# 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 retornará el mock de capa lógica

\* **@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

/\*\*

\* 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 modifican las entidades con los datos introducidos

propietarioEnModificacion.getDireccion().setNumero(alturaCalle)

.setCalle(calle)

.setBarrio(barrio)

.setPiso(piso)

.setDepartamento(departamento)

.setLocalidad(localidad)

.setOtros(otros);

propietarioEnModificacion.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setEmail(correoElectronico);

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

ResultadoModificarPropietario resultado = coordinador.modificarPropietario(propietarioEnModificacion);

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

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

**for**(ErrorModificarPropietario 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("Otro cliente posee 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 modificado el propietario con éxito", stage);

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

}

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

presentador.presentarExcepcion(e, stage);

}

}

}

Código del test de modificar en ModificarPropietarioController.java

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

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[] {

//String nombre,apellido,tipoDocumento,nroDocumento,telefono,email,direccion,resultadoModificarPropietarioEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,excepcion

//prueba correcta

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

//prueba nombre incorrecto

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

//prueba apellido incorrecto

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

//prueba documento incorrecto

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

//prueba teléfono incorrecto

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

//prueba email incorrecto

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

//prueba direccion incorrecta

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

//prueba ya existe propietario

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

//prueba ya existe propietario

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

//prueba nombre vacio

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

//prueba PersistenciaException

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

};

}

/\*\*

\* Test para probar al controlador de modificar 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** resultadoModificarPropietarioEsperado

\* resultado que retornará el mock de capa lógica

\* **@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** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testModificarPropietario(String nombre,

String apellido,

TipoDocumento tipoDocumento,

String numeroDocumento,

String telefono,

String email,

Direccion direccion,

ResultadoModificarPropietario resultadoModificarPropietarioEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Exception excepcion) **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*(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.modificarPropietario(*propietario*)).thenReturn(resultadoModificarPropietarioEsperado);

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

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

}

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

tipos.add(tipoDocumento);

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

ModificarPropietarioController modificarPropietarioController = **new** ModificarPropietarioController() {

@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) {

}

@Override

**public** **void** setPropietarioEnModificacion(Propietario propietarioEnModificacion) {

**this**.propietarioEnModificacion = propietarioEnModificacion;

}

};

modificarPropietarioController.setPropietarioEnModificacion(*propietario*);

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(ModificarPropietarioController.***URLVista***, modificarPropietarioController);

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

@Override

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

modificarPropietarioController.acceptAction();

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

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

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

}

};

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

}

Código de baja en AdministrarPropietarioController.java

/\*\*

\* Acción que se ejecuta al presionar el botón eliminar

\* Se muestra una ventana emergente para confirmar la operación

\*/

@FXML

**protected** ResultadoControlador handleEliminar() {

ArrayList<ErrorControlador> erroresControlador = **new** ArrayList<>();

**if**(tablaPropietarios.getSelectionModel().getSelectedItem() == **null**){

**return** **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***);

}

//solicito confirmación al usuario

VentanaConfirmacion ventana = presentador.presentarConfirmacion("Eliminar propietario", "Está a punto de eliminar al propietario.\n ¿Está seguro que desea hacerlo?", **this**.stage);

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

**return** **new** ResultadoControlador(); //si no acepta

}

**try**{

ResultadoEliminarPropietario resultado = coordinador.eliminarPropietario(tablaPropietarios.getSelectionModel().getSelectedItem());

**if**(resultado.hayErrores()){ // si hay errores lo muestro al usuario

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

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

**switch**(err) {

}

}

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

}

**else**{ //si no hay errores muestro notificación

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

}

//actualizo la tabla

tablaPropietarios.getItems().clear();

tablaPropietarios.getItems().addAll(coordinador.obtenerPropietarios());

**return** **new** ResultadoControlador(erroresControlador.toArray(**new** ErrorControlador[0]));

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

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

} **catch**(Exception e){ // alguna otra excepción inesperada

presentador.presentarExcepcionInesperada(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

}

Código del test de baja en AdministrarPropietarioControllerTest.java

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

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

Boolean acepta = **true**;

ResultadoControlador resultadoControladorCorrecto = **new** ResultadoControlador();

ResultadoControlador resultadoControladorErrorPersistencia = **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

ResultadoControlador resultadoControladorErrorDesconocido = **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

ResultadoEliminarPropietario resultadoLogicaCorrecto = **new** ResultadoEliminarPropietario();

Throwable excepcionPersistencia = **new** ObjNotFoundException("", **new** Exception());

Throwable excepcionInesperada = **new** Exception();

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

//propietario, acepta, resultadoControlador,resultadoLogica,excepcion

/\*0\*/ **new** Object[] { propietario, acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario acepta y el propietario se elimina correctamente

/\*1\*/ **new** Object[] { propietario, !acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario no acepta, pero de haber aceptado, se hubiese eliminado el propietario correctamente

/\*2\*/ **new** Object[] { propietario, acepta, resultadoControladorErrorPersistencia, **null**, excepcionPersistencia }, //test donde el controlador tira una excepción de persistencia

/\*3\*/ **new** Object[] { propietario, acepta, resultadoControladorErrorDesconocido, **null**, excepcionInesperada } //test donde el controlador tira unaexcepción inesperada

};

}

/\*\*

\* Test para probar eliminar propietario en el controlador administrar cuando el usuario presiona eliminar

\*

\* **@param** propietario

\* propietario a eliminar

\* **@param** acepta

\* si el usuario acepta la confirmación de eliminar

\* **@param** resultadoControlador

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

\* **@param** resultadoLogica

\* resultado que devuelve la operación de capa lógica

\* **@param** excepcion

\*/

**public** **void** testEliminarPropietario(Propietario propietario, Boolean acepta, ResultadoControlador resultadoControlador, ResultadoEliminarPropietario resultadoLogica, Throwable excepcion) **throws** Exception {

CoordinadorJavaFX coordinadorMock = **new** CoordinadorJavaFX() {

@Override

**public** ResultadoEliminarPropietario eliminarPropietario(Propietario propietario) **throws** PersistenciaException {

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

**return** resultadoLogica;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

@Override

**public** ArrayList<Propietario> obtenerPropietarios() **throws** PersistenciaException {

ArrayList<Propietario> propietarios = **new** ArrayList<>();

propietarios.add(propietario);

**return** propietarios;

}

};

PresentadorVentanas presentadorMock = **new** PresentadorVentanasMock(acepta);

AdministrarPropietarioController administrarPropietarioController = **new** AdministrarPropietarioController() {

@Override

**public** ResultadoControlador handleEliminar() {

tablaPropietarios.getSelectionModel().select(propietario);

