# Código Fuente y Pruebas de Unidad

## Iteración 1:

#### Taskcard 2 Entidad vendedor

Código del archivo en Vendedor.java

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

@Entity

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

/\*\*

\* Entidad que modela a un vendedor

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

\*/

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

@Id

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

**protected** Integer id; //ID

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

**private** String nombre;

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

**private** String apellido;

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

**private** String numeroDocumento;

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

**private** String password;

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

**private** String salt;

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

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

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

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

**private** TipoDocumento tipoDocumento;

//estado lógico que tiene la entidad

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

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

**private** Estado estado;

//getters, setters, constructores y otros métodos

(…)

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

Código del archivo GestorVendedor.java

/\*\*

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

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

\*

\* **@param** vendedor

\* vendedor a crear

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

\*

\* **@throws** PersistenciaException

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

\* **@throws** GestionException

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

\*/

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

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

// valida formato de datos

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

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

}

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

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

}

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

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

}

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

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

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

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

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

}

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

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

}

}

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

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

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

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

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

}

}

persistidorVendedor.guardarVendedor(vendedor);

}

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

}

/\*\*

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

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

\*

\* **@param** vendedor

\* vendedor a modificar

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

\*

\* **@throws** PersistenciaException

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

\*/

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

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

// valida formato de datos

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

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

}

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

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

}

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

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

}

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

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

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

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

}

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

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

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

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

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

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

}

}

}

persistidorVendedor.modificarVendedor(vendedor);

}

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

}

/\*\*

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

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

\*

\* **@param** vendedor

\* vendedor a eliminar

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

\*

\* **@throws** PersistenciaException

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

\*/

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

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

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

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

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

vendedor.setEstado(e);

}

}

persistidorVendedor.modificarVendedor(vendedor);

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

}

Código del test GestorVendedorTest.java

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

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

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

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

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

*vendedor*.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setSalt("abcd")

.setPassword("1234");

//Parámetros de JUnitParams

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

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

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

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

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

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

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

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

};

}

/\*\*

\* Test para la función crearVendedor

\*

\* **@param** resValNombre

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

\* **@param** resValApellido

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

\* **@param** resValDocumento

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

\* **@param** resObtenerVendedor

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

\* **@param** guardar

\* 1 si va a guardar, 0 si no

\* **@param** resultadoCrearVendedorEsperado

\* resultado esperado que retorne el método a probar

\* **@throws** Exception

\*/

@Test

@Parameters

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

//Inicialización de los mocks

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

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

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

//Clase anónima necesaria para inyectar dependencias

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

{

**this**.persistidorVendedor = vendedorServiceMock;

**this**.validador = validadorFormatoMock;

**this**.gestorDatos = gestorDatosMock;

}

};

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

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

//Setear valores esperados a los mocks

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

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

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

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

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

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

//Llamar al método a testear

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

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

*assertEquals*(resultadoCrearVendedorEsperado, resultadoCrearVendedor);

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

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

}

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

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

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

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

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

}

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

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

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

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

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

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

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

**this**.id = id;

**return** **this**;

}

{

**this**.setId(1)

.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setSalt("abcd")

.setPassword("1234")

.setEstado(est);

}

};

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

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

**this**.id = id;

**return** **this**;

}

{

**this**.setId(2)

.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setSalt("abcd")

.setPassword("1234");

}

};

//Parámetros de JUnitParams

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

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

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

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

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

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

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

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

};

}

/\*\*

\* Test para la función modificarVendedor

\*

\* **@param** resValNombre

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

\* **@param** resValApellido

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

\* **@param** resValDocumento

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

\* **@param** resObtenerVendedor

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

\* **@param** modificar

\* 1 si va a modificar, 0 si no

\* **@param** resultadoModificarVendedorEsperado

\* resultado esperado que retorne el método a probar

\* **@throws** Exception

\*/

@Test

@Parameters

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

//Inicialización de los mocks

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

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

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

//Clase anónima necesaria para inyectar dependencias

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

{

**this**.persistidorVendedor = vendedorServiceMock;

**this**.validador = validadorFormatoMock;

**this**.gestorDatos = gestorDatosMock;

}

};

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

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

//Setear valores esperados a los mocks

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

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

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

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

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

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

//Llamar al método a testear

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

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

*assertEquals*(resultadoModificarVendedorEsperado, resultadoModificarVendedor);

