# 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();

}

#### Taskcard 6 Persistidor vendedor

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

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

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