**return** **super**.handleEliminar();

}

@Override

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

}

};

administrarPropietarioController.setCoordinador(coordinadorMock);

administrarPropietarioController.setPresentador(presentadorMock);

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarPropietarioController.***URLVista***, administrarPropietarioController);

administrarPropietarioController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

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

@Override

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

*assertEquals*(resultadoControlador, administrarPropietarioController.handleEliminar());

}

};

**try**{

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

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

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

Código del alta en AltaClienteControllerTest.java

/\*\*

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

\*

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

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

\*/

@FXML

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

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

String correo = textFieldCorreo.getText().trim();

TipoDocumento tipoDoc = comboBoxTipoDocumento.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**(tipoDoc == **null**){

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

}

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

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

}

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

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

}

**if**(correo.isEmpty()){

error.append("Inserte una dirección de correo electrónico").append("\n");

}

**if**(cliente.getInmuebleBuscado() == **null**){

error.append("Debe cargar un inmueble buscado al cliente").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

cliente.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setCorreo(correo);

**try**{

//relevo la operación a capa lógica

ResultadoCrearCliente resultado = coordinador.crearCliente(cliente);

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

// si hay algún error se muestra al usuario

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

**for**(ErrorCrearCliente 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\_Correo\_Incorrecto***:

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

**break**;

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

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

**break**;

**case** ***Ya\_Existe\_Cliente***:

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 clientes

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

cambiarmeAScene(AdministrarClienteController.***URLVista***);

}

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

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

//el cliente existe pero fué dado de baja

VentanaConfirmacion ventana = presentador.presentarConfirmacion("El cliente ya existe", "El cliente 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 cliente

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

controlador.setClienteEnModificacion(cliente);

}

}

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

presentador.presentarExcepcion(e, stage);

}

}

}

/\*\*

\* Acción que se ejecuta al presionar el botón cargar inmueble.

\* Se pasa a la pantalla de inmueble buscado

\*/

@FXML

**private** **void** cargarInmueble() {

//se guardan en el cliente los datos introducidos por el usuario

cliente.setNombre(textFieldNombre.getText().trim())

.setApellido(textFieldApellido.getText().trim())

.setTipoDocumento(comboBoxTipoDocumento.getValue())

.setNumeroDocumento(textFieldNumeroDocumento.getText().trim())

.setTelefono(textFieldTelefono.getText().trim())

.setCorreo(textFieldCorreo.getText().trim());

//se pasa a la pantalla de cargar inmueble

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

controlador.setCliente(cliente);

}

Código del test del alta en AltaClienteControllerTest.java

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

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

InmuebleBuscado inm = **new** InmuebleBuscado(**null**, 1000000.00, 10.0, 10, 1, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

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

//nombre,apellido,tipoDocumento,numeroDocumento,telefono,correo,inmueble,resultadoCrearClienteEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,llamaACrearCliente,excepcion,aceptarVentanaConfirmacion,llamaACambiarScene

//prueba correcta

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

//prueba nombre incorrecto

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

//prueba apellido incorrecto

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

//prueba documento incorrecto

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

//prueba teléfono incorrecto

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

//prueba correo incorrecto

/\*5\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCrearCorreoIncorrecto***, 1, 0, 1, **null**, **true**, 0 },

//prueba ya existe cliente

/\*6\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCrearYaExiste***, 1, 0, 1, **null**, **true**, 0 },

//prueba ya existe cliente

/\*7\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, **new** ResultadoCrearCliente(ErrorCrearCliente.***Formato\_Nombre\_Incorrecto***, ErrorCrearCliente.***Formato\_Apellido\_Incorrecto***), 1, 0, 1, **null**, **true**, 0 },

//prueba nombre vacio

/\*8\*/ **new** Object[] { "", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, **null**, 1, 0, 0, **null**, **true**, 0 },

//prueba cliente Existente y acepta

/\*9\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **true**, 1 },

//prueba cliente Existente y cancela

/\*10\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 0, 1, **new** EntidadExistenteConEstadoBajaException(), **false**, 0 },

//prueba PersistenciaException

/\*11\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 1, 1, **new** SaveUpdateException(**new** Throwable()), **false**, 0 }

};

}

/\*\*

\* Test para probar crear un cliente cuando el usuario presiona el botón aceptar

\*

\* **@param** nombre

\* nombre que es introducido por el usuario

\* **@param** apellido

\* appellido que es introducido por el usuario

\* **@param** tipoDocumento

\* tipo de documento que es introducido por el usuario

\* **@param** numeroDocumento

\* número de documento que es introducido por el usuario

\* **@param** telefono

\* teléfono que es introducido por el usuario

\* **@param** correo

\* correo que es introducido por el usuario

\* **@param** inmueble

\* inmueble buscado cargado por el usuario en la pantalla de cargar inmueble buscado

\* **@param** resultadoCrearClienteEsperado

\* resultado que retornará el mock de capa lógica

\* **@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** llamaACrearCliente

\* 1 si llama al método crear cliente 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** testCrearCliente(String nombre,

String apellido,

TipoDocumento tipoDocumento,

String numeroDocumento,

String telefono,

String correo,

InmuebleBuscado inmueble,

ResultadoCrearCliente resultadoCrearClienteEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Integer llamaACrearCliente,

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**);

ModificarClienteController modificarClienteControllerMock = *mock*(ModificarClienteController.**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(modificarClienteControllerMock);

*doNothing*().when(modificarClienteControllerMock).setClienteEnModificacion(*any*());

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

*cliente* = **new** Cliente()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setInmuebleBuscado(inmueble)

.setTelefono(telefono)

.setCorreo(correo);

inmueble.setCliente(*cliente*);

*when*(coordinadorMock.crearCliente(*cliente*)).thenReturn(resultadoCrearClienteEsperado);

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

*when*(coordinadorMock.crearCliente(*cliente*)).thenThrow(excepcion);

}

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

tipos.add(tipoDocumento);

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

AltaClienteController altaClienteController = **new** AltaClienteController() {

@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**.textFieldCorreo.setText(correo);

**super**.acceptAction();

}

@Override

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

}

@Override

**public** **void** setCliente(Cliente cliente) {

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

**this**.cliente = cliente;

}

**else**{

**this**.cliente = **new** Cliente();

}

}

};

altaClienteController.setCliente(*cliente*);

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(AltaClienteController.***URLVista***, altaClienteController);

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

@Override

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

altaClienteController.acceptAction();

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

Mockito.*verify*(coordinadorMock, Mockito.*times*(llamaACrearCliente)).crearCliente(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(ModificarClienteController.***URLVista***, **false**);