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

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

}

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

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

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

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

}

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

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

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

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

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

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

*vendedorE*.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setSalt("abcd")

.setPassword("1234")

.setEstado(est);

//Parámetros de JUnitParams

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

//resultadoObtenerVendedor, eliminar, resultadoEsperado

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

};

}

@Test

@Parameters

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

//Inicialización de los mocks

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

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

//Clase anónima necesaria para inyectar dependencias

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

{

**this**.persistidorVendedor = vendedorServiceMock;

**this**.gestorDatos = gestorDatosMock;

}

};

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

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

//Setear valores esperados a los mocks

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

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

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

//Llamar al método a testear

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

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

*assertEquals*(resultadoEliminarVendedorEsperado, resultadoEliminarVendedor);

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

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

}

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

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

}

#### Taskcard 6 Persistidor vendedor

Código del archivo en VendedorServiceImpl.java

/\*\*

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

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

\*

\*/

@Repository

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

**private** SessionFactory sessionFactory;

@Autowired

**public** VendedorServiceImpl(SessionFactory sessionFactory) {

**this**.sessionFactory = sessionFactory;

}

**public** SessionFactory getSessionFactory() {

**return** sessionFactory;

}

@Override

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

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

Session session = getSessionFactory().getCurrentSession();

**try**{

session.save(vendedor);

} **catch**(Exception e){

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

}

}

@Override

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

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

Session session = getSessionFactory().getCurrentSession();

**try**{

session.update(vendedor);

} **catch**(Exception e){

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

}

}

@Override

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

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

Vendedor vendedor = **null**;

Session session = getSessionFactory().getCurrentSession();

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

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

} **catch**(NoResultException e){

**return** **null**;

} **catch**(NonUniqueResultException e){

**return** **null**;

} **catch**(Exception e){

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

}

**return** vendedor;

}

@Override

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

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

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

Session session = getSessionFactory().getCurrentSession();

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

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

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

vendedores.add((Vendedor) o);

}

}

} **catch**(Exception e){

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

}

**return** vendedores;

}

}

#### Taskcard 8 Vista alta, modificar y baja propietario

Código del alta en AltaPropietarioController.java

/\*\*

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

\*

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

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

\*/

@FXML

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

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

//obtengo datos introducidos por el usuario

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

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

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

String alturaCalle = textFieldAlturaCalle.getText().trim();

String piso = textFieldPiso.getText().trim();

String departamento = textFieldDepartamento.getText().trim();

String telefono = textFieldTelefono.getText().trim();

String correoElectronico = textFieldCorreoElectronico.getText().trim();

String otros = textFieldOtros.getText().trim();

Localidad localidad = comboBoxLocalidad.getValue();

TipoDocumento tipoDoc = comboBoxTipoDocumento.getValue();

Barrio barrio = comboBoxBarrio.getValue();

Calle calle = comboBoxCalle.getValue();

//verifico que no estén vacíos

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

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

}

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

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

}

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

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

}

**if**(alturaCalle.isEmpty()){

error.append("Inserte una altura").append("\n ");

}

**if**(telefono.isEmpty()){

error.append("Inserte un telefono").append("\n ");

}

**if**(correoElectronico.isEmpty()){

error.append("Inserte un correo electrónico").append("\n ");

}

**if**(calle == **null**){

error.append("Elija una calle").append("\n");

}

**if**(barrio == **null**){

error.append("Elija un barrio").append("\n");

}

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

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

}

**if**(localidad == **null**){

error.append("Elija una localidad").append("\n");

}

**if**(!error.toString().isEmpty()){ //si hay algún error lo muestro al usuario

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

}

**else**{

//Si no hay errores se crean las entidades con los datos introducidos

Direccion direccion = **new** Direccion();

direccion.setNumero(alturaCalle)

.setCalle(calle)

.setBarrio(barrio)

.setPiso(piso)

.setDepartamento(departamento)

.setLocalidad(localidad)

.setOtros(otros);

Propietario propietario = **new** Propietario();

propietario.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setEmail(correoElectronico)

.setDireccion(direccion);

**try**{ //relevo la operación a capa lógica

ResultadoCrearPropietario resultado = coordinador.crearPropietario(propietario);

