# 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

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

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

/\*\*

\* 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 //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();

}