Mockito.*verify*(modificarClienteControllerMock, *times*(llamaACambiarScene)).setClienteEnModificacion(*cliente*);

}

};

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

}

Código de modificar en ModificarClienteController.java

/\*\*

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

\*

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

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

\*/

@FXML

**protected** **void** acceptAction() {

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

String correo = textFieldCorreo.getText().trim();

TipoDocumento tipoDoc = comboBoxTipoDocumento.getValue();

//verifico que no estén vacíos

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

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

}

**if**(correo.isEmpty()){

error.append("Inserte una dirección de correo electrónico").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 modifican las entidades con los datos introducidos

clienteEnModificacion.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDoc)

.setNumeroDocumento(numeroDocumento)

.setTelefono(telefono)

.setCorreo(correo);

**try**{

//relevo la operación a capa lógica

ResultadoModificarCliente resultado = coordinador.modificarCliente(clienteEnModificacion);

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

// si hay algún error se muestra al usuario

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

**for**(ErrorModificarCliente 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\_Correo\_Incorrecto***:

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

**break**;

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

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

**break**;

**case** ***Otro\_Cliente\_Posee\_Mismo\_Documento\_Y\_Tipo***:

stringErrores.append("Otro cliente ya posee 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 clientes

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

cambiarmeAScene(AdministrarClienteController.***URLVista***);

}

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

presentador.presentarExcepcion(e, stage);

}

}

}

Código del test de modificar en ModificarClienteControllerTest.java

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

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

InmuebleBuscado inm = **new** InmuebleBuscado(**null**, 1000000.00, 10.0, 10, 1, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

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

//nombre,apellido,tipoDocumento,numeroDocumento,telefono,correo,inmueble,resultadoModificarClienteEsperado,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,excepcion

//prueba correcta

/\*0\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 0, **null** },

//prueba nombre incorrecto

/\*1\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarNombreIncorrecto***, 1, 0, **null** },

//prueba apellido incorrecto

/\*2\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarApellidoIncorrecto***, 1, 0, **null** },

//prueba documento incorrecto

/\*3\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarDocumentoIncorrecto***, 1, 0, **null** },

//prueba teléfono incorrecto

/\*4\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarTelefonoIncorrecto***, 1, 0, **null** },

//prueba correo incorrecto

/\*5\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarCorreoIncorrecto***, 1, 0, **null** },

//prueba ya existe cliente

/\*6\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoModificarYaExiste***, 1, 0, **null** },

//prueba ya existe cliente

/\*7\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, **new** ResultadoModificarCliente(ErrorModificarCliente.***Formato\_Nombre\_Incorrecto***, ErrorModificarCliente.***Formato\_Apellido\_Incorrecto***), 1, 0, **null** },

//prueba nombre vacio

/\*8\*/ **new** Object[] { "", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, **null**, 1, 0, **null** },

//prueba PersistenciaException

/\*9\*/ **new** Object[] { "Juan", "Perez", doc, "12345678", "123-123", "asdf@asf.com", inm, ***resultadoCorrecto***, 0, 1, **new** SaveUpdateException(**new** Throwable()) }

};

}

/\*\*

\* Test para probar crear un cliente cuando el usuario presiona el botón aceptar

\*

\* **@param** nombre

\* nombre que es introducido por el usuario

\* **@param** apellido

\* appellido que es introducido por el usuario

\* **@param** tipoDocumento

\* tipo de documento que es introducido por el usuario

\* **@param** numeroDocumento

\* número de documento que es introducido por el usuario

\* **@param** telefono

\* teléfono que es introducido por el usuario

\* **@param** correo

\* correo que es introducido por el usuario

\* **@param** inmueble

\* inmueble buscado cargado por el usuario en la pantalla de cargar inmueble buscado

\* **@param** resultadoModificarClienteEsperado

\* resultado que retornará el mock de capa lógica

\* **@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** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testModificarCliente(String nombre,

String apellido,

TipoDocumento tipoDocumento,

String numeroDocumento,

String telefono,

String correo,

InmuebleBuscado inmueble,

ResultadoModificarCliente resultadoModificarClienteEsperado,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Exception excepcion) **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**);

ModificarClienteController modificarClienteControllerMock = *mock*(ModificarClienteController.**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*(scenographyChangerMock.cambiarScenography(*any*(String.**class**), *any*())).thenReturn(modificarClienteControllerMock);

*doNothing*().when(modificarClienteControllerMock).setClienteEnModificacion(*any*());

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

*cliente* = **new** Cliente()

.setNombre(nombre)

.setApellido(apellido)

.setTipoDocumento(tipoDocumento)

.setNumeroDocumento(numeroDocumento)

.setInmuebleBuscado(inmueble)

.setTelefono(telefono)

.setCorreo(correo);

inmueble.setCliente(*cliente*);

*when*(coordinadorMock.modificarCliente(*cliente*)).thenReturn(resultadoModificarClienteEsperado);

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

*when*(coordinadorMock.modificarCliente(*cliente*)).thenThrow(excepcion);

}

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

tipos.add(tipoDocumento);

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

ModificarClienteController modificarClienteController = **new** ModificarClienteController() {

@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**.textFieldCorreo.setText(correo);

**super**.acceptAction();

}

@Override

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

}

@Override

**public** **void** setClienteEnModificacion(Cliente clienteEnModificacion) {

**this**.clienteEnModificacion = clienteEnModificacion;

}

};

modificarClienteController.setClienteEnModificacion(*cliente*);

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(ModificarClienteController.***URLVista***, modificarClienteController);

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

@Override

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

modificarClienteController.acceptAction();

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

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

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

}

};

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

}

Código de la baja en AdministrarClienteController.java

/\*\*

\* Acción que se ejecuta al presionar el botón eliminar

\* Se muestra una ventana emergente para confirmar la operación

\*/

@FXML

**protected** ResultadoControlador handleEliminar() {

ArrayList<ErrorControlador> erroresControlador = **new** ArrayList<>();

**if**(tablaClientes.getSelectionModel().getSelectedItem() == **null**){

**return** **new** ResultadoControlador(ErrorControlador.***Campos\_Vacios***);

}

//solicito confirmación al usuario

VentanaConfirmacion ventana = presentador.presentarConfirmacion("Eliminar cliente", "Está a punto de eliminar al cliente.\n ¿Está seguro que desea hacerlo?", **this**.stage);

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

**return** **new** ResultadoControlador();//si no acepta

}

**try**{

ResultadoEliminarCliente resultado = coordinador.eliminarCliente(tablaClientes.getSelectionModel().getSelectedItem());