**if**(resultado.hayErrores()){ // si hay algún error se muestra al usuario

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

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

**switch**(err) {

**case** ***Formato\_Nombre\_Incorrecto***:

stringErrores.append("Formato de nombre incorrecto.\n");

**break**;

**case** ***Formato\_Apellido\_Incorrecto***:

stringErrores.append("Formato de apellido incorrecto.\n");

**break**;

**case** ***Formato\_Telefono\_Incorrecto***:

stringErrores.append("Formato de teléfono incorrecto.\n");

**break**;

**case** ***Formato\_Documento\_Incorrecto***:

stringErrores.append("Tipo y formato de documento incorrecto.\n");

**break**;

**case** ***Formato\_Email\_Incorrecto***:

stringErrores.append("Formato de email incorrecto.\n");

**break**;

**case** ***Formato\_Direccion\_Incorrecto***:

stringErrores.append("Formato de dirección incorrecto.\n");

**break**;

**case** ***Ya\_Existe\_Propietario***:

stringErrores.append("Ya existe un cliente con ese tipo y número de documento.\n");

**break**;

}

}

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

}

**else**{ //si no hay errores se muestra notificación y se vuelve a la pantalla de listar propietarios

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

cambiarmeAScene(AdministrarPropietarioController.***URLVista***);

}

} **catch**(GestionException e){ //excepción originada en gestor

**if**(e.getClass().equals(EntidadExistenteConEstadoBajaException.**class**)){

//el propietario existe pero fué dado de baja

VentanaConfirmacion ventana = presentador.presentarConfirmacion("El propietario ya existe", "El propietario ya existía anteriormente pero fué dado de baja.\n ¿Desea volver a darle de alta?", stage);

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

//usuario acepta volver a darle de alta. Se pasa a la pantalla de modificar propietario

ModificarPropietarioController controlador = (ModificarPropietarioController) cambiarmeAScene(ModificarPropietarioController.***URLVista***);

controlador.setPropietarioEnModificacion(propietario);

}

}

} **catch**(PersistenciaException e){ //excepción originada en la capa de persistencia

presentador.presentarExcepcion(e, stage);

}

}

}

Código del test del alta en AltaPropietarioControllerTest.java

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

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

Pais pais = **new** Pais("Argentina");

Provincia provincia = **new** Provincia("Santa fé", pais);

Localidad localidad = **new** Localidad("Capital", provincia);

Direccion dir = **new** Direccion("1865", "1", "B", **new** Calle("San martín", localidad), **new** Barrio("Centro", localidad), localidad, "");

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

//nombre,apellido,tipoDocumento,nroDocumento,telefono,email,direccion,resultadoEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,llamaACrearPropietario,excepcion,aceptarVentanaConfirmacion,llamaACambiarScene

//prueba correcta

/\*0\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 0, 1, **null**, **true**, 0 },

//prueba nombre incorrecto

/\*1\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearNombreIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba apellido incorrecto

/\*2\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearApellidoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba documento incorrecto

/\*3\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearDocumentoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba teléfono incorrecto

/\*4\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearTelefonoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba email incorrecto

/\*5\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearEmailIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba direccion incorrecta

/\*6\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearDireccionIncorrecta***, 1, 0, 1, **null**, **true**, 0 },

//prueba ya existe propietario

/\*7\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCrearYaExiste***, 1, 0, 1, **null**, **true**, 0 },

//prueba ya existe propietario

/\*8\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, **new** ResultadoCrearPropietario(ErrorCrearPropietario.***Formato\_Nombre\_Incorrecto***, ErrorCrearPropietario.***Formato\_Apellido\_Incorrecto***), 1, 0, 1, **null**, **true**, 0 },

//prueba nombre vacio

/\*9\*/ **new** Object[] { "", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, **null**, 1, 0, 0, **null**, **true**, 0 },

//prueba propietario Existente y acepta

/\*10\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **true**, 1 },

//prueba propietario Existente y cancela

/\*11\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **false**, 0 },

//prueba PersistenciaException

/\*12\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "juanperez@hotmail.com", dir, ***resultadoCorrecto***, 0, 1, 1, **new** SaveUpdateException(**new** Throwable()), **false**, 0 }

};

