

\* **@throws** Throwable

\*/

@Test

@Parameters

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

//Se crean los mocks necesarios

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Vendedor vendedor = **new** Vendedor()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setPassword(contraseña);

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

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

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

}

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

tipos.add(tipoDocumento);

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

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

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

@Override

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

**this**.coordinador = coordinadorMock;

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

**this**.presentador = presentadorVentanasMock;

setScenographyChanger(scenographyChangerMock);

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

}

@Override

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

**this**.coordinador = coordinadorMock;

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

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

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

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

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

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

**super**.acceptAction();

};

@Override

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

}

};

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

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

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

@Override

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

//Método a probar

altaVendedorController.acceptAction();

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

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

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

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

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

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

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

}

};

//Se corre el test en el hilo de JavaFX

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

}

Código del archivo ModificarVendedorController.java

/\*\*

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

\*

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

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

\*/

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

//Se toman los datos de la vista

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

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

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

String passwordAntigua = passwordFieldContraseñaAntigua.getText();

String password1 = passwordFieldContraseñaNueva.getText();

String password2 = passwordFieldRepiteContraseña.getText();

TipoDocumento tipoDoc = comboBoxTipoDocumento.getValue();

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

//Se validan los campos

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

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

}

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

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

}

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

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

}

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

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

}

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

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

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

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

}

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

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

}

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

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

}

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

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

}

}

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

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

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

}

**else**{

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

vendedor.setNombre(nombre)

.setApellido(apellido)

.setNumeroDocumento(numeroDocumento)

.setTipoDocumento(tipoDoc);

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

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

}

ResultadoModificarVendedor resultadoModificarVendedor = **null**;

**try**{

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

resultadoModificarVendedor = coordinador.modificarVendedor(vendedor);

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

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

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

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

}

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

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

}

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

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

}

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

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

}

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

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

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

}

**else**{

/\*

\* primero sale, y después presenta el Toast

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

\*/

salir();

**if**(esAltaNuevamente){

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

}

**else**{

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

}

}

} **catch**(PersistenciaException e){

presentador.presentarExcepcion(e, stage);

} **catch**(Exception e){

presentador.presentarExcepcionInesperada(e, stage);

}

}

}

Prueba de unidad del archivo ModificarVendedorControllerTest.java

//Casos de prueba

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

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

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

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

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

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

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

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

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

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

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

/\*\*

\* **@param** nombre

\* nombre del vendedor a modificar

\* **@param** apellido

\* apellido del vendedor a modificar

\* **@param** tipoDocumento

\* tipo de documenteo del vendedor a modificar

\* **@param** numeroDocumento

\* número de documento del vendedor a modificar

\* **@param** resultadoEncriptar

\* resultado devuelto por el mock encriptadorPassword

\* **@param** contraseñaAntigua

\* contraseña antigua del vendedor

\* **@param** contraseñaNueva

\* contraseña nueva del vendedor

\* **@param** contraseñaNueva2

\* campo repetir contraseña del vendedor

\* **@param** resultadoModificarVendedorEsperado

\* resultado devuelto por la lógica

\* **@param** llamaAPresentadorVentanasPresentarError

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

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

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

\* **@param** llamaAPresentadorVentanasPresentarExcepcionInesperada

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

\* **@param** llamaAModificarVendedor

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

\* **@param** excepcion

\* excepción devuelta por la capa lógica

\* **@param** llamaACambiarScene

\* indica si se debe llamar a cambiar de pantalla

\* **@param** checkBoxSeleccionado

\* checkBox de cambiar contraseña seleccionado o no seleccionado

\* **@throws** Throwable

\*/

@Test

@Parameters

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

//Se crean los mocks necesarios

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

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

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

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

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

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

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

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

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

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

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

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

Vendedor vendedor = **new** Vendedor()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setPassword(contraseñaAntigua);

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

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

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

}

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

tipos.add(tipoDocumento);

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

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

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

@Override

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

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

**this**.encriptador = encriptadorPasswordMock;

**this**.setVendedor(vendedor);

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

}

@Override

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

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

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

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

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

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

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

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

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

**super**.acceptAction();

};

@Override

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

}

};

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

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

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

@Override

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

//Método a probar

modificarVendedorController.acceptAction();

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

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

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

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

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

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

}

};

//Se corre el test en el hilo de JavaFX

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

}