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

// si hay errores lo muestro al usuario

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

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

**switch**(err) {

}

}

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

}

**else**{ //si no hay errores muestro notificación

presentador.presentarToast("Se ha eliminado el cliente con éxito", stage);

}

//actualizo la tabla

tablaClientes.getItems().clear();

tablaClientes.getItems().addAll(coordinador.obtenerClientes());

**return** **new** ResultadoControlador(erroresControlador.toArray(**new** ErrorControlador[0]));

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

presentador.presentarExcepcion(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

} **catch**(Exception e){// alguna otra excepción inesperada

presentador.presentarExcepcionInesperada(e, stage);

**return** **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

}

}

Código del test de la baja en AdministrarClienteControllerTest.java

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

Cliente cliente = **new** Cliente();

Boolean acepta = **true**;

ResultadoControlador resultadoControladorCorrecto = **new** ResultadoControlador();

ResultadoControlador resultadoControladorErrorPersistencia = **new** ResultadoControlador(ErrorControlador.***Error\_Persistencia***);

ResultadoControlador resultadoControladorErrorDesconocido = **new** ResultadoControlador(ErrorControlador.***Error\_Desconocido***);

ResultadoEliminarCliente resultadoLogicaCorrecto = **new** ResultadoEliminarCliente();

Throwable excepcionPersistencia = **new** ObjNotFoundException("", **new** Exception());

Throwable excepcionInesperada = **new** Exception();

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

//cliente,acepta,resultadoControlador,resultadoLogica,excepcion

/\*0\*/ **new** Object[] { cliente, acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario acepta y el cliente se elimina correctamente

/\*1\*/ **new** Object[] { cliente, !acepta, resultadoControladorCorrecto, resultadoLogicaCorrecto, **null** }, //test donde el usuario no acepta, pero de haber aceptado, se hubiese eliminado el cliente correctamente

/\*2\*/ **new** Object[] { cliente, acepta, resultadoControladorErrorPersistencia, **null**, excepcionPersistencia }, //test donde el controlador tira una excepción de persistencia

/\*3\*/ **new** Object[] { cliente, acepta, resultadoControladorErrorDesconocido, **null**, excepcionInesperada } //test donde el controlador tira unaexcepción inesperada

};

}

@Test

@Parameters

/\*\*

\* Test para probar la baja de un cliente en el controlador de administrar clientes

\*

\* **@param** cliente

\* cliente a eliminar

\* **@param** acepta

\* si usuario acepta confirmación de eliminar

\* **@param** resultadoControlador

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

\* **@param** resultadoLogica

\* resultado que retornará el mock de la capa lógica

\* **@param** excepcion

\* excepción que se simula lanzar desde la capa lógica

\*/

**public** **void** testEliminarCliente(Cliente cliente, Boolean acepta, ResultadoControlador resultadoControlador, ResultadoEliminarCliente resultadoLogica, Throwable excepcion) **throws** Exception {

CoordinadorJavaFX coordinadorMock = **new** CoordinadorJavaFX() {

@Override

**public** ResultadoEliminarCliente eliminarCliente(Cliente cliente) **throws** PersistenciaException {

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

**return** resultadoLogica;

}

**if**(excepcion **instanceof** PersistenciaException){

**throw** (PersistenciaException) excepcion;

}

**new** Integer("asd");

**return** **null**;

}

@Override

**public** ArrayList<Cliente> obtenerClientes() **throws** PersistenciaException {

ArrayList<Cliente> clientes = **new** ArrayList<>();

clientes.add(cliente);

**return** clientes;

}

};

PresentadorVentanas presentadorMock = **new** PresentadorVentanasMock(acepta);

AdministrarClienteController administrarClienteController = **new** AdministrarClienteController() {

@Override

**public** ResultadoControlador handleEliminar() {

tablaClientes.getSelectionModel().select(cliente);

**return** **super**.handleEliminar();

}

@Override

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

}

};

administrarClienteController.setCoordinador(coordinadorMock);

administrarClienteController.setPresentador(presentadorMock);

ControladorTest corredorTestEnJavaFXThread = **new** ControladorTest(AdministrarClienteController.***URLVista***, administrarClienteController);

administrarClienteController.setStage(corredorTestEnJavaFXThread.getStagePrueba());

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

@Override

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

*assertEquals*(resultadoControlador, administrarClienteController.handleEliminar());

}

};

**try**{

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

} **catch**(Throwable e){

**throw** **new** Exception(e);

}

}

Código de la pantalla de cargar inmueble buscado para alta y modificación de cliente en InmuebleBuscadoController.java

/\*\*

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

\*

\* Valida que se hayan insertado datos y los carga al inmueble buscado.

\* Si el cliente no posee inmueble buscado se crea uno, sino, se modifica el existente.

\*

\* Al finalizar regresa a la pantalla correspondiente, ya sea alta cliente o modificar cliente.

\*/

@FXML

**protected** **void** acceptAction() {

Double supeficieMinima = **null**;

Integer antiguedadMaxima = **null**;

Integer dormitoriosMinimos = **null**;

Integer bañosMinimos = **null**;

Double precioMaximo = **null**;

StringBuffer errores = **new** StringBuffer("");

//obtengo datos introducidos por el usuario

**try**{

supeficieMinima = Double.*valueOf*(textFieldSuperficie.getText().trim());

} **catch**(Exception e){

errores.append("Superficie incorrecta. Introduzca solo números y un punto para decimales.\n");

}

**try**{

precioMaximo = Double.*valueOf*(textFieldPrecio.getText().trim());

} **catch**(Exception e){

errores.append("Precio incorrecto. Introduzca solo números y un punto para decimales.\n");

}

**try**{

antiguedadMaxima = Integer.*valueOf*(textFieldAntiguedad.getText().trim());

} **catch**(Exception e){

errores.append("Antigüedad incorrecta. Introduzca solo números\n");

}

**try**{

dormitoriosMinimos = Integer.*valueOf*(textFieldDormitorios.getText().trim());

} **catch**(Exception e){

errores.append("Dormitorios incorrecto. Introduzca solo números\n");

}

**try**{

bañosMinimos = Integer.*valueOf*(textFieldBaños.getText().trim());

} **catch**(Exception e){

errores.append("Baños incorrecto. Introduzca solo números\n");

}

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

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

}

**else**{

//Si no hay errores se terminan de obtener los datos que no se verifican

Boolean propiedadHorizontal = checkBoxPropiedadHorizontal.isSelected();

Boolean garage = checkBoxGarage.isSelected();

Boolean patio = checkBoxPatio.isSelected();

Boolean piscina = checkBoxPiscina.isSelected();