}

/\*\*

\* Test para probar al controlador de alta propietario cuando el usuario presiona aceptar

\*

\* **@param** nombre

\* nombre que se introduce por el usuario

\* **@param** apellido

\* apellido que se introduce por el usuario

\* **@param** tipoDocumento

\* tipo de documento que se introduce por el usuario

\* **@param** numeroDocumento

\* número de documento que se introduce por el usuario

\* **@param** telefono

\* teléfono que se introduce por el usuario

\* **@param** email

\* email que se introduce por el usuario

\* **@param** direccion

\* dirección que se introduce por el usuario

\* **@param** resultadoCrearPropietarioEsperado

\* resultado que se espera que retorne el método a probar

\* **@param** llamaAPresentadorVentanasPresentarError

\* 1 si llama al método presentar error del presentador de ventanas, 0 si no

\* **@param** llamaAPresentadorVentanasPresentarExcepcion

\* 1 si llama al método presentar excepción del presentador de ventanas, 0 si no

\* **@param** llamaACrearPropietario

\* 1 si llama al método crear propietario de la capa lógica, 0 si no

\* **@param** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@param** aceptarVentanaConfirmacion

\* si el usuario acepta la ventana de confirmación

\* **@param** llamaACambiarScene

\* 1 si llama al método cambiar scene, 0 si no

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCrearPropietario(String nombre,

String apellido,

TipoDocumento tipoDocumento,

String numeroDocumento,

String telefono,

String email,

Direccion direccion,

ResultadoCrearPropietario resultadoCrearPropietarioEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Integer llamaACrearPropietario,

Exception excepcion,

Boolean aceptarVentanaConfirmacion,

Integer llamaACambiarScene) **throws** Throwable {

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

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

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

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

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

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

ModificarPropietarioController modificarPropietarioControllerMock = *mock*(ModificarPropietarioController.**class**);

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

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

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

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

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

*doNothing*().when(modificarPropietarioControllerMock).setPropietarioEnModificacion(*any*());

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

*propietario* = **new** Propietario()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setEmail(email)

.setDireccion(direccion);

*when*(coordinadorMock.crearPropietario(*propietario*)).thenReturn(resultadoCrearPropietarioEsperado);

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

*when*(coordinadorMock.crearPropietario(*propietario*)).thenThrow(excepcion);

}

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

tipos.add(tipoDocumento);

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

AltaPropietarioController altaPropietarioController = **new** AltaPropietarioController() {

@Override

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

**this**.coordinador = coordinadorMock;

**this**.presentador = presentadorVentanasMock;

setScenographyChanger(scenographyChangerMock);

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

}

@Override

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

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

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

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

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

**this**.textFieldTelefono.setText(telefono);

**this**.textFieldCorreoElectronico.setText(email);

**this**.comboBoxPais.getSelectionModel().select(direccion.getLocalidad().getProvincia().getPais());

**this**.comboBoxProvincia.getSelectionModel().select(direccion.getLocalidad().getProvincia());

**this**.comboBoxLocalidad.getSelectionModel().select(direccion.getLocalidad());

**this**.comboBoxBarrio.getSelectionModel().select(direccion.getBarrio());

**this**.comboBoxCalle.getSelectionModel().select(direccion.getCalle());

**this**.textFieldAlturaCalle.setText(direccion.getNumero());

**this**.textFieldDepartamento.setText(direccion.getDepartamento());

**this**.textFieldPiso.setText(direccion.getPiso());

**this**.textFieldOtros.setText(direccion.getOtros());

**super**.acceptAction();

}

@Override

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

}

};

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(AltaPropietarioController.***URLVista***, altaPropietarioController);

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

@Override

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

altaPropietarioController.acceptAction();

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

Mockito.*verify*(coordinadorMock, Mockito.*times*(llamaACrearPropietario)).crearPropietario(Mockito.*any*());

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

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

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

Mockito.*verify*(modificarPropietarioControllerMock, *times*(llamaACambiarScene)).setPropietarioEnModificacion(*propietario*);

}

};

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

}

Código del modificar en ModificarPropietarioController.java

#### Taskcard 17 Vista alta, modificar y baja cliente

Código del archivo en Vendedor.java

#### Taskcard 18 Lógica alta, modificar y baja cliente

Código del archivo en Vendedor.java