Boolean aguaCorriente = checkBoxAguaCorriente.isSelected();

Boolean cloaca = checkBoxCloaca.isSelected();

Boolean gasNatural = checkBoxGasNatural.isSelected();

Boolean aguaCaliente = checkBoxAguaCaliente.isSelected();

Boolean telefono = checkBoxTelefono.isSelected();

Boolean lavadero = checkBoxLavadero.isSelected();

Boolean pavimento = checkBoxPavimento.isSelected();

Boolean casa = checkBoxCasa.isSelected();

Boolean departamento = checkBoxDepartamento.isSelected();

Boolean local = checkBoxLocal.isSelected();

Boolean galpon = checkBoxGalpon.isSelected();

Boolean terreno = checkBoxTerreno.isSelected();

Boolean quinta = checkBoxQuinta.isSelected();

InmuebleBuscado inmuebleBuscado = cliente.getInmuebleBuscado();

**if**(alta){ //si se está dando de alta el cliente, se crea un nuevo inmueble

inmuebleBuscado = **new** InmuebleBuscado();

cliente.setInmuebleBuscado(inmuebleBuscado);

inmuebleBuscado.setCliente(cliente);

}

//cargo los datos al inmueble

inmuebleBuscado.setSuperficieMin(supeficieMinima)

.setAntiguedadMax(antiguedadMaxima)

.setDormitoriosMin(dormitoriosMinimos)

.setBañosMin(bañosMinimos)

.setPrecioMax(precioMaximo)

.setPropiedadHorizontal(propiedadHorizontal)

.setGaraje(garage)

.setPatio(patio)

.setPiscina(piscina)

.setAguaCaliente(aguaCaliente)

.setAguaCorriente(aguaCorriente)

.setCloacas(cloaca)

.setGasNatural(gasNatural)

.setTelefono(telefono)

.setLavadero(lavadero)

.setPavimento(pavimento);

inmuebleBuscado.getLocalidades().clear();

inmuebleBuscado.getLocalidades().addAll(listaLocalidadesSeleccionadas);

inmuebleBuscado.getBarrios().clear();

inmuebleBuscado.getBarrios().addAll(listaBarriosSeleccionados);

**try**{

**for**(TipoInmueble tipo: coordinador.obtenerTiposInmueble()){

**switch**(tipo.getTipo()) {

**case** ***CASA***:

**if**(casa){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***DEPARTAMENTO***:

**if**(departamento){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***GALPON***:

**if**(galpon){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***LOCAL***:

**if**(local){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***QUINTA***:

**if**(quinta){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

**case** ***TERRENO***:

**if**(terreno){

inmuebleBuscado.getTiposInmueblesBuscados().add(tipo);

}

**break**;

}

}

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

presentador.presentarExcepcion(e, stage);

}

**if**(alta){ //si se está dando de alta vuelvo a la vista de alta cliente

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

controlador.setCliente(cliente);

}

**else**{ //si se está modificando vuelvo a la vista de modificar cliente

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

controlador.setClienteEnModificacion(cliente);

}

}

}

Código del test de la pantalla de cargar inmueble buscado para alta y modificación de cliente en InmuebleBuscadoControllerTest.java

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

ArrayList<Localidad> localidades = **new** ArrayList<>();

localidades.add(**new** Localidad("Federal", **new** Provincia("Entre ríos", **new** Pais("Argentina"))));

ArrayList<Barrio> barrios = **new** ArrayList<>();

Cliente clienteEnModificacion = **new** Cliente();

clienteEnModificacion.setInmuebleBuscado(**new** InmuebleBuscado());

clienteEnModificacion.getInmuebleBuscado().setCliente(clienteEnModificacion);

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

//String superficie,antiguedad,dormitorios,baños,precio,localidades,barrios,local,casa,departamento,terreno,galpon,quinta,propiedadHorizontal,garage,patio,piscina,aguaCorriente,cloaca,gasNatural,aguaCaliente,telefono,lavadero,pavimento,clienteNuevo,clienteEnModificacion,camposCorrectos,llamaAPresentadorVentanasPresentarError,llamaAPresentadorVentanasPresentarExcepcion,excepcion

//prueba correcta de inmueble nuevo

/\*0\*/ **new** Object[] { "30.0", "10", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **new** Cliente(), **null**, 1, 0, 0, **null** },

//prueba campo de texto incorrecto de inmueble nuevo

/\*1\*/ **new** Object[] { "30.0", "abc", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **new** Cliente(), **null**, 0, 1, 0, **null** },

//prueba campo texto vacio de inmueble nuevo

/\*2\*/ **new** Object[] { "30.0", "10", "", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **new** Cliente(), **null**, 0, 1, 0, **null** },

//prueba PersistenciaException de inmueble nuevo

/\*3\*/ **new** Object[] { "30.0", "10", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **new** Cliente(), **null**, 1, 0, 1, **new** SaveUpdateException(**new** Throwable()) },

//prueba correcta de inmueble en modificación

/\*4\*/ **new** Object[] { "30.0", "10", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **null**, clienteEnModificacion, 1, 0, 0, **null** },

//prueba campo de texto incorrecto de inmueble en modificación

/\*5\*/ **new** Object[] { "30.0", "abc", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **null**, clienteEnModificacion, 0, 1, 0, **null** },

//prueba campo texto vacio de inmueble en modificación

/\*6\*/ **new** Object[] { "30.0", "10", "", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **null**, clienteEnModificacion, 0, 1, 0, **null** },

//prueba PersistenciaException de inmueble en modificación

/\*7\*/ **new** Object[] { "30.0", "10", "2", "1", "3000000.0", localidades, barrios, **false**, **true**, **true**, **false**, **false**, **false**, **false**, **false**, **true**, **false**, **true**, **true**, **true**, **true**, **false**, **true**, **true**, **null**, clienteEnModificacion, 1, 0, 1, **new** SaveUpdateException(**new** Throwable()) }

};

}

/\*\*

\* Test para probar cargar un inmueble buscado cuando se está creando un cliente y cuando se está modificando un cliente

\* al momento en que el usuario presiona el botón de aceptar

\*

\* **@param** superficie

\* introducida por el usuario

\* **@param** antiguedad

\* introducida por el usuario

\* **@param** dormitorios

\* introducidos por el usuario

\* **@param** baños

\* introducidos por el usuario

\* **@param** precio

\* introducido por el usuario

\* **@param** localidades

\* introducidas por el usuario

\* **@param** barrios

\* introducidos por el usuario

\* **@param** local

\* si es seleccionado el checkbox por el usuario

\* **@param** casa

\* si es seleccionado el checkbox por el usuario

\* **@param** departamento

\* si es seleccionado el checkbox por el usuario

\* **@param** terreno

\* si es seleccionado el checkbox por el usuario

\* **@param** galpon

\* si es seleccionado el checkbox por el usuario

\* **@param** quinta

\* si es seleccionado el checkbox por el usuario

\* **@param** propiedadHorizontal

\* si es seleccionado el checkbox por el usuario

\* **@param** garage

\* si es seleccionado el checkbox por el usuario

\* **@param** patio

\* si es seleccionado el checkbox por el usuario

\* **@param** piscina

\* si es seleccionado el checkbox por el usuario

\* **@param** aguaCorriente

\* si es seleccionado el checkbox por el usuario

\* **@param** cloaca

\* si es seleccionado el checkbox por el usuario

\* **@param** gasNatural

\* si es seleccionado el checkbox por el usuario

\* **@param** aguaCaliente

\* si es seleccionado el checkbox por el usuario

\* **@param** telefono

\* si es seleccionado el checkbox por el usuario

\* **@param** lavadero

\* si es seleccionado el checkbox por el usuario

\* **@param** pavimento

\* si es seleccionado el checkbox por el usuario

\* **@param** clienteNuevo

\* cliente que se está creando

\* **@param** clienteEnModificacion

\* cliente que se está modificando

\* **@param** camposCorrectos

\* 1 si los campos son correctos, 0 si no

\* **@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** excepcion

\* excepción que se simula lanzar desde la capa lógica

\* **@throws** Throwable

\*/

@Test

@Parameters

**public** **void** testCargarInmuebleBuscado(String superficie,

String antiguedad,

String dormitorios,

String baños,

String precio,

ArrayList<Localidad> localidades,

ArrayList<Barrio> barrios,

Boolean local,

Boolean casa,

Boolean departamento,

Boolean terreno,

Boolean galpon,

Boolean quinta,

Boolean propiedadHorizontal,

Boolean garage,

Boolean patio,

Boolean piscina,

Boolean aguaCorriente,

Boolean cloaca,

Boolean gasNatural,

Boolean aguaCaliente,

Boolean telefono,

Boolean lavadero,

Boolean pavimento,

Cliente clienteNuevo,

Cliente clienteEnModificacion,

Integer camposCorrectos,

Integer llamaAPresentadorVentanasPresentarError,

Integer llamaAPresentadorVentanasPresentarExcepcion,

Exception excepcion) **throws** Throwable {

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

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

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

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

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

AltaClienteController altaClienteControllerMock = *mock*(AltaClienteController.**class**);

ModificarClienteController modificarClienteControllerMock = *mock*(ModificarClienteController.**class**);

*when*(coordinadorMock.obtenerPaises()).thenReturn(**new** ArrayList<Pais>());

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

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

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

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

}

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

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

}

*doNothing*().when(altaClienteControllerMock).setCliente(*any*());

*doNothing*().when(modificarClienteControllerMock).setClienteEnModificacion(*any*());

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

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

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***CASA***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***DEPARTAMENTO***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***GALPON***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***LOCAL***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***QUINTA***));

tipos.add(**new** TipoInmueble(TipoInmuebleStr.***TERRENO***));

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

}

**else**{

*when*(coordinadorMock.obtenerTiposInmueble()).thenThrow(excepcion);

}

InmuebleBuscadoController inmuebleBuscadoController = **new** InmuebleBuscadoController() {

@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**.textFieldAntiguedad.setText(antiguedad);

**this**.textFieldBaños.setText(baños);

**this**.textFieldDormitorios.setText(dormitorios);

**this**.textFieldPrecio.setText(precio);

**this**.textFieldSuperficie.setText(superficie);

**this**.listaBarriosSeleccionados.clear();

**this**.listaBarriosSeleccionados.addAll(barrios);

**this**.listaLocalidadesSeleccionadas.clear();

**this**.listaLocalidadesSeleccionadas.addAll(localidades);

**this**.checkBoxAguaCaliente.setSelected(aguaCaliente);

**this**.checkBoxAguaCorriente.setSelected(aguaCorriente);

**this**.checkBoxCasa.setSelected(casa);

**this**.checkBoxCloaca.setSelected(cloaca);

**this**.checkBoxDepartamento.setSelected(departamento);

**this**.checkBoxGalpon.setSelected(galpon);

**this**.checkBoxGarage.setSelected(garage);

**this**.checkBoxGasNatural.setSelected(gasNatural);

**this**.checkBoxLavadero.setSelected(lavadero);

**this**.checkBoxLocal.setSelected(local);

**this**.checkBoxPatio.setSelected(patio);

**this**.checkBoxPavimento.setSelected(pavimento);

**this**.checkBoxPiscina.setSelected(piscina);

**this**.checkBoxPropiedadHorizontal.setSelected(propiedadHorizontal);

**this**.checkBoxQuinta.setSelected(quinta);

**this**.checkBoxTelefono.setSelected(telefono);

**this**.checkBoxTerreno.setSelected(terreno);

**super**.acceptAction();

}

@Override

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

}

@Override

**public** **void** setCliente(Cliente cliente) {

**this**.cliente = cliente;

InmuebleBuscado inmueble = cliente.getInmuebleBuscado();

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

**this**.alta = **false**;

}

**else**{

**this**.alta = **true**;

}

}

};

ControladorTest corredorTestEnJavaFXThread =

**new** ControladorTest(InmuebleBuscadoController.***URLVista***, inmuebleBuscadoController);

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

@Override

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

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

inmuebleBuscadoController.setCliente(clienteNuevo);

}

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

inmuebleBuscadoController.setCliente(clienteEnModificacion);

}

inmuebleBuscadoController.acceptAction();

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

Mockito.*verify*(altaClienteControllerMock, *times*(camposCorrectos)).setCliente(clienteNuevo);

}

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

Mockito.*verify*(modificarClienteControllerMock, *times*(camposCorrectos)).setClienteEnModificacion(clienteEnModificacion);

}

Mockito.*verify*(coordinadorMock, *times*(camposCorrectos)).obtenerTiposInmueble();

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

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

}

};

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

}

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

Código del archivo en GestorCliente.java

/\*\*

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

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

\*

\* Modificada en TaskCard 27 de la iteración 2

\*

\* **@param** cliente

\* cliente a crear

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

\*

\* **@throws** PersistenciaException

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

\* **@throws** GestionException

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

\*/

**public** ResultadoCrearCliente crearCliente(Cliente cliente) **throws** PersistenciaException, GestionException {

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

// valida formato de datos

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

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

}

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

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

}

**if**(!validador.validarTelefono(cliente.getTelefono())){

errores.add(ErrorCrearCliente.***Formato\_Telefono\_Incorrecto***);

}

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

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

}

**if**(!validador.validarEmail(cliente.getCorreo())){

errores.add(ErrorCrearCliente.***Formato\_Correo\_Incorrecto***);

}

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

Cliente clienteAuxiliar = persistidorCliente.obtenerCliente(**new** FiltroCliente(cliente.getTipoDocumento().getTipo(),

cliente.getNumeroDocumento()));

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

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

errores.add(ErrorCrearCliente.***Ya\_Existe\_Cliente***);// 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***)){

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

}

}

persistidorCliente.guardarCliente(cliente);

}

**return** **new** ResultadoCrearCliente(errores.toArray(**new** ErrorCrearCliente[0]));

}

/\*\*

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

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

\*

\* Modificada en TaskCard 27 de la iteración 2

\*

\* **@param** cliente

\* cliente a modificar

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

\*

\* **@throws** PersistenciaException

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

\*/

**public** ResultadoModificarCliente modificarCliente(Cliente cliente) **throws** PersistenciaException {

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

// valida formato de datos

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

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

}

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

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

}

**if**(!validador.validarTelefono(cliente.getTelefono())){

errores.add(ErrorModificarCliente.***Formato\_Telefono\_Incorrecto***);

}

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

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

}

**if**(!validador.validarEmail(cliente.getCorreo())){

errores.add(ErrorModificarCliente.***Formato\_Correo\_Incorrecto***);

}

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

Cliente clienteAuxiliar = persistidorCliente.obtenerCliente(**new** FiltroCliente(cliente.getTipoDocumento().getTipo(),

cliente.getNumeroDocumento()));

**if**(clienteAuxiliar != **null** && !cliente.equals(clienteAuxiliar)){

errores.add(ErrorModificarCliente.***Otro\_Cliente\_Posee\_Mismo\_Documento\_Y\_Tipo***);

}

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

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

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

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

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

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

}

}

}

persistidorCliente.modificarCliente(cliente);

}

**return** **new** ResultadoModificarCliente(errores.toArray(**new** ErrorModificarCliente[0]));

}

/\*\*

\* Se encarga de validar que exista el cliente 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** cliente

\* cliente 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** ResultadoEliminarCliente eliminarCliente(Cliente cliente) **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***)){

cliente.setEstado(e);

}

}

persistidorCliente.modificarCliente(cliente);

**return** **new** ResultadoEliminarCliente();

}

Código de los test en GestorCliente.java

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

//Se carga cliente 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***);

*cliente* = **new** Cliente();

InmuebleBuscado inm = **new** InmuebleBuscado(*cliente*, 4567.2, 345.9, 10, 2, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

*cliente*.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setTelefono("1234-1234")

.setCorreo("asdf@asf.com")

.setInmuebleBuscado(inm);

//Parámetros de JUnitParams

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

//resValNombre,resValApellido,resValDocumento,resValTelefono,resValCorreo,resObtenerCliente,guardar,resultadoCrearClienteEsperado

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

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

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

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

/\*4\*/ **new** Object[] { **true**, **true**, **true**, **false**, **true**, **null**, 0, ***resultadoCrearTelefonoIncorrecto*** },

/\*5\*/ **new** Object[] { **true**, **true**, **true**, **true**, **false**, **null**, 0, ***resultadoCrearCorreoIncorrecto*** },

/\*6\*/ **new** Object[] { **false**, **false**, **true**, **true**, **true**, **null**, 0, **new** ResultadoCrearCliente(ErrorCrearCliente.***Formato\_Nombre\_Incorrecto***, ErrorCrearCliente.***Formato\_Apellido\_Incorrecto***) },

/\*7\*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, *cliente*, 0, ***resultadoCrearYaExiste*** }

};

}

/\*\*

\* Test para probar el método crearCliente

\*

\* **@param** resValNombre

\* resultado devuelto por el mock validador de formato al validar nombre

\* **@param** resValApellido

\* resultado devuelto por el mock validador de formato al validar apellido

\* **@param** resValDocumento

\* resultado devuelto por el mock validador de formato al validar documento

\* **@param** resValTelefono

\* resultado devuelto por el mock validador de formato al validar teléfono

\* **@param** resValCorreo

\* resultado devuelto por el mock validador de formato al validar correo

\* **@param** resObtenerCliente

\* resultado devuelto por el mock persistidor al obtener cliente

\* **@param** guardar

\* 1 si se espera que se ejecute el guardar hacia capa de persistencia, 0 si no

\* **@param** resultadoCrearClienteEsperado

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

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testCrearCliente(Boolean resValNombre, Boolean resValApellido, Boolean resValDocumento, Boolean resValTelefono, Boolean resValCorreo, Cliente resObtenerCliente, Integer guardar, ResultadoCrearCliente resultadoCrearClienteEsperado) **throws** Exception {

//Inicialización de los mocks

ClienteService clienteServiceMock = *mock*(ClienteService.**class**);

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

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

GestorCliente gestorCliente = **new** GestorCliente() {

{

**this**.persistidorCliente = clienteServiceMock;

**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(*cliente*.getNombre())).thenReturn(resValNombre);

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

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

*when*(validadorFormatoMock.validarTelefono(*cliente*.getTelefono())).thenReturn(resValTelefono);

*when*(validadorFormatoMock.validarEmail(*cliente*.getCorreo())).thenReturn(resValCorreo);

*when*(clienteServiceMock.obtenerCliente(*any*())).thenReturn(resObtenerCliente);

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

*doNothing*().when(clienteServiceMock).guardarCliente(*cliente*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoCrearCliente resultadoCrearCliente = gestorCliente.crearCliente(*cliente*);

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

*assertEquals*(resultadoCrearClienteEsperado, resultadoCrearCliente);

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

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

}

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

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

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

*verify*(validadorFormatoMock).validarTelefono(*cliente*.getTelefono());

*verify*(validadorFormatoMock).validarEmail(*cliente*.getCorreo());

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

*verify*(clienteServiceMock, *times*(guardar)).guardarCliente(*cliente*);

}

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

//Se carga cliente 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***);

*clienteM* = **new** Cliente() {

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

**this**.id = id;

**return** **this**;

}

{

**this**.setId(1)

.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setEstado(est);

}

};

InmuebleBuscado inm = **new** InmuebleBuscado(*clienteM*, 4567.2, 345.9, 10, 2, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

*clienteM*.setInmuebleBuscado(inm);

*clienteM2* = **new** Cliente() {

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

**this**.id = id;

**return** **this**;

}

{

**this**.setId(2)

.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc);

}

};

InmuebleBuscado inm2 = **new** InmuebleBuscado(*clienteM2*, 4567.2, 345.9, 10, 2, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

*clienteM2*.setInmuebleBuscado(inm2);

//Parámetros de JUnitParams

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

//resValNombre,resValApellido,resValDocumento,resValTelefono,resValCorreo,resObtenerCliente,modificar,resultadoModificarClienteEsperado

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

/\*1\*/ **new** Object[] { **false**, **true**, **true**, **true**, **true**, *clienteM*, 0, ***resultadoModificarNombreIncorrecto*** },

/\*2\*/ **new** Object[] { **true**, **false**, **true**, **true**, **true**, *clienteM*, 0, ***resultadoModificarApellidoIncorrecto*** },

/\*3\*/ **new** Object[] { **true**, **true**, **false**, **true**, **true**, *clienteM*, 0, ***resultadoModificarDocumentoIncorrecto*** },

/\*4\*/ **new** Object[] { **true**, **true**, **true**, **false**, **true**, *clienteM*, 0, ***resultadoModificarTelefonoIncorrecto*** },

/\*5\*/ **new** Object[] { **true**, **true**, **true**, **true**, **false**, *clienteM*, 0, ***resultadoModificarCorreoIncorrecto*** },

/\*6\*/ **new** Object[] { **false**, **false**, **true**, **true**, **true**, *clienteM*, 0, **new** ResultadoModificarCliente(ErrorModificarCliente.***Formato\_Nombre\_Incorrecto***, ErrorModificarCliente.***Formato\_Apellido\_Incorrecto***) },

/\*7\*/ **new** Object[] { **true**, **true**, **true**, **true**, **true**, *clienteM2*, 0, ***resultadoModificarYaSePoseeMismoDocumento*** }

};

}

/\*\*

\* Test para probar el método modificarCliente

\*

\* **@param** resValNombre

\* resultado devuelto por el mock validador de formato al validar nombre

\* **@param** resValApellido

\* resultado devuelto por el mock validador de formato al validar apellido

\* **@param** resValDocumento

\* resultado devuelto por el mock validador de formato al validar documento

\* **@param** resValTelefono

\* resultado devuelto por el mock validador de formato al validar teléfono

\* **@param** resValCorreo

\* resultado devuelto por el mock validador de formato al validar correo

\* **@param** resObtenerCliente

\* resultado devuelto por el mock persistidor al obtener cliente

\* **@param** modificar

\* 1 si se espera que se ejecute el modificar hacia capa de persistencia, 0 si no

\* **@param** resultadoModificarClienteEsperado

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

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testModificarCliente(Boolean resValNombre, Boolean resValApellido, Boolean resValDocumento, Boolean resValTelefono, Boolean resValCorreo, Cliente resObtenerCliente, Integer modificar, ResultadoModificarCliente resultadoModificarClienteEsperado) **throws** Exception {

//Inicialización de los mocks

ClienteService clienteServiceMock = *mock*(ClienteService.**class**);

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

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

//Clase anónima necesaria para inyectar dependencias hasta que funcione Spring

GestorCliente gestorCliente = **new** GestorCliente() {

{

**this**.persistidorCliente = clienteServiceMock;

**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(*clienteM*.getNombre())).thenReturn(resValNombre);

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

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

*when*(validadorFormatoMock.validarTelefono(*clienteM*.getTelefono())).thenReturn(resValTelefono);

*when*(validadorFormatoMock.validarEmail(*clienteM*.getCorreo())).thenReturn(resValCorreo);

*when*(clienteServiceMock.obtenerCliente(*any*())).thenReturn(resObtenerCliente);

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

*doNothing*().when(clienteServiceMock).modificarCliente(*clienteM*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoModificarCliente resultadoModificarCliente = gestorCliente.modificarCliente(*clienteM*);

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

*assertEquals*(resultadoModificarClienteEsperado, resultadoModificarCliente);

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

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

}

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

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

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

*verify*(validadorFormatoMock).validarTelefono(*clienteM*.getTelefono());

*verify*(validadorFormatoMock).validarEmail(*clienteM*.getCorreo());

*verify*(clienteServiceMock, *times*(modificar)).modificarCliente(*clienteM*);

}

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

//Se carga cliente 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***);

*clienteE* = **new** Cliente();

InmuebleBuscado inm = **new** InmuebleBuscado(*cliente*, 4567.2, 345.9, 10, 2, 1, **true**, **true**, **false**, **false**, **true**, **true**, **true**, **true**, **true**, **true**, **false**);

*clienteE*.setNombre("Juan")

.setApellido("Pérez")

.setNumeroDocumento("12345678")

.setTipoDocumento(doc)

.setEstado(est)

.setInmuebleBuscado(inm);

//Parámetros de JUnitParams

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

//resObtenerCliente,eliminar,resultadoEliminarClienteEsperado

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

};

}

/\*\*

\* Test para probar el método eliminarCliente

\*

\* **@param** resObtenerCliente

\* resultado devuelto por el mock persistidor al obtener cliente

\* **@param** eliminar

\* 1 si se espera que se ejecute el modificar hacia capa de persistencia (ya que es baja lógica), 0 si no

\* **@param** resultadoEliminarClienteEsperado

\* resultado que se espera retorne el método a probar

\* **@throws** Exception

\*/

@Test

@Parameters

**public** **void** testEliminarCliente(Cliente resObtenerCliente, Integer eliminar, ResultadoEliminarCliente resultadoEliminarClienteEsperado) **throws** Exception {

//Inicialización de los mocks

ClienteService clienteServiceMock = *mock*(ClienteService.**class**);

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

//Clase anónima necesaria para inyectar dependencias hasta que funcione Spring

GestorCliente gestorCliente = **new** GestorCliente() {

{

**this**.persistidorCliente = clienteServiceMock;

**this**.gestorDatos = gestorDatosMock;

}

};

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

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

//Setear valores esperados a los mocks

*when*(clienteServiceMock.obtenerCliente(*any*())).thenReturn(resObtenerCliente);

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

*doNothing*().when(clienteServiceMock).modificarCliente(*clienteE*); //Para métodos void la sintaxis es distinta

//Llamar al método a testear

ResultadoEliminarCliente resultadoEliminarCliente = gestorCliente.eliminarCliente(*clienteE*);

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

*assertEquals*(resultadoEliminarClienteEsperado, resultadoEliminarCliente);

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

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

}

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

*verify*(clienteServiceMock, *times*(eliminar)).modificarCliente(*clienteE*);

}

## Iteración 2: