Proyecto Adalid, Plataforma jee1
Guía de Meta-Programación

Tabla de Contenido

¿Qué es Adalid?

Adalid es un sistema de procesamiento de plantillas de código abierto (*open-source*) diseñado específicamente para apoyar el desarrollo de aplicaciones de software. Utilizando un motor de plantillas, Adalid combina un modelo de datos abstracto para producir gran variedad de documentos, típicamente código fuente. También puede generar documentación de la aplicación, datos de prueba, archivos de control de acceso y otros archivos de configuración. Además, a diferencia de la mayoría de los generadores de código actualmente disponibles en el mercado, Adalid ha sido diseñado para permitirle construir sus propios generadores, dándole total control sobre el proceso de generación y los documentos producidos.

Los generadores de código han probado ser efectivos en la reducción de costos del desarrollo de software y en la mejora de su calidad. Estas herramientas son particularmente útiles para hacer cumplir patrones y estándares, reducir el tiempo y el esfuerzo requerido para la codificación y las pruebas, garantizar consistencia y minimizar errores. En particular, un sistema de procesamiento de plantillas, como Adalid, también permite escalar el mejor talento de la organización, ya que las habilidades de las personas encargadas de preparar y mantener los modelos y plantillas se reflejan en cada uno de los documentos generados. Además, dado que Adalid ofrece la capacidad de generar código para múltiples plataformas, cuando se trata de una compañía de software, Adalid permite ampliar el mercado de los productos de la empresa.

Adalid es una adaptación libre y simplificada de la arquitectura dirigida por modelos conocida como MDA, por las siglas en inglés de *Model-Driven Architecture*. El modelo de datos de Adalid es un modelo de los componentes de la aplicación y su funcionalidad, es independiente de la plataforma y conocido como PIM, por las siglas en inglés de *Platform Independent Model*. La mayoría de los procesadores de plantillas utilizan una base de datos o archivos de texto, en formatos tales como XML, para almacenar el modelo de datos. En lugar de esto, Adalid provee una librería de clases Java de artefactos que se utilizan para definir los elementos del modelo; y los modelos definidos a su vez se almacenan como librerías de clases Java. Los artefactos de Adalid se extienden para traducir el PIM en uno o más modelos específicos de la plataforma, conocido como PSM por las siglas en inglés de *Platform Specific Model*. Finalmente, los PSM se combinan con plantillas para generar código fuente para cualquier plataforma de software.

Adalid utiliza *Apache Velocity Engine* como motor de plantillas, de manera que las plantillas pueden ser elaboradas utilizando el poderoso lenguaje de plantillas de Velocity. La facilidad de extender el modelo de datos y de añadir nuevas plantillas permite virtualmente la generación de cualquier clase de documentos.

La extensibilidad y la reusabilidad son principios fundamentales en Adalid. Como cualquier otra aplicación Java *open-source*, agregar nuevas funciones y modificar las existentes se puede lograr simplemente extendiendo los artefactos de Adalid. Y, como se mencionó antes, los modelos de datos son definidos y almacenados como librerías de clases Java, de manera que también son fácilmente reutilizados, extendidos, controlados (con cualquier sistema de control de versiones) e incluso organizados en una jerarquía de modelos.

Por otra parte, Adalid no es una herramienta de modelaje. Actualmente, los modelos pueden ser definidos utilizando cualquier IDE que soporte Java, como Eclipse y NetBeans. Interfaces con herramientas de modelaje, como *Visual Paradigm* y *Magic Draw*, serían incluidas en una próxima versión de Adalid, de manera que sea posible importar modelos UML. Una vez importados, estos modelos podrán ser extendidos (al igual que los modelos definidos utilizando un IDE) de manera que los detalles específicos de plataformas, los cuales usualmente no están presentes en modelos UML, puedan ser añadidos.

¿Qué es jee1?

A diferencia de la mayoría de los generadores de código disponibles en la actualidad, Adalid está diseñado para que usted pueda construir sus propios generadores, lo que le da control total sobre el proceso de generación y los documentos resultantes. Usando artefactos y plantillas específicas es posible generar código fuente para cualquier plataforma de software, o tantas variantes como usted necesite para una plataforma específica.

Para ayudarle a comenzar, Adalid se distribuye con todos los artefactos, modelos y plantillas necesarias para generar una aplicación Java que ejecuta con servidores de aplicaciones GlassFish y JBoss AS, y con el servidor de bases de datos PostgreSQL. Estos artefactos, modelos y plantillas se denominan colectivamente **plataforma jee1** o simplemente **jee1**. Esta plataforma sirve tanto de ejemplo de la capacidad de Adalid como de base para el desarrollo de otras plataformas.

Con aproximadamente 800 plantillas, **jee1** permite crear aplicaciones Web, transaccionales, centradas en bases de datos, en un ambiente Java Enterprise Edition. Entre los componentes más relevantes de **jee1** tenemos los generadores de:

- scripts de base de datos, configuración del ambiente e instalación
- interfaz de usuario Web
- archivos e informes
- reglas de negocio e interfaces de servicio de procesos
- control de acceso, gestión de tareas y auditoría.

Scripts de base de datos, configuración del ambiente e instalación

Podría sonar anticuado, pero la generación de scripts de base de datos sigue siendo una tarea esencial. La capacidad y el rendimiento de los sistemas de información dependen en gran medida del diseño de base de datos y de cómo se implementa dicho diseño. Una buena parte de las plantillas de **jee1** son plantillas de scripts para la definición de objetos de base de datos. Junto con las plantillas para generar scripts de objetos básicos, tales como tablas, índices, secuencias, restricciones, etc., hay plantillas avanzadas para generar disparadores, funciones de reglas de negocio, funciones de servicio de procesos de negocio, vistas actualizables de múltiples tablas, funciones de control de acceso, gestión de tareas, auditoría, generación de meta-data y de datos de prueba, etc.

Además de los scripts de base de datos, **jee1** también incluye plantillas de scripts para la configuración del servidor de aplicaciones GlassFish y para la instalación y administración de la aplicación generada en ambientes de prueba y producción, tanto Linux como Windows.

Interfaz de usuario Web

Con **jee1** se pueden generar diferentes tipos de páginas web para cada clase de entidad. Las páginas de de consulta y registro (CRUD) pueden ser cualquier combinación de las presentaciones (Tabular, Árbol y Forma, con o sin pestañas), los formatos (Independiente, Maestro/Detalle) y los modos disponibles (Lectura y Escritura, Sólo Lectura). Existe además una plantilla de página para ejecutar procesos de negocio y otros procesos, tales como generación de informes y archivos, carga y descarga de archivos, etc.

También se generan las reglas de navegación entre las páginas generadas. Una página de menú y una página de "Favoritos" están disponibles. Por lo general, cada usuario mantiene su propia lista de favoritos, pero los roles definidos mediante el módulo de control de acceso también pueden tener estas listas, de modo que puedan ser compartidos por todos los usuarios asignados al rol.

Cada página generada utiliza archivos de recursos (*resource bundles*) para implementar una interfaz de usuario multilingüe. Las páginas se presentan a cada usuario en el idioma su elección, siempre que el archivo correspondiente esté disponible. En este punto **jee1** se distribuye sólo con archivos en español e inglés. Traducir estos archivos es todo lo que necesita hacer para agregar idiomas adicionales a su aplicación.

Cada página de presentación Tabular permite hacer consultas a la base de datos a través de campos de búsqueda predefinidos o de filtros definidos por el usuario final en tiempo de ejecución. Los resultados de la consulta se almacenan en un caché de memoria, por lo tanto, se reduce la transferencia de datos entre el servidor de aplicaciones y el servidor de base de datos. Los resultados de la consulta se pueden almacenar de forma permanente, ya sea como un archivo PDF o como un archivo CSV, que puede descargarse para ser procesado con la aplicación de hoja de cálculo de su elección. Los resultados de la consulta se pueden guardar como un informe detallado, resumido o gráfico, utilizando la vista predeterminada de la clase de entidad de la página o vistas definidas por el usuario final. Además de permitir seleccionar las propiedades de la entidad que se muestran en el informe, las vistas definidas por el usuario final también permiten especificar operadores de agregación, criterios de agrupación, criterios de ordenamiento, etc.

Todas las páginas CRUD generadas cumplen con las reglas de negocio especificadas en el modelo. La verificación de las reglas se lleva a cabo lo antes posible, mediante JavaScript incrustado en la página y/o un componente de Java en el servidor de aplicaciones y/o una función de SQL en el servidor de base de datos.

Las referencias a otras entidades en una página web se manejan ya sea a través de listas desplegables o de páginas de búsqueda. Estas páginas de búsqueda suelen ser de presentación Tabular y de Sólo Lectura; en ellas el usuario puede realizar consultas a través de campos de búsqueda predefinidos o de filtros definidos por el usuario final.

Como se mencionó antes, las páginas se generan para cada clase de entidad, no para cada tabla de la base de datos. Esta diferencia es particularmente importante en la generación de páginas para entidades en una jerarquía de clases para la que ha especificado una estrategia de herencia distinta a TABLE_PER_CLASS.

Archivos e informes

Con **jee1** se pueden generar diferentes tipos de archivos e informes para cada clase de entidad. Los archivos se generan en formato CSV, y pueden ser detallados o resumidos. Los informes de generan en formato PDF, y pueden ser detallados, resumidos o gráficos. Tanto para los archivos como para los informes, el resultado de la correspondiente consulta se puede acotar mediante campos de búsqueda predefinidos.

Cada informe generado utiliza archivos de recursos (*resource bundles*) para implementar una interfaz de usuario multilingüe. Los informes se presentan a cada usuario en el idioma su elección, siempre que el archivo correspondiente esté disponible. En este punto **jee1** se distribuye sólo con archivos en español e inglés. Traducir estos archivos es todo lo que necesita hacer para agregar idiomas adicionales a su aplicación.

Reglas de negocio e interfaces de servicio de procesos

Como parte de su modelo, puede definir reglas de negocio de varias maneras. Existen métodos sencillos para definir reglas simples, como campos opcionales o requeridos, campos únicos, valores iniciales y por defecto, valores mínimos y máximos, longitudes mínimas y máximas de una cadena de caracteres, patrones de cadenas de caracteres, etc.

Adalid proporciona también métodos más sofisticados para definir sus reglas de negocio. Ellos son: expresiones, transiciones y disparadores.

Hay varias clases de expresiones, por ejemplo: Comparación, Condicional, Escalar, Agregados de datos y de filas. Y en cada una de estas clases de expresiones existe una expresión para cada tipo de datos, por ejemplo: booleanos, caracteres, datos numéricos y temporales. Pero es con expresiones booleanas que se logra definir la mayor parte de las reglas de negocio. Adalid ofrece un conjunto de operadores booleanos en el que además de los operadores fundamentales AND, OR y NOT, se encuentran otros como NAND, NOR, XOR, XNOR e IMPLIES.

Las expresiones booleanas se utilizan para definir las restricciones, segmentos y estados de cada clase de entidad. Las restricciones son expresiones que deben ser verdaderas en todas las instancias de la clase. Los segmentos son expresiones que son verdaderas sólo en algunos casos. Y los estados son un tipo especial de segmentos, el tipo que le permite definir transiciones, que son caminos válidos entre dos estados. Al definir las operaciones de la clase de entidad pueden especificar las transiciones realizadas por cada operación, y así se define una máquina de estados. También puede definir directamente restricciones para las operaciones de la entidad. Por último, es posible utilizar un disparador para definir las operaciones que se deben realizar cuando una instancia de la entidad llega a un estado específico.

En cuanto a los procesos de negocio, **jee1** no genera el código de proceso en sí, sino sus interfaces de servicio, es decir el código necesario para llegar donde se encuentra el código de negocio, ejecutarlo y devolver una respuesta. Los procesos de negocio se codifican ya sea como un método de un *Enterprise Java Bean* (EJB) o como una función de SQL. En el modelo se define la forma cómo se debe ejecutar el proceso de negocio. Podría ser sincrónica o asincrónica. La ejecución sincrónica se implementa a través de EJB, y la ejecución asíncrona a través de *Java Message Service* (JMS), que es la manera ortodoxa de implementar múltiples hilos (*multi-threading*) en la plataforma Java EE.

Control de acceso, gestión de tareas y auditoría

El módulo de control de acceso de **jee1** se basa en el modelo de control de acceso por roles, donde se asignan funciones (operaciones) y usuarios a roles, y los permisos de un usuario son la suma de los permisos de los roles a los que se ha asignado el usuario. Con **jee1** se extiende el modelo para incorporar autorizaciones de lectura y/o escritura a nivel de campos y también los conceptos de personalización y segmentación. La personalización consiste en definir un propietario para una clase de entidad y luego otorgar permisos para ejecutar una función solo sobre las instancias de la clase que pertenezcan al usuario. La segmentación consiste en definir conjuntos de segmentos para una clase de entidad y luego otorgar permisos para ejecutar una función solo sobre las instancias de la clase que pertenezcan al conjunto seleccionado. Además, al asignar una función a un rol se puede especificar si la función es o no una tarea de los usuarios asignados al rol; esto permite diferenciar quien puede y quien “puede y debe” ejecutar una función. El módulo provee un mecanismo de autenticación, muy útil para ambientes de desarrollo y pruebas. En el ambiente de producción se debe utilizar un mecanismo de autenticación estándar, tal como LDAP.

La gestión de tareas consiste en llevar el control de las tareas que cada usuario debe realizar. Mediante el uso de restricciones, estados y disparadores, se pueden especificar las operaciones (funciones) que deben ejecutarse cuando una instancia de una clase de entidad llega a un estado determinado. El gestor de tareas evalúa el estado de la instancia después de agregarla o actualizarla, y también después de ejecutar una función de negocio sobre la instancia. Se evalúan todos los estados finales de las transiciones definidas para la operación. Si la entidad alcanzó un estado para el cual existe un disparador, entonces el gestor procede a notificar a los usuarios que deben ejecutar la operación especificada en el disparador. El gestor envía una notificación a todos aquellos usuarios que "pueden y deben" ejecutar la operación. Además, al notificar se toma en cuenta si la autorización es personalizada y/o segmentada. Cuando el usuario recibe una notificación puede asumir la responsabilidad de ejecutar la tarea. Automáticamente el gestor de tareas oculta la notificación de los demás usuarios que "pueden y deben" ejecutar la tarea. Si después de asumir una tarea, un usuario decide que no puede ejecutarla, entonces puede abandonarla. Automáticamente el gestor de tareas vuelve a mostrar la notificación a todos los usuarios que "pueden y deben" ejecutar la tarea. Además, si cuenta con la autorización correspondiente, un usuario podrá asignar o quitar la responsabilidad de ejecutar una tarea a otro

usuario; y cancelar la ejecución de una tarea cuando no sea posible o no sea necesario realizarla. El gestor de tareas también actualiza automáticamente la condición de la tarea a "ejecutada" cuando se ejecuta exitosamente la correspondiente operación.

Por último, pero no menos importante, el módulo de Auditoría mantiene un registro de la ejecución de todas las funciones, procesos e informes de la aplicación y de todos los archivos cargados (*uploaded*) al servidor.

Meta-programación

Adalid provee una librería de clases Java de artefactos que se utilizan para definir los elementos del modelo de la aplicación; y los modelos definidos a su vez se almacenan como librerías de clases Java. De manera que el modelo se define mediante una técnica conocida como **meta-programación**, utilizando Java como **meta-lenguaje**. En términos muy simples, meta-programación es escribir programas que van a escribir otros programas.

El uso de Java como meta-lenguaje es una característica especial de Adalid que permite que los modelos se puedan reutilizar, extender, controlar (con cualquier sistema de control de versiones) e incluso organizar en una jerarquía de modelos.

Además, los artefactos de Adalid integran conceptos que normalmente se encuentran en fuentes independientes, tales como diagramas de clases, diagramas de estado, diagramas de entidad-relación, etc., con el fin de simplificar el proceso de definición del modelo de la aplicación.

El proceso de generación de aplicaciones con Adalid es bastante sencillo y se puede resumir en los siguientes pasos:

1. Definir las entidades de la aplicación, como se explica en el capítulo Meta-programación de entidades, y sus operaciones, como se explica en el capítulo Meta-programación de operaciones de negocio.
2. Definir los módulos de la aplicación y un proyecto maestro que incluya todos los módulos, como se explica en el capítulo Meta-programación de proyectos.
3. Ejecutar el proyecto maestro. Esta acción genera la aplicación.

Una vez generada la aplicación, puede crear su correspondiente base de datos y configurar el servidor de aplicaciones, como se explica en el apéndice 3 y en el apéndice 4, respectivamente. Después de crear la base de datos y configurar el servidor puede ejecutar la aplicación.

Para refinar la aplicación, agregue los artefactos necesarios (por ejemplo, clases EJB y stored-procedures para implementar reglas y operaciones de negocio, informes personalizados, etc.), teniendo el cuidado de no modificar ni eliminar los artefactos generados por Adalid, de manera que posteriormente pueda incorporar cambios al modelo y regenerar la aplicación tantas veces como sea necesario. Para más información, consulte el apéndice 5.

Meta-programación de entidades

Cada entidad de su aplicación se debe definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **Entity**.

Cada entidad persistente de su aplicación se debe definir mediante una clase Java que extienda, directa o indirectamente, la clase **AbstractPersistentEntity**, la implementa la interfaz **Entity**.

Si utiliza la plataforma **jee1**, se puede definir extendiendo **PersistentEntityBase**, que a su vez extiende **AbstractPersistentEntity** e incluye las propiedades **id** y **version**, correspondientes a la clave primaria y a la propiedad versión de la entidad, respectivamente.

Si la entidad representa una enumeración, se debe definir mediante una clase Java que extienda, directa o indirectamente, la clase **AbstractPersistentEnumerationEntity**, la cual a su vez extiende **AbstractPersistentEntity** y, por lo tanto, también implementa la interfaz **Entity**.

Si utiliza la plataforma **jee1**, se puede definir extendiendo **PersistentEnumerationEntityBase**, que a su vez extiende **AbstractPersistentEnumerationEntity** e incluye las propiedades **numero** y **codigo**, correspondientes a la clave primaria y a la clave de negocio de la entidad, respectivamente.

Nota: en esta versión solo se toman en cuenta las entidades persistentes. Futuras versiones podrían tomar en cuenta entidades no persistentes.

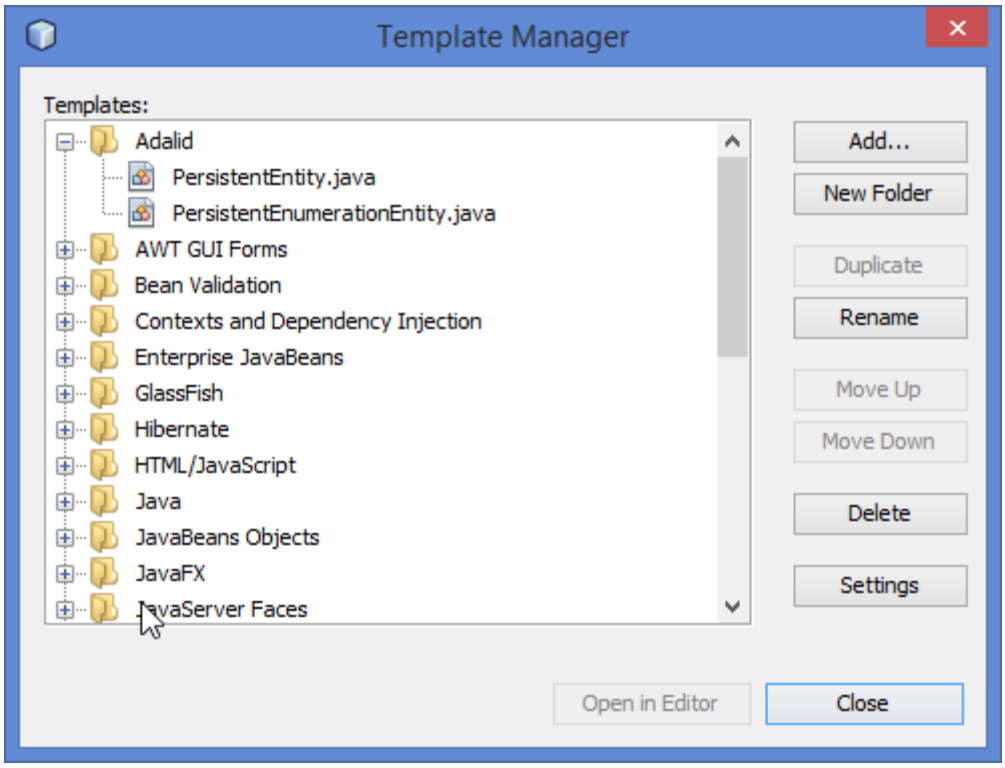
A las clases Java que definen las entidades de su aplicación se les conoce como **meta-entidades**.

Toda meta-entidad debe tener dos constructores, tal como se ilustra a continuación, utilizando como ejemplo una meta-entidad de nombre Persona:

```
public class Persona extends PersistentEntityBase {

    private Persona() {
        this(null, null);
    }

    public Persona(Artifact declaringArtifact, Field declaringField) {
        super(declaringArtifact, declaringField);
    }

}
```

Si utiliza la plataforma **jee1**, debe tener cuidado de no dar a las meta-entidades de su aplicación el nombre de alguna de las meta-entidades predefinidas de **jee1**. El apéndice 1 de este documento contiene la lista completa de meta-entidades predefinidas de **jee1**.

Si utiliza NetBeans entonces dispone de las siguientes plantillas de archivo: **PersistentEntity.java** y **PersistentEnumerationEntity.java**; además dispone de casi 100 plantillas de código para ayudarle a definir sus entidades persistentes. Las plantillas de archivo se deberían encontrar en la carpeta **Adalid** del "Template Manager" de NetBeans. Ambas incluyen una guía de referencia rápida de las plantillas de código en el *editor-fold* denominado *code templates*.



Para obtener información detallada sobre la instalación de estas plantillas, consulte la sección **Instalación de complementos para NetBeans** de la Guía de Instalación y Configuración, disponible para ambientes Linux y Windows.

Definición de atributos de entidades mediante anotaciones

Una parte de los atributos de una entidad se establecen decorando la meta-entidad con anotaciones, tal como se describe a continuación. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe en la sección Definición de atributos mediante métodos de este mismo documento.

Las anotaciones para establecer atributos de meta-entidades se dividen en los siguientes grupos:

- Anotaciones básicas:
  - EntityClass
  - EntityWarnings
- Anotaciones relacionadas con las vistas (páginas):
  - EntityTableView
  - EntityDetailView
  - EntityTreeView
  - EntityConsoleView
- Anotaciones relacionadas con las operaciones:
  - EntitySelectOperation
  - EntityExportOperation
  - EntityReportOperation
  - EntityInsertOperation
  - EntityUpdateOperation
  - EntityDeleteOperation

- Anotaciones relacionadas con la base de datos:
  - AbstractClass
  - DiscriminatorValue
  - EntityCodeGen
  - EntityTriggers
  - InheritanceMapping

Anotación AbstractClass

La anotación **AbstractClass** se utiliza para designar una entidad como abstracta. Una entidad abstracta debe generar una clase abstracta. A diferencia de las clases concretas, las clases abstractas no se pueden instanciar. Para obtener una instancia de esta clase hay que instanciar una clase concreta que la extienda. Esta anotación no tiene elementos**.**

Anotación DiscriminatorValue

La anotación **DiscriminatorValue** se utiliza para especificar el valor de la columna discriminadora (vea Anotación DiscriminatorColumn) que identifica la subclase a la que pertenecen las instancias de la entidad**.** Esta anotación tiene un único elemento, el cual se describe a continuación:

- **value**: valor de la columna discriminadora que identifica la subclase a la que pertenecen las instancias de la entidad. Si la columna discriminadora es una referencia a una entidad que corresponde a una enumeración, entonces este valor debe ser el valor de la clave primaria de la entidad referenciada.

Para información complementaria sobre la discriminación de las instancias en una jerarquía de clases, consulte la documentación de Java (la sección Entity Inheritance del capítulo Introduction to the Java Persistence API de The Java EE 7 Tutorial es un buen punto de partida).

Anotación EntityClass

La anotación **EntityClass** se utiliza para establecer atributos básicos de la entidad. Los elementos de la anotación son**:**

- **independent** {UNSPECIFIED, TRUE, FALSE}: indica si la entidad es, o no, existencialmente independiente. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la entidad es existencialmente independiente; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **resourceType** {UNSPECIFIED, CONFIGURATION, TESTING, OPERATION}: especifica el tipo de recurso de la entidad. Su valor es uno de los elementos de la enumeración **ResourceType**. Seleccione CONFIGURATION, TESTING u OPERATION si la entidad es de configuración, prueba u operación, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED si el tipo de recurso es indeterminado.
- **resourceGender** {UNSPECIFIED, MASCULINE, FEMININE, COMMON, NEUTER}: especifica el género gramatical de la entidad. Su valor es uno de los elementos de la enumeración **ResourceGender**. Seleccione MASCULINE, FEMININE, COMMON o NEUTER si la entidad es de género masculino, femenino, común o neutro, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado es NEUTER.
- **startWith** [**0:2.147.483.647**]: especifica el valor de la clave primaria de la primera instancia definida en la meta-entidad. Su valor debe ser un número entero entre 0 y 2.147.483.647. El valor predeterminado es 1. Para más información sobre la definición de instancias en la meta-entidad, vea la sección Definición de instancias de este mismo documento.

Anotación EntityCodeGen

La anotación **EntityCodeGen** se utiliza para especificar los generadores de código opcionales que se utilizarán para la entidad. En esta versión solo existe un generador de código opcional (los demás no son opcionales) y, por lo tanto, Esta anotación tiene un único elemento, el cual se describe a continuación:

- **sql** {UNSPECIFIED, TRUE, FALSE}: indica si se debe, o no, generar código SQL para la entidad. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para generar código SQL; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.

Anotación EntityConsoleView

La anotación **EntityConsoleView** se utiliza para controlar la generación de las vistas (páginas) de ejecución de operaciones de negocio de la entidad, conocidas como **Consolas de Procesamiento**. Los elementos de la anotación son:

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si se debe, o no, generar las vistas. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para generar las vistas; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE si la entidad tiene al menos una operación de negocio definida; en caso contrario es FALSE. Para más información sobre la definición de operaciones de negocio en la meta-entidad, vea la sección Definición de operaciones de negocio de este mismo documento.
- **width** [800:2400]: específica el número de pixeles de ancho de la vista. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor debe ser un número entero entre 800 y 2400. El valor predeterminado es 1200.

Anotación EntityDeleteOperation

La anotación **EntityDeleteOperation** se utiliza para establecer atributos de la operación **delete** de la entidad. Los elementos de la anotación son**:**

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas (páginas) de registro deben permitir, o no, eliminar instancias de la entidad, es decir, eliminar filas de la tabla de la base de datos correspondiente a la entidad. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir eliminar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **access** {UNSPECIFIED, PRIVATE, PUBLIC, PROTECTED, RESTRICTED}: especifica el tipo de control de acceso de la operación. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **OperationAccess**. Seleccione PRIVATE, PUBLIC, PROTECTED o RESTRICTED si la operación es de acceso privado, público, protegido o restringido, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RESTRICTED. Las operaciones con acceso privado no pueden ser ejecutadas directamente por los usuarios del sistema. Son ejecutadas solo por otras operaciones, a través de la Interfaz de Programación (API). Las operaciones con acceso público, protegido y restringido si pueden ser ejecutadas directamente por los usuarios del sistema, a través de la Interfaz de Usuario (UI). Las operaciones con acceso público pueden ser ejecutadas por todos los usuarios del sistema, aun cuando no tengan autorización explícita para ello. Las operaciones con acceso protegido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Al igual que las operaciones con acceso protegido, las operaciones con acceso restringido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios

explícitamente autorizados. Además, a diferencia de las operaciones con acceso protegido, las operaciones personalizables con acceso restringido, también pueden ser ejecutadas por usuarios que no tengan autorización explícita, pero solo sobre las instancias de la entidad que sean propiedad del usuario.

- **logging** {UNSPECIFIED, SUCCESS, FAILURE, BOTH}: especifica cuando se deben registrar pistas de auditoría de la ejecución de la operación. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **OperationLogging**. Seleccione SUCCESS, FAILURE o BOTH si las pistas se deben registrar cuando la operación se ejecute exitosamente, cuando se produzca un error al ejecutar la operación, o en ambos casos, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es SUCCESS.

Anotación EntityDetailView

La anotación **EntityDetailView** se utiliza para controlar la generación de las vistas (páginas) de consulta y/o registro detallado de la entidad, conocidas como **Detalles de Consulta** y/o **Detalles de Registro**, respectivamente. Los elementos de la anotación son:

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si se debe, o no, generar las vistas. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para generar las vistas; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **heading** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas Maestro/Detalle, donde la entidad es el detalle, muestran, o no, un encabezado con propiedades del maestro. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para mostrar el encabezado; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **width** [800:2400]: específica el número de pixeles de ancho de la vista. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor debe ser un número entero entre 800 y 2400. El valor predeterminado es 1200.

Anotación EntityExportOperation

La anotación **EntityExportOperation** se utiliza para establecer atributos de la operación **export** de la entidad. Los elementos de la anotación son**:**

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas (páginas) de consulta y registro deben permitir, o no, exportar el resultado de la consulta. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir exportar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **rowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben exportar. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.
- **sortOption** {ASC, DESC}: especifica el criterio de ordenamiento por omisión de las filas exportadas. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para exportar las filas ordenadas por el valor de su clave primaria, de manera ascendente o descendente, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor

predeterminado del atributo es ASC. Este criterio de ordenamiento se utiliza solo si no se establece otro criterio mediante el método setOrderBy.

Anotación EntityInsertOperation

La anotación **EntityInsertOperation** se utiliza para establecer atributos de la operación **insert** de la entidad. Los elementos de la anotación son**:**

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas (páginas) de registro deben permitir, o no, agregar nuevas instancias de la entidad, es decir, insertar nuevas filas de la tabla de la base de datos correspondiente a la entidad. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir agregar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **access** {UNSPECIFIED, PRIVATE, PUBLIC, PROTECTED, RESTRICTED}: especifica el tipo de control de acceso de la operación. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **OperationAccess**. Seleccione PRIVATE, PUBLIC, PROTECTED o RESTRICTED si la operación es de acceso privado, público, protegido o restringido, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RESTRICTED. Las operaciones con acceso privado no pueden ser ejecutadas directamente por los usuarios del sistema. Son ejecutadas solo por otras operaciones, a través de la Interfaz de Programación (API). Las operaciones con acceso público, protegido y restringido si pueden ser ejecutadas directamente por los usuarios del sistema, a través de la Interfaz de Usuario (UI). Las operaciones con acceso público pueden ser ejecutadas por todos los usuarios del sistema, aun cuando no tengan autorización explícita para ello. Las operaciones con acceso protegido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Al igual que las operaciones con acceso protegido, las operaciones con acceso restringido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Además, a diferencia de las operaciones con acceso protegido, las operaciones personalizables con acceso restringido, también pueden ser ejecutadas por usuarios que no tengan autorización explícita, pero solo sobre las instancias de la entidad que sean propiedad del usuario.
- **logging** {UNSPECIFIED, SUCCESS, FAILURE, BOTH}: especifica cuando se deben registrar pistas de auditoría de la ejecución de la operación. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **OperationLogging**. Seleccione SUCCESS, FAILURE o BOTH si las pistas se deben registrar cuando la operación se ejecute exitosamente, cuando se produzca un error al ejecutar la operación, o en ambos casos, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es SUCCESS.

Anotación EntityReportOperation

La anotación **EntityReportOperation** se utiliza para establecer atributos de la operación **report** de la entidad. Los elementos de la anotación son**:**

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas (páginas) de consulta y registro deben permitir, o no, reportar el resultado de la consulta. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir reportar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **rowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben reportar. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es

TRUE. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.

- **sortOption** {ASC, DESC}: especifica el criterio de ordenamiento por omisión de las filas reportadas. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para exportar las filas ordenadas por el valor de su clave primaria, de manera ascendente o descendente, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es ASC. Este criterio de ordenamiento se utiliza solo si no se establece otro criterio mediante el método setOrderBy.

Anotación EntitySelectOperation

La anotación **EntitySelectOperation** se utiliza para establecer atributos de la operación **select** de la entidad. Los elementos de la anotación son**:**

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si, además de las vistas (páginas) de registro, también se deben generar vistas de consulta de la entidad. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para generar las vistas de consulta; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **access** {UNSPECIFIED, PRIVATE, PUBLIC, PROTECTED, RESTRICTED}: especifica el tipo de control de acceso de la operación. Su valor es uno de los elementos de la enumeración **OperationAccess**. Seleccione PRIVATE, PUBLIC, PROTECTED o RESTRICTED si la operación es de acceso privado, público, protegido o restringido, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RESTRICTED. Las operaciones con acceso privado no pueden ser ejecutadas directamente por los usuarios del sistema. Son ejecutadas solo por otras operaciones, a través de la Interfaz de Programación (API). Las operaciones con acceso público, protegido y restringido si pueden ser ejecutadas directamente por los usuarios del sistema, a través de la Interfaz de Usuario (UI). Las operaciones con acceso público pueden ser ejecutadas por todos los usuarios del sistema, aun cuando no tengan autorización explícita para ello. Las operaciones con acceso protegido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Al igual que las operaciones con acceso protegido, las operaciones con acceso restringido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Además, a diferencia de las operaciones con acceso protegido, las operaciones personalizables con acceso restringido, también pueden ser ejecutadas por usuarios que no tengan autorización explícita, pero solo sobre las instancias de la entidad que sean propiedad del usuario.
- **rowsLimit** [0:10.000]: específica el número de máximo de filas que muestra la vista. Su valor debe ser un número entero entre 0 y 10.000. Utilice 0 cuando no exista límite. El valor predeterminado es 500.
- **sortOption** {ASC, DESC}: especifica el criterio de ordenamiento por omisión de las filas que muestra la vista. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para mostrar las filas ordenadas por el valor de su clave primaria, de manera ascendente o descendente, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es ASC. Este criterio de ordenamiento se utiliza solo si no se establece otro criterio mediante el método setOrderBy.

Anotación EntityTableView

La anotación **EntityTableView** se utiliza para controlar la generación de las vistas (páginas) de consulta y/o registro tabular de la entidad, conocidas como **Tablas de Consulta** y/o **Tablas de Registro**, respectivamente. Los elementos de la anotación son**:**

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si se debe, o no, generar las vistas. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para generar las vistas; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **inserts** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas deben permitir, o no, agregar nuevas instancias de la entidad, es decir, insertar nuevas filas a la tabla de la base de datos correspondiente a la entidad. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir agregar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **updates** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas deben permitir, o no, modificar instancias de la entidad, es decir, actualizar filas de la tabla de la base de datos correspondiente a la entidad. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir modificar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **deletes** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas deben permitir, o no, eliminar instancias de la entidad, es decir, eliminar filas de la tabla de la base de datos correspondiente a la entidad. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir eliminar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **heading** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas Maestro/Detalle, donde la entidad es el detalle, muestran, o no, un encabezado con propiedades del maestro. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para mostrar el encabezado; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **rows** [1:50]: específica el número de filas por página que muestra la vista. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor debe ser un número entero entre 1 y 50. El valor predeterminado es 10.
- **width** [800:2400]: específica el número de pixeles de ancho de la vista. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor debe ser un número entero entre 800 y 2400. El valor predeterminado es 1200.

Anotación EntityTreeView

La anotación **EntityTreeView** se utiliza para controlar la generación de las vistas (páginas) de consulta y/o registro jerárquico de la entidad, conocidas como **Árboles de Consulta** y/o **Árboles de Registro**, respectivamente. Solo aplica si la entidad es jerárquica, es decir, tiene una relación de varios-a-uno con ella misma y la propiedad que establece tal relación ha sido designada como propiedad padre (vea Anotación ParentProperty**)**. Esta anotación tiene un único elemento, el cual se describe a continuación:

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si se debe, o no, generar las vistas. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para generar las vistas; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.

Anotación EntityTriggers

La anotación **EntityTriggers** se utiliza para configurar la generación de los disparadores (triggers) de la tabla de la base de datos correspondiente a la entidad. Los elementos de la anotación son**:**

- **beforeValue** {UNSPECIFIED, TRUE, FALSE}: indica si los *triggers* **before insert row** y **before update row** deben, o no, ejecutar la función **before_value** antes de asignar el valor por omisión a las columnas de la tabla. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para ejecutar la función; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **afterValue** {UNSPECIFIED, TRUE, FALSE}: indica si los *triggers* **before insert row** y **before update row** deben, o no, ejecutar la función **after_value** después de asignar el valor por omisión a las columnas de la tabla. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para ejecutar la función; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **beforeCheck** {UNSPECIFIED, TRUE, FALSE}: indica si los *triggers* **before insert row** y **before update row** deben, o no, ejecutar la función **before_check** antes de comprobar las restricciones (*check constraints*) de la tabla. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para ejecutar la función; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **afterCheck** {UNSPECIFIED, TRUE, FALSE}: indica si los *triggers* **before insert row** y **before update row** deben, o no, ejecutar la función **after_check** después de comprobar las restricciones (*check constraints*) de la tabla. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para ejecutar la función; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.

Anotación EntityUpdateOperation

La anotación **EntityUpdateOperation** se utiliza para establecer atributos de la operación **update** de la entidad. Los elementos de la anotación son**:**

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas (páginas) de registro deben permitir, o no, modificar instancias de la entidad, es decir, actualizar filas de la tabla de la base de datos correspondiente a la entidad. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir modificar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **access** {UNSPECIFIED, PRIVATE, PUBLIC, PROTECTED, RESTRICTED}: especifica el tipo de control de acceso de la operación. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **OperationAccess**. Seleccione PRIVATE, PUBLIC, PROTECTED o RESTRICTED si la operación es de acceso privado, público, protegido o restringido, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RESTRICTED. Las operaciones con acceso privado no pueden ser ejecutadas directamente por los usuarios del sistema. Son ejecutadas solo por otras operaciones, a través de la Interfaz de Programación (API). Las operaciones con acceso público, protegido y restringido si pueden ser ejecutadas directamente por los usuarios del sistema, a través de la Interfaz de Usuario (UI). Las operaciones con acceso público pueden ser ejecutadas por todos los usuarios del sistema, aun cuando no tengan autorización explícita para ello. Las operaciones con acceso protegido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Al igual que las operaciones con acceso protegido, las operaciones con acceso

restringido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Además, a diferencia de las operaciones con acceso protegido, las operaciones personalizables con acceso restringido, también pueden ser ejecutadas por usuarios que no tengan autorización explícita, pero solo sobre las instancias de la entidad que sean propiedad del usuario.

- **logging** {UNSPECIFIED, SUCCESS, FAILURE, BOTH}: especifica cuando se deben registrar pistas de auditoría de la ejecución de la operación. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **enabled** es TRUE. Su valor es uno de los elementos de la enumeración **OperationLogging**. Seleccione SUCCESS, FAILURE o BOTH si las pistas se deben registrar cuando la operación se ejecute exitosamente, cuando se produzca un error al ejecutar la operación, o en ambos casos, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es SUCCESS.

Anotación EntityWarnings

La anotación **EntityWarnings** se utiliza para controlar la emisión de mensajes de advertencia al construir y generar la aplicación. Esta anotación tiene un único elemento, el cual se describe a continuación:

- **enabled** {UNSPECIFIED, TRUE, FALSE}: indica si se debe, o no, emitir mensajes de advertencia. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para emitir los mensajes; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.

Anotación InheritanceMapping

La anotación **InheritanceMapping** se utiliza para especificar la estrategia que se debe utilizar para generar las tablas de la base de datos que corresponden a la entidad y a las entidades que la extienden; por lo tanto, esta anotación es relevante solo si la entidad es extendida por otras entidades**.** Esta anotación tiene un único elemento, el cual se describe a continuación:

- **strategy** {UNSPECIFIED, SINGLE_TABLE, JOINED, TABLE_PER_CLASS}: especifica la estrategia que se debe utilizar para generar las tablas de la base de datos. Su valor es uno de los elementos de la enumeración **InheritanceMappingStrategy**. Seleccione SINGLE_TABLE para generar una sola tabla para almacenar las filas que corresponden a instancias de todas las entidades de la jerarquía. Seleccione JOINED para generar una tabla para cada entidad, abstracta o concreta, de la jerarquía; salvo que correspondan a propiedades básicas de la entidad (vea Anotación BaseField), las tablas generadas no tienen columnas para propiedades heredadas pero tienen las relaciones uno-a-uno necesarias para hacer el enlace (*join*) entre ellas. Seleccione TABLE_PER_CLASS para generar una tabla para cada entidad concreta de la jerarquía; las tablas generadas tienen columnas para todas las propiedades de la entidad, incluyendo las heredadas, y no tienen relación unas con otras. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es SINGLE_TABLE.

Para información complementaria sobre las estrategias para generar las tablas de la base de datos, consulte la documentación de Java (la sección Entity Inheritance del capítulo Introduction to the Java Persistence API de The Java EE 7 Tutorial es un buen punto de partida).

Anotaciones de AbstractPersistentEntity

La clase **AbstractPersistentEntity** está decorada con las siguientes anotaciones:

```
@EntityClass(resourceType = ResourceType.OPERATION)
@EntityTreeView(enabled = Kleenean.FALSE)
@EntityReferenceSearch(searchType = SearchType.DISPLAY,
    displayMode = DisplayMode.READING)
```

En una clase que extienda **AbstractPersistentEntity**, el efecto de una cualquiera de estas anotaciones se anula al decorar la subclase con esa misma anotación.

Anotaciones de AbstractPersistentEnumerationEntity

La clase **AbstractPersistentEnumerationEntity** está decorada con las siguientes anotaciones:

```
@EntityClass(resourceType = ResourceType.CONFIGURATION)
@EntitySelectOperation(rowsLimit = 0)
@EntityInsertOperation(enabled = Kleenean.FALSE)
@EntityUpdateOperation(enabled = Kleenean.TRUE)
@EntityDeleteOperation(enabled = Kleenean.FALSE)
@EntityReportOperation(enabled = Kleenean.FALSE)
@EntityExportOperation(enabled = Kleenean.FALSE)
@EntityDetailView(enabled = Kleenean.FALSE)
@EntityTreeView(enabled = Kleenean.FALSE)
@EntityConsoleView(enabled = Kleenean.FALSE)
@EntityReferenceSearch(searchType = SearchType.LIST)
```

En una clase que extienda **AbstractPersistentEnumerationEntity**, el efecto de una cualquiera de estas anotaciones se anula al decorar la subclase con esa misma anotación.

Definición de atributos de entidades mediante métodos

Una parte de los atributos de una entidad se establecen decorando la meta-entidad con anotaciones, tal como se describe en la sección Definición de atributos mediante anotaciones de este mismo documento. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Los métodos para establecer atributos de meta-entidades se dividen en los siguientes grupos:

- Métodos relacionados con la interfaz de la aplicación:
  - setDefaultLabel
  - setDefaultShortLabel
  - setDefaultCollectionLabel
  - setDefaultCollectionShortLabel
  - setDefaultDescription
  - setDefaultShortDescription
  - setDefaultTooltip
  - setDefaultTab
- Métodos relacionados con las operaciones:
  - setOrderBy
  - setSelectFilter
  - setUpdateFilter
  - setDeleteFilter
  - setInsertFilter
  - setMasterDetailFilter
- Métodos relacionados con la base de datos:
  - setSchema

- setSqlName

Método setDefaultCollectionLabel

El método **setDefaultCollectionLabel** de la meta-entidad se utiliza para establecer la etiqueta de colección de la entidad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultCollectionLabel** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene varias firmas, las cuales se describen a continuación.

setDefaultCollectionLabel(String defaultLabel)

Establece la etiqueta de colección de la entidad. Su único parámetro es:

- **defaultLabel**: sustantivo plural que se usa como etiqueta de colección de la entidad.

setDefaultCollectionLabel(EntityReference reference, String defaultLabel)

Establece la etiqueta de colección de la entidad para las vistas (páginas) Maestro/Detalle con el maestro identificado por el parámetro **reference**. Sus parámetros son:

- **reference**: entidad que hace la función de maestro en la vista Maestro/Detalle.
- **defaultLabel**: sustantivo plural que se usa como etiqueta de colección de la entidad.

Método setDefaultCollectionShortLabel

El método **setDefaultCollectionShortLabel** de la meta-entidad se utiliza para establecer la etiqueta corta de colección de la entidad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultCollectionShortLabel** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene varias firmas, las cuales se describen a continuación.

setDefaultCollectionShortLabel(String defaultLabel)

Establece la etiqueta corta de colección de la entidad. Su único parámetro es:

- **defaultLabel**: sustantivo plural, preferiblemente simple, que se usa como etiqueta corta de colección de la entidad.

setDefaultCollectionShortLabel(EntityReference reference, String defaultLabel)

Establece la etiqueta corta de colección de la entidad para las vistas (páginas) Maestro/Detalle con el maestro identificado por el parámetro **reference**. Sus parámetros son:

- **reference**: entidad que hace la función de maestro en la vista Maestro/Detalle.
- **defaultLabel**: sustantivo plural, preferiblemente simple, que se usa como etiqueta corta de colección de la entidad.

Método setDefaultDescription

El método **setDefaultDescription** de la meta-entidad se utiliza para establecer la descripción de la entidad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la entidad. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la entidad.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-entidad se utiliza para establecer la descripción corta de la entidad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la entidad. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la entidad.

Método setDefaultLabel

El método **setDefaultLabel** de la meta-entidad se utiliza para establecer la etiqueta de la entidad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultLabel** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene varias firmas, las cuales se describen a continuación.

setDefaultLabel(String defaultLabel)

Establece la etiqueta de la entidad. Su único parámetro es:

- **defaultLabel**: sustantivo singular que se usa como etiqueta de la entidad.

setDefaultLabel(EntityReference reference, String defaultLabel)

Establece la etiqueta de la entidad para las vistas (páginas) Maestro/Detalle con el maestro identificado por el parámetro **reference**. Sus parámetros son:

- **reference**: entidad que hace la función de maestro en la vista Maestro/Detalle.
- **defaultLabel**: sustantivo singular que se usa como etiqueta de la entidad.

Método setDefaultShortLabel

El método **setDefaultShortLabel** de la meta-entidad se utiliza para establecer la etiqueta corta de la entidad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultShortLabel** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene varias firmas, las cuales se describen a continuación.

setDefaultShortLabel(String defaultLabel)

Establece la etiqueta corta de la entidad. Su único parámetro es:

- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta de la entidad.

setDefaultShortLabel(EntityReference reference, String defaultLabel)

Establece la etiqueta corta de la entidad para las vistas (páginas) Maestro/Detalle con el maestro identificado por el parámetro **reference**. Sus parámetros son:

- **reference**: entidad que hace la función de maestro en la vista Maestro/Detalle.
- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta de la entidad.

Método setDefaultTab

El método **setDefaultTab** de la meta-entidad se utiliza para establecer la pestaña (*tab*) por defecto, es decir, la pestaña que recibe el enfoque al abrir una vista (página) con pestañas de la entidad.

El método **setDefaultTab** debe ejecutarse en el método **settleTabs** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultTab(Tab defaultTab)

Establece la pestaña por defecto de la entidad. Su único parámetro es:

- **defaultTab**: pestaña que recibe el enfoque al abrir una vista con pestañas de la entidad.

Método setDefaultTooltip

El método **setDefaultTooltip** de la meta-entidad se utiliza para establecer la descripción emergente (*tooltip*) de la entidad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultTooltip** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultTooltip(String defaultTooltip)

Establece la descripción emergente de la entidad. Su único parámetro es:

- **defaultTooltip**: una o más oraciones que describen muy brevemente la entidad.

Método setDeleteFilter

El método **setDeleteFilter** de la meta-entidad se utiliza para establecer el filtro de selección de la operación **delete** de las vistas (páginas) de registro de la entidad. Solo las instancias de la entidad que cumplen con los criterios del filtro podrán ser eliminadas con la operación **delete**.

El método **setDeleteFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDeleteFilter(BooleanExpression filter)

Establece el filtro de selección de la operación **delete**. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setInsertFilter

El método **setInsertFilter** de la meta-entidad se utiliza para establecer el filtro de selección de la operación **insert** de las vistas (páginas) de registro, Maestro/Detalle, de la entidad. Solo las instancias de la entidad referenciada que cumplen con los criterios del filtro podrán ser utilizadas como maestro por la operación **insert**.

El método **setInsertFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setInsertFilter(BooleanExpression filter)

Establece el filtro de selección de la operación **insert**. Su único parámetro es:

- **filter**: expresión booleana, definida en una entidad referenciada (por una relación con cardinalidad *varios-a-uno*), que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setMasterDetailFilter

El método **setMasterDetailFilter** de la meta-entidad se utiliza para establecer el filtro de selección de la operación **select** de las vistas (páginas) de consulta y registro, Maestro/Detalle, de la entidad. Solo las instancias de la entidad referenciada que cumplen con los criterios del filtro podrán ser utilizadas como maestro por la operación **select**.

El método **select** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setMasterDetailFilter(BooleanExpression filter)

Establece el filtro de selección de la operación **insert**. Su único parámetro es:

- **filter**: expresión booleana, definida en una entidad referenciada (por una relación con cardinalidad *varios-a-uno*), que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setOrderBy

El método **setOrderBy** de la meta-entidad se utiliza para establecer el criterio de ordenamiento de las operaciones **select**, **export** y **report** de las vistas (páginas) de consulta y/o registro de la entidad.

El método **setOrderBy** tiene varias firmas, las cuales se describen a continuación.

setOrderBy(Key orderBy)

Establece el criterio de ordenamiento de las operaciones **select**, **export** y **report**. Debe ejecutarse en el método **settleKeys** de la meta-entidad. Su único parámetro es:

- **orderBy**: clave de acceso cuyas propiedades se utilizan como criterio de ordenamiento, en la misma secuencia en la que aparecen en la clave. Para más información sobre la definición de claves de acceso en la meta-entidad, vea la sección Definición de claves de acceso de este mismo documento.

setOrderBy(Property... orderBy)

Establece el criterio de ordenamiento de las operaciones **select**, **export** y **report**. Debe ejecutarse en el método **settleProperties** de la meta-entidad. Su único parámetro es:

- **orderBy**: una o más propiedades (separadas por una coma) que se utilizan como criterio de ordenamiento, en la misma secuencia en la que son escritas.

Método setSchema

El método **setSchema** de la meta-entidad se utiliza para establecer el nombre del esquema de la base de datos en el que se define la tabla correspondiente a la entidad.

El método **setSchema** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setSchema(String schema)

Establece el nombre del esquema de la tabla correspondiente a la entidad. Su único parámetro es:

- **schema**: nombre del esquema.

Método setSelectFilter

El método **setSelectFilter** de la meta-entidad se utiliza para establecer el filtro de selección de las operaciones **select**, **export** y **report** de las vistas (páginas) de consulta y/o registro de la entidad. Solo las instancias de la entidad que cumplen con los criterios del filtro son incluidas en el resultado de estas operaciones.

El método **setSelectFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setSelectFilter(BooleanExpression filter)

Establece el filtro de selección de las operaciones **select**, **export** y **report**. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setSqlName

El método **setSqlName** de la meta-entidad se utiliza para establecer el nombre de la tabla de la base de datos correspondiente a la entidad. Si este método no es ejecutado, el nombre de la tabla se determina a partir del nombre de la meta-entidad, sustituyendo cada letra mayúscula por un guion bajo (*underscore*) seguido de la letra convertida en minúscula.

El método **setSqlName** debe ejecutarse en el método **settleAttributes** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setSqlName(String sqlName)

Establece el nombre de la tabla correspondiente a la entidad. Su único parámetro es:

- **sqlName**: nombre de la tabla.

Método setUpdateFilter

El método **setUpdateFilter** de la meta-entidad se utiliza para establecer el filtro de selección de la operación **update** de las vistas (páginas) de registro de la entidad. Solo las instancias de la entidad que cumplen con los criterios del filtro podrán ser modificadas con la operación **update**.

El método **setUpdateFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setUpdateFilter(BooleanExpression filter)

Establece el filtro de selección de la operación **update**. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Definición de propiedades

Cada una de las propiedades de la entidad se debe definir como un campo de la meta-entidad. A los campos de la meta-entidad que definen propiedades se les conoce como **meta-propiedades**.

Las meta-propiedades se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **Property**.

En función del tipo de dato de la propiedad que definen, las meta-propiedades se dividen en dos grupos:

- **Primitivas**: si la propiedad es de un tipo primitivo o elemental, es decir, Boolean, Character, String, BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short, Date, Time o Timestamp.
- **Referencias**: si la propiedad es una referencia a una entidad.

Para definir primitivas se utilizan las siguientes clases:

- **BigDecimalProperty**: para definir números decimales, con signo y precisión variable.
- **BigIntegerProperty**: para definir números enteros, con signo y precisión variable.
- **BinaryProperty**: para definir “objetos binarios” (imágenes, vídeos, etc.).
- **BooleanProperty**: para definir booleanos (TRUE o FALSE).
- **ByteProperty**: para definir números enteros, con signo, entre -128 y 127.
- **CharacterProperty**: para definir un solo carácter (letra, dígito o carácter especial).
- **DateProperty**: para definir fechas.
- **DoubleProperty**: para definir una representación de coma flotante de números reales (*floating point*) en formato (precisión) doble.
- **FloatProperty**: para definir una representación de coma flotante de números reales (*floating point*) en formato (precisión) simple.
- **IntegerProperty**: para definir números enteros, con signo, entre -2.147.483.648 y 2.147.483.647.
- **LongProperty**: para definir números enteros, con signo, entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.
- **ShortProperty**: para definir números enteros, con signo, entre -32.768 y 32.767.
- **StringProperty**: para definir cadenas de caracteres de longitud variable.
- **TimeProperty**: para definir horas.
- **TimestampProperty**: para definir “sellos de tiempo” (*timestamps*),

Para definir referencias se utiliza cualquier meta-entidad. Tanto la clase **AbstractPersistentEntity** como la clase **AbstractPersistentEnumerationEntity**, utilizadas para definir meta-entidades, implementan la interfaz **Property**.

Toda meta-propiedad, primitiva o referencia, se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-propiedades.

```
public class Persona extends PersistentEntityBase {

    constructores

    @BusinessKey
    @StringField(regex = "[veVE][-][0-9]{8}", letterCase = LetterCase.UPPER)
    public StringProperty cedula;

    @NameProperty
    @StringField(maxLength = 30)
    public StringProperty nombre;

    @ColumnField(nullable = Kleenean.FALSE)
    public DateProperty fechaNacimiento;

    @PropertyField(create = Kleenean.FALSE, update = Kleenean.FALSE)
    public DateProperty fechaDefuncion;

    @ColumnField(nullable = Kleenean.FALSE)
    @ManyToOne(view = MasterDetailView.NONE)
    @PropertyField(create = Kleenean.TRUE)
    public Sexo sexo;

    @ManyToOne(view = MasterDetailView.NONE)
    @PropertyField(create = Kleenean.FALSE, update = Kleenean.FALSE)
    public Persona conyuge;

    public BigDecimalProperty limiteCredito;

    @StringField(regex = "[0-9]{3}[-][0-9]{7}")
    @PropertyField(create = Kleenean.TRUE)
    public StringProperty telefono;

    @PropertyField(create = Kleenean.TRUE)
    public StringProperty direccion;

    @ManyToOne(view = MasterDetailView.NONE)
    public Estado estado;

    @ManyToOne(view = MasterDetailView.NONE)
    public Pais pais;

    @Override
    protected void settleProperties() {
        super.settleProperties();
        cedula.setDefaultDescription("número de cédula de identidad; "
            + "debe comenzar por la letra V ó E "
            + "seguida de un guión y 8 dígitos");
        telefono.setDefaultDescription("número de teléfono; "
            + "debe contener 2 grupos de dígitos, separados por guión; "
            + "el primer grupo es de 3 digitos y el segundo de 7; "
            + "el primer grupo corresponde al código de área");
        limiteCredito.setDefaultValue(10000);
    }

}
```

Note que las propiedades sexo, cónyuge, estado y país no son primitivas sino referencias a otras entidades, de nombre Sexo, Persona, Estado y País, respectivamente.

Definición de atributos de propiedades mediante anotaciones

Una parte de los atributos de una propiedad se establecen decorando la meta-propiedad con anotaciones, tal como se describe a continuación. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe en la sección Definición de atributos de propiedades mediante métodos de este mismo documento.

Las anotaciones para establecer atributos de meta-propiedades se dividen en los siguientes grupos:

- Anotaciones básicas:
  - PropertyField
  - BigDecimalField
  - BooleanField
  - NumericField
  - StringField
  - TimeField
  - TimestampField

- Anotaciones para establecer el rol de la propiedad en la entidad:
  - BusinessKey
  - NameProperty
  - DescriptionProperty
  - UrlProperty
  - VersionProperty
  - ParentProperty
  - OwnerProperty
  - SegmentProperty
  - InactiveIndicator

- Anotaciones para referencias:
  - Allocation
  - ManyToOne
  - OneToOne
  - EntityReferenceSearch
  - Filter

- Anotaciones relacionadas con la base de datos:
  - PrimaryKey
  - CharacterKey
  - NumericKey
  - UniqueKey
  - ForeignKey
  - BaseField
  - ColumnField
  - DiscriminatorColumn

- Otras anotaciones:
  - FileReference

Anotación Allocation

La anotación **Allocation** se utiliza para establecer límites a la instanciación de las referencias de la entidad. Estos límites se deben ajustar en función de las expresiones programadas en la meta-entidad,

ya que éstas pueden utilizar propiedades de entidades referenciadas. Si una expresión utiliza una propiedad de una entidad referenciada que está fuera de alcance (*out-of-scope*) entonces, al generar la aplicación, se produce un **NullPointerException** en la instrucción correspondiente a la expresión. La solución a este problema es aumentar los límites establecidos (explícitamente o por omisión) para la instanciación de la referencia que está fuera de alcance.

El primero de los límites determina la profundidad máxima que se puede alcanzar al instanciar la referencia. Las propiedades de la entidad tienen profundidad 1. Las propiedades de las entidades referenciadas en la entidad tienen profundidad 2. Las propiedades de las entidades referenciadas por las entidades referenciadas en la entidad tienen profundidad 3, y así sucesivamente, ad-infinitum.

El segundo de los límites determina la cantidad máxima de referencias circulares que se puede alcanzar al instanciar una referencia. Una referencia circular es una referencia a la misma entidad, hecha de forma directa o indirecta (en otras palabras, hecha a cualquier profundidad).

Los elementos de la anotación son**:**

- **maxDepth**: específica la profundidad máxima de la referencia. Su valor debe ser un número entero mayor o igual a 1. El valor predeterminado es 1.
- **maxRound**: específica la cantidad máxima de referencias circulares de la referencia. Su valor debe ser un número entero mayor o igual a 0. El valor predeterminado es 1 para las propiedades padre y 0 para las demás referencias.

Anotación BaseField

La anotación **BaseField** se utiliza para designar las propiedades básicas de la entidad. Esta anotación solo es relevante cuando la entidad es extendida y la estrategia para generar las tablas de la base de datos de la jerarquía es JOINED (vea Anotación InheritanceMapping). Cuando la estrategia es JOINED, las tablas generadas no tienen columnas para propiedades heredadas a menos que la propiedad heredada sea una propiedad básica. Las claves primarias y las propiedades versión son implícitamente básicas y, por lo tanto, no es necesario decorarlas con la anotación **BaseField**. Esta anotación no tiene elementos**.**

Anotación BigDecimalField

La anotación **BigDecimalField** se utiliza para establecer atributos de propiedades **BigDecimalProperty**. Los elementos de la anotación son**:**

- **precision** [1:1.000]: específica la precisión o cantidad de dígitos significativos de la propiedad. Su valor debe ser un número entero entre 1 y 1.000. El valor predeterminado es 16.
- **scale**: [0:1.000]: específica la escala o cantidad de decimales de la propiedad. Su valor debe ser un número entero entre 0 y 1.000 y debe ser menor o igual a **precision**. El valor predeterminado es 0.

Anotación BooleanField

La anotación **BooleanField** se utiliza para establecer atributos de propiedades **BooleanProperty**. Esta anotación tiene un único elemento, el cual se describe a continuación**:**

- **displayType** {UNSPECIFIED, DROPDOWN, CHECKBOX}: indica el tipo de componente que se utiliza para mostrar el valor de la propiedad en las vistas (páginas) de consulta y registro. Su valor es uno de los elementos de la enumeración **BooleanDisplayType**. Seleccione DROPDOWN o CHECKBOX para utilizar una lista desplegable o una casilla de verificación, respectivamente. La opción CHECKBOX solo aplica si la columna de la base de datos que corresponde a la propiedad no permite valores nulos (vea el elemento **nullable** de la Anotación ColumnField). Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor

predeterminado del atributo. El valor predeterminado del atributo es DROPDOWN si la columna de la base de datos que corresponde a la propiedad permite valores nulos; y CHECKBOX si no los permite.

Anotación BusinessKey

La anotación **BusinessKey** se utiliza para designar una propiedad como clave de negocio de la entidad. Cada entidad puede tener una sola clave de negocio. Las claves de negocio, también llamadas claves naturales, son claves candidatas (por lo tanto no admiten valores duplicados) que representan atributos del mundo real con los que los usuarios de la aplicación están familiarizados y, por lo tanto, permiten identificar fácilmente las entidades. Solo se puede designar como clave de negocio a propiedades de la clase **StringProperty**. Esta anotación no tiene elementos.

Anotación CharacterKey

La anotación **CharacterKey** se utiliza para designar una propiedad como clave alfanumérica de la entidad. Cada entidad puede tener una sola clave de alfanumérica. Las claves alfanuméricas son claves candidatas (por lo tanto no admiten valores duplicados) que permiten valores alfanuméricos. Solo se puede designar como clave alfanumérica a propiedades de la clase **StringProperty**. Esta anotación no tiene elementos.

Anotación ColumnField

La anotación **ColumnField** se utiliza para establecer atributos de la columna de la base de datos que corresponde a la propiedad. Los elementos de la anotación son**:**

- **nullable** {UNSPECIFIED, TRUE, FALSE}: indica si la columna admite, o no, valores nulos. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la columna admite valores nulos; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **insertable** {UNSPECIFIED, TRUE, FALSE}: indica si la columna es, o no, insertable; es decir, si los componentes para el manejo de la persistencia incluyen, o no, la columna en las operaciones **insert**. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la columna es insertable; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **updateable** {UNSPECIFIED, TRUE, FALSE}: indica si la columna es, o no, actualizable; es decir, si los componentes para el manejo de la persistencia incluyen, o no, la columna en las operaciones **update**. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la columna es actualizable; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **unique** {UNSPECIFIED, TRUE, FALSE}: indica si la columna admite, o no, valores duplicados. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione FALSE si la columna admite valores duplicados; en caso contrario, seleccione TRUE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE. Las columnas que corresponden a claves primarias (vea Anotación PrimaryKey), claves de negocio (vea Anotación BusinessKey), claves numéricas (vea Anotación NumericKey), claves alfanuméricas (vea Anotación CharacterKey) y claves únicas (vea Anotación UniqueKey), no admiten valores duplicados, independientemente del valor de este elemento.

Anotación DescriptionProperty

La anotación **DescriptionProperty** se utiliza para designar una propiedad como propiedad descripción de la entidad. Cada entidad puede tener una sola propiedad descripción. Solo se puede designar como propiedad descripción a propiedades de la clase **StringProperty**. Esta anotación no tiene elementos.

Anotación DiscriminatorColumn

La anotación **DiscriminatorColumn** se utiliza para designar una propiedad como la columna discriminadora cuyo valor identifica la subclase a la que pertenecen las instancias de la entidad (vea Anotación DiscriminatorValue)**.** Esta anotación no tiene elementos.

Para información complementaria sobre la discriminación de las instancias en una jerarquía de clases, consulte la documentación de Java (la sección Entity Inheritance del capítulo Introduction to the Java Persistence API de The Java EE 7 Tutorial es un buen punto de partida).

Anotación EntityReferenceSearch

La anotación **EntityReferenceSearch** se utiliza para configurar la forma en que las vistas (páginas) implementan la búsqueda del valor de la referencia (propiedad que hace referencia a otra entidad). Los elementos de la anotación son**:**

- **searchType** {UNSPECIFIED, LIST, DISPLAY, NONE}: especifica el tipo de búsqueda. Su valor es uno de los elementos de la enumeración **SearchType**. Seleccione LIST, DISPLAY o NONE para buscar mediante una lista desplegable (*drop-down list*), una vista (página), o para no implementar ningún mecanismo de búsqueda, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es LIST, si la entidad corresponde a una enumeración; y DISPLAY, en los demás casos.
- **listStyle** {UNSPECIFIED, CHARACTER_KEY, NAME, CHARACTER_KEY_AND_NAME, PRIMARY_KEY_AND_CHARACTER_KEY, PRIMARY_KEY_AND_NAME}: especifica el tipo de lista desplegable que se utiliza para la búsqueda. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **searchType** es LIST. Su valor es uno de los elementos de la enumeración **ListStyle**. Seleccione CHARACTER_KEY, NAME, CHARACTER_KEY_AND_NAME, PRIMARY_KEY_AND_CHARACTER_KEY o PRIMARY_KEY_AND_NAME para que la lista desplegable muestre la clave alfanumérica (o de negocio), el nombre, la clave alfanumérica y el nombre, la clave primaria y la clave alfanumérica, o la clave primaria y el nombre, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es CHARACTER_KEY.
- **displayMode** {UNSPECIFIED, READING, WRITING}: especifica el tipo de página que se utiliza para la búsqueda. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **searchType** es DISPLAY. Su valor es uno de los elementos de la enumeración **DisplayMode**. Seleccione READING o WRITING para utilizar una vista (página) de solo consulta o una vista de registro, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es READING.

Anotación FileReference

La anotación **FileReference** se utiliza para designar propiedades **StringProperty** como referencias a archivos cargados en el servidor. Los elementos de la anotación son**:**

- **max**: específica el tamaño máximo (en bytes). Su valor debe ser un número entero, mayor o igual a 0. Utilice 0 para permitir la carga de archivos de cualquier tamaño. Alternativamente, omita el elemento para utilizar el valor predeterminado del atributo. El valor predeterminado es 1.000.000 (1 MB).

- **types** {APPLICATION, AUDIO, DRAWING, IMAGE, MUSIC, TEXT, VIDEO}: lista de extensiones MIME (*Multipurpose Internet Mail Extensions*) válidas. Su valor es una lista de elementos de la enumeración **MimeType**. Alternativamente, omita el elemento permitir la carga de archivos de cualquier tipo.
- **storage** {UNSPECIFIED, FILE, ROW, ROW_AND_FILE}: especifica el tipo de almacenamiento de los archivos. Su valor es uno de los elementos de la enumeración **UploadStorageOption**. Seleccione FILE, ROW o ROW_AND_FILE para almacenar el archivo en el servidor de aplicaciones (web), en la base de datos, o en ambos, respectivamente. Alternativamente, omita el elemento para utilizar el valor predeterminado del atributo. El valor predeterminado es ROW_AND_FILE.
- **joinField**: nombre de la propiedad (de la misma entidad) que hace referencia a la tabla de la base de datos donde se almacena el archivo. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **storage** es ROW o ROW_AND_FILE.

Anotación Filter

La anotación **Filter** se utiliza para especificar los filtros automáticos que las vistas (páginas) implementan en la búsqueda del valor de la referencia (propiedad que hace referencia a otra entidad). Los elementos de la anotación son**:**

- **inactive** {UNSPECIFIED, TRUE, FALSE}: indica si se deben filtrar (ignorar), o no, las instancias de la entidad referenciada que se encuentran inactivas (eliminadas lógicamente). Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para filtrar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **owner** {UNSPECIFIED, TRUE, FALSE}: indica si se deben filtrar (ignorar), o no, las instancias de la entidad referenciada que son propiedad de un usuario diferente al que realiza la búsqueda (vea Anotación OwnerProperty). Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para filtrar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **segment** {UNSPECIFIED, TRUE, FALSE}: indica si se deben filtrar (ignorar), o no, las instancias de la entidad referenciada que no pertenecen a uno de los segmentos autorizados al usuario que realiza la búsqueda (vea Anotación SegmentProperty). Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para filtrar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.

Anotación ForeignKey

La anotación **ForeignKey** se utiliza para generar una clave foránea en la base de datos para la referencia (propiedad que hace referencia a otra entidad). Los elementos de la anotación son**:**

- **onDelete** {UNSPECIFIED, NONE, CASCADE, NULLIFY}: especifica la acción del gestor de base de datos al eliminar filas de la tabla de la base de datos correspondiente a la entidad referenciada. Su valor es uno de los elementos de la enumeración **OnDeleteAction**. Seleccione NONE, CASCADE o NULLIFY para impedir la eliminación cuando existan referencias a la instancia eliminada; eliminar las filas que contienen referencias a la instancia eliminada; o actualizar las filas que contienen referencias a la instancia eliminada, colocando valor nulo en la columna correspondiente a la referencia, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es NONE.
- **onUpdate** {UNSPECIFIED, NONE, CASCADE, NULLIFY}: especifica la acción del gestor de base de datos al modificar la clave primaria de filas de la tabla de la base de datos correspondiente a la entidad referenciada. Su valor es uno de los elementos de la enumeración **OnUpdateAction**. Seleccione NONE, CASCADE o NULLIFY para impedir la actualización cuando existan referencias a la instancia modificada; actualizar las filas que contienen referencias a la instancia modificada,

colocando el nuevo valor de la clave primaria en la columna correspondiente a la referencia; o actualizar las filas que contienen referencias a la instancia modificada, colocando valor nulo en la columna correspondiente a la referencia, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es NONE.

Anotación InactiveIndicator

La anotación **InactiveIndicator** se utiliza para designar una propiedad como indicador de inactividad (eliminación lógica) de la entidad. Cada entidad puede tener un solo indicador de inactividad. Solo se puede designar como indicador de inactividad a propiedades de la clase **BooleanProperty**. Esta anotación no tiene elementos.

Anotación ManyToOne

La anotación **ManyToOne** se utiliza para establecer atributos de una referencia (propiedad que hace referencia a otra entidad) para relaciones con cardinalidad *varios-a-uno*. Los elementos de la anotación son**:**

- **navigability** {UNSPECIFIED, BIDIRECTIONAL, UNIDIRECTIONAL}: especifica la navegabilidad entre las entidades relacionadas. Su valor es uno de los elementos de la enumeración **Navigability**. Seleccione BIDIRECTIONAL o UNIDIRECTIONAL para especificar navegabilidad bidireccional o unidireccional, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es UNIDIRECTIONAL. La navegabilidad bidireccional utiliza apuntadores en ambas entidades relacionadas para permitir ir de una a otra, en ambos sentidos. La navegabilidad unidireccional solo utiliza apuntadores en la entidad que contiene la referencia (el extremo *varios* de la relación) hacia la entidad referenciada (el extremo *uno* de la relación).
- **view** {UNSPECIFIED, TABLE, TABLE_AND_DETAIL, NONE}: especifica la disponibilidad de vistas (páginas) Maestro/Detalle de la entidad que tengan a la entidad referenciada como maestro. Su valor es uno de los elementos de la enumeración **MasterDetailView**. Seleccione TABLE, TABLE_AND_DETAIL o NONE para especificar solo vistas tabulares, vistas tabulares y detalladas, o ninguna vista, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TABLE, si la entidad es del mismo tipo de recurso que la entidad referenciada (maestro) y además es existencialmente independiente (vea Anotación EntityClass); TABLE_AND_DETAIL, si la entidad es del mismo tipo de recurso que la entidad referenciada (maestro) pero no es existencialmente independiente (vea Anotación EntityClass); NONE en los demás casos.

Anotación NameProperty

La anotación **NameProperty** se utiliza para designar una propiedad como propiedad nombre de la entidad. Cada entidad puede tener una sola propiedad nombre. Solo se puede designar como propiedad nombre a propiedades de la clase **StringProperty**. Esta anotación no tiene elementos.

Anotación NumericField

La anotación **NumericField** se utiliza para establecer atributos de propiedades numéricas. Los elementos de la anotación son**:**

- **divisor** [1:N]: específica el divisor para la regla **divisorRule**. Su valor debe ser un número entero entre 1 y otro número que depende de la clase de propiedad: 100, para **ByteProperty**; 10.000, para **ShortProperty**; y 1.000.000, para las demás clases. El valor predeterminado es 100.
- **divisorRule** {UNSPECIFIED, CHECK, CEILING, FLOOR, ROUND}: especifica la regla que se debe aplicar al almacenar valores de la propiedad en la base de datos. Su valor es uno de los elementos de la enumeración **DivisorRule**. Seleccione CHECK para comprobar que el valor de la

propiedad sea múltiplo de **divisor**, Seleccione CEILING para ajustar el valor de la propiedad al menor múltiplo de **divisor** que sea mayor o igual al valor suministrado. Seleccione FLOOR para ajustar el valor de la propiedad al mayor múltiplo de **divisor** que sea menor o igual al valor suministrado. Seleccione ROUND para ajustar el valor de la propiedad al múltiplo de **divisor** más cercano al valor suministrado. Alternativamente, omita el elemento o seleccione UNSPECIFIED para no ejecutar acción alguna.

Anotación NumericKey

La anotación **NumericKey** se utiliza para designar una propiedad como clave numérica de la entidad. Cada entidad puede tener una sola clave de numérica. Las claves numéricas son claves candidatas (por lo tanto no admiten valores duplicados) que permiten solo valores numéricos. Solo se puede designar como clave numérica a propiedades de la clase **IntegerProperty**. Esta anotación no tiene elementos.

Anotación OneToOne

La anotación **OneToOne** se utiliza para establecer atributos de una referencia (propiedad que hace referencia a otra entidad) para relaciones con cardinalidad *uno-a-uno*. Esta anotación tiene un único elemento, el cual se describe a continuación:

- **navigability** {UNSPECIFIED, BIDIRECTIONAL, UNIDIRECTIONAL}: especifica la navegabilidad entre las entidades relacionadas. Su valor es uno de los elementos de la enumeración **Navigability**. Seleccione BIDIRECTIONAL o UNIDIRECTIONAL para especificar navegabilidad bidireccional o unidireccional, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es UNIDIRECTIONAL. La navegabilidad bidireccional utiliza apuntadores en ambas entidades relacionadas para permitir ir de una a otra, en ambos sentidos. La navegabilidad unidireccional solo utiliza apuntadores en la entidad que contiene la referencia hacia la entidad referenciada.

Anotación OwnerProperty

La anotación **OwnerProperty** se utiliza para designar una referencia (propiedad que hace referencia a otra entidad) como propiedad propietario de la entidad. Cada entidad puede tener una sola propiedad propietario. Solo se puede designar como propiedad propietario a referencias a la entidad **Usuario** del proyecto (vea Método setUserEntityClass). Al establecer la propiedad propietario, la entidad se convierte en *personalizable*, es decir, en una entidad que permite implementar *control de acceso personalizado* (por usuario) a las instancias de la entidad. Esta anotación no tiene elementos.

Anotación ParentProperty

La anotación **ParentProperty** se utiliza para designar una referencia (propiedad que hace referencia a otra entidad) como propiedad padre de la entidad. Cada entidad puede tener una sola propiedad padre. Solo se puede designar como propiedad padre a referencias de varios-a-uno de la entidad a sí misma. Al establecer la propiedad padre, la entidad se convierte en *jerárquica*. Esta anotación no tiene elementos.

Anotación PrimaryKey

La anotación **PrimaryKey** se utiliza para designar una propiedad como clave primaria de la entidad. Cada entidad puede tener una sola clave primaria. Solo se puede designar como clave primaria a propiedades de dos clases: **IntegerProperty** y **LongProperty**. Al utilizar la plataforma **jee1**, si la entidad representa una enumeración y, por lo tanto, su correspondiente meta-entidad extiende, directa o indirectamente, la clase **AbstractPersistentEnumerationEntity**, entonces debe tener como clave primaria una propiedad de la clase **IntegerProperty**; las demás entidades deben tener como clave primaria una propiedad de la clase **LongProperty**. Esta anotación no tiene elementos.

Anotación PropertyField

La anotación **PropertyField** se utiliza para establecer atributos básicos de la propiedad. Los elementos de la anotación son**:**

- **access** {UNSPECIFIED, RESTRICTED_WRITING, RESTRICTED_READING}: especifica el tipo de control de acceso de la propiedad. Su valor es uno de los elementos de la enumeración **PropertyAccess**. Seleccione RESTRICTED_WRITING o RESTRICTED_READING para especificar acceso restringido de escritura o lectura, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para especificar acceso no restringido.
- **auditable** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad se debe incluir, o no, en las pistas de auditoría de las funciones de **insert** y **update** de la tabla de la base de datos que corresponde a la entidad. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para incluir la propiedad; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE para propiedades que corresponden a "objetos binarios" o a contraseñas; y TRUE para las demás propiedades.
- **password** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, una contraseña. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es una contraseña; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **hidden** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad permanece, o no, oculta en las vistas (páginas) e informes. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad permanece oculta; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **required** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, obligatoriamente requerida por las vistas (páginas) de registro. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es obligatoriamente requerida; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE si la propiedad admite nulos (vea Anotación ColumnField) o tiene valor por omisión (vea Método setDefaultValue); en caso contrario es TRUE.
- **create** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, requerida por la operación **insert** de las vistas (páginas) de registro. Este elemento es relevante solo si la propiedad es insertable (vea Anotación ColumnField). Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es requerida por la operación **insert**; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE si la propiedad no es requerida (vea el elemento **required** de esta misma anotación) o no es insertable (vea Anotación ColumnField) o está enlazada a un parámetro de un proceso de instancia (vea Anotación ParameterField); en caso contrario es TRUE.
- **update** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, requerida por la operación **update** de las vistas (páginas) de registro. Este elemento es relevante solo si la propiedad es actualizable (vea Anotación ColumnField). Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es requerida por la operación **update**; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE si la propiedad no es actualizable (vea Anotación ColumnField) o está enlazada a un parámetro de un proceso de instancia (vea Anotación ParameterField); en caso contrario es TRUE.
- **search** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, un criterio de búsqueda básica en las vistas (páginas) de consulta y registro. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es un criterio de búsqueda básica; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del

atributo es TRUE si la propiedad no admite valores duplicados (vea Anotación ColumnField); o es una clave única (vea Anotación UniqueKey); o es la clave de negocio (vea Anotación BusinessKey); o es la clave numérica (vea Anotación NumericKey); o es la clave alfanumérica (vea Anotación CharacterKey); o es la propiedad nombre (vea Anotación NameProperty); o es la columna discriminadora (vea Anotación DiscriminatorColumn); o es el indicador de inactividad (vea Anotación InactiveIndicator); o es visible en las vistas (páginas) de consulta y registro tabular (vea el elemento **table** de esta misma anotación); en caso contrario, o si la propiedad es una contraseña (vea el elemento **password** de esta misma anotación), es FALSE.

- **filter** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, un criterio de búsqueda avanzada en las vistas (páginas) de consulta y registro. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es un criterio de búsqueda avanzada; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE si la propiedad es una contraseña (vea el elemento **password** de esta misma anotación); en caso contrario es TRUE.
- **table** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, visible en las vistas (páginas) de consulta y registro tabular. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es visible; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE si la propiedad no admite valores duplicados (vea Anotación ColumnField); o es una clave única (vea Anotación UniqueKey); o es la clave de negocio (vea Anotación BusinessKey); o es la clave numérica (vea Anotación NumericKey); o es la clave alfanumérica (vea Anotación CharacterKey); o es la propiedad nombre (vea Anotación NameProperty); o es la columna discriminadora (vea Anotación DiscriminatorColumn); o es el indicador de inactividad (vea Anotación InactiveIndicator); o es requerida (vea el elemento **required** de esta misma anotación); en caso contrario es FALSE.
- **detail** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, visible en las vistas (páginas) de consulta y registro detallado. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es un criterio de búsqueda avanzada; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.
- **report** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, incluida en el informe producido por la operación **report** de las vistas (páginas) de consulta y registro. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es incluida; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE si la propiedad no admite valores duplicados (vea Anotación ColumnField); o es una clave única (vea Anotación UniqueKey); o es la clave de negocio (vea Anotación BusinessKey); o es la clave numérica (vea Anotación NumericKey); o es la clave alfanumérica (vea Anotación CharacterKey); o es la propiedad nombre (vea Anotación NameProperty); o es la columna discriminadora (vea Anotación DiscriminatorColumn); o es el indicador de inactividad (vea Anotación InactiveIndicator); o es requerida (vea el elemento **required** de esta misma anotación); en caso contrario, o si la propiedad es una contraseña (vea el elemento **password** de esta misma anotación), es FALSE.
- **export** {UNSPECIFIED, TRUE, FALSE}: indica si la propiedad es, o no, incluida en el informe producido por la operación **export** de las vistas (páginas) de consulta y registro. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es incluida; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE si la propiedad es una contraseña (vea el elemento **password** de esta misma anotación); en caso contrario es TRUE.
- **submit** {UNSPECIFIED, TRUE, FALSE}: indica si propiedad es un disparador, es decir, si las vistas (páginas) de registro envían la información al servidor de aplicaciones inmediatamente que el valor de esta propiedad es modificado. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si la propiedad es un disparador; en caso contrario, seleccione

FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.

- **defaultCondition** {IF_NULL_ON_INSERT, IF_NULL_ON_UPDATE, IF_NULL, UNCONDITIONALLY_ON_INSERT, UNCONDITIONALLY_ON_UPDATE, UNCONDITIONALLY}: especifica en qué circunstancias aplicar el valor por omisión de la propiedad. Su valor es uno de los elementos de la enumeración **DefaultCondition**. Seleccione NULL_ON_INSERT, IF_NULL_ON_UPDATE, IF_NULL, UNCONDITIONALLY_ON_INSERT, UNCONDITIONALLY_ON_UPDATE o UNCONDITIONALLY para aplicar el valor por omisión si el valor es nulo en la operación **insert**; o si el valor es nulo en la operación **update**; o si el valor es nulo, tanto en **insert** como en **update**; o incondicionalmente en la operación **insert**; o incondicionalmente en la operación **update**; o incondicionalmente tanto en **insert** como en **update**, respectivamente. Alternativamente, omita el elemento para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es IF_NULL.
- **sequence** [**0:2.147.483.647**]: específica el número de secuencia o posición relativa en la que se muestra la propiedad en las vistas (páginas) de consulta y registro. Su valor debe ser un número entero entre 0 y 2.147.483.647. Alternativamente, omita el elemento para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es 0. Si todas las propiedades tienen el mismo número de secuencia (0 o cualquier otro), entonces las vistas las muestran en el orden en el mismo orden en el que las meta-propiedades están definidas en la meta-entidad.

Anotación SegmentProperty

La anotación **SegmentProperty** se utiliza para designar una referencia (propiedad que hace referencia a otra entidad) como propiedad segmento de la entidad. Cada entidad puede tener una sola propiedad segmento. Se puede designar como propiedad segmento a referencias a cualquier entidad del proyecto. Al establecer la propiedad segmento, la entidad se convierte en *segmentable*, es decir, en una entidad que permite implementar *control de acceso segmentado* (por segmento o conjunto de segmentos) a las instancias de la entidad. Esta anotación no tiene elementos.

Anotación StringField

La anotación **StringField** se utiliza para establecer atributos de propiedades **StringProperty**. Los elementos de la anotación son**:**

- **minLength** [0:8.000]: específica la cantidad mínima de caracteres que deben tener los valores de la propiedad. Su valor debe ser un número entero entre 0 y 8.000. El valor predeterminado es 0.
- **maxLength**: [1:8.000]: específica la cantidad máxima de caracteres que pueden tener los valores de la propiedad. Su valor debe ser un número entero entre 0 y 8.000. El valor predeterminado es 8.000.
- **regex**: especifica la expresión regular que deben satisfacer los valores de la propiedad. Para más información sobre expresiones regulares consulte la documentación de Java (la página *Regular Expressions* es un buen punto de partida).
- **letterCase** {UNSPECIFIED, LOWER, UPPER, CAPITALIZED}: especifica la conversión que se debe realizar al almacenar valores de la propiedad en la base de datos. Su valor es uno de los elementos de la enumeración **LetterCase**. Seleccione LOWER, UPPER o CAPITALIZED para convertir todos los caracteres a minúsculas, todos a mayúsculas, o para capitalizar (convertir el primer carácter de cada palabra a mayúscula y el resto a minúsculas), respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para no ejecutar conversión alguna.
- **allowDiacritics** {UNSPECIFIED, TRUE, FALSE}: indica si se permiten, o no, signos diacríticos al almacenar valores de la propiedad en la base de datos. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir signos diacríticos; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.

Anotación TimeField

La anotación **TimeField** se utiliza para establecer atributos de propiedades **TimeProperty**. Esta anotación tiene un único elemento, el cual se describe a continuación:

- **precision** [0:6]: específica la precisión o cantidad de decimales (en los segundos) de la propiedad. Su valor debe ser un número entero entre 0 y 6. El valor predeterminado dependerá de cada plataforma.

Anotación TimestampField

La anotación **TimestampField** se utiliza para establecer atributos de propiedades **TimestampProperty**. Esta anotación tiene un único elemento, el cual se describe a continuación:

- **precision** [0:6]: específica la precisión o cantidad de decimales (en los segundos) de la propiedad. Su valor debe ser un número entero entre 0 y 6. El valor predeterminado dependerá de cada plataforma.

Anotación UniqueKey

La anotación **UniqueKey** se utiliza para designar una propiedad como clave única de la entidad. Una entidad puede tener varias claves únicas. Se puede designar como clave única a propiedades de cualquier clase. Esta anotación no tiene elementos.

Anotación UrlProperty

La anotación **UrlProperty** se utiliza para designar una propiedad como propiedad URL de la entidad. Cada entidad puede tener una sola propiedad URL. Solo se puede designar como propiedad URL a propiedades de la clase **StringProperty**. Los elementos de la anotación son:

- **urlType** {UNSPECIFIED, INTERNAL, EXTERNAL}: indica el tipo de URL. Su valor es uno de los elementos de la enumeración **UrlType**. Seleccione INTERNAL si la URL corresponde a una vista (página) de la aplicación generada; en caso contrario seleccione EXTERNAL. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es EXTERNAL.
- **urlDisplayType** {UNSPECIFIED, TEXT, HYPERLINK, BUTTON}: indica el tipo de componente que se utiliza para mostrar el valor de la propiedad en las vistas (páginas) de consulta y registro. Su valor es uno de los elementos de la enumeración **UrlDisplayType**. Seleccione TEXT para utilizar un cuadro de texto. Seleccione HYPERLINK para utilizar un hipervínculo. Seleccione BUTTON para utilizar botón de acción. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es HYPERLINK.

Anotación VersionProperty

La anotación **VersionProperty** se utiliza para designar una propiedad como propiedad versión de la entidad. Cada entidad puede tener una sola propiedad versión. Solo se puede designar como propiedad versión a propiedades de la clase **LongProperty**. Esta anotación no tiene elementos.

Definición de atributos de propiedades mediante métodos

Una parte de los atributos de una propiedad se establecen decorando la meta-propiedad con anotaciones, tal como se describe en la sección Definición de atributos de propiedades mediante anotaciones de este mismo documento. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Los métodos para establecer atributos de meta-propiedades se dividen en los siguientes grupos:

- Métodos relacionados con la interfaz de la aplicación:
  - setDefaultLabel
  - setDefaultShortLabel
  - setDefaultDescription
  - setDefaultShortDescription
  - setDefaultTooltip

- Métodos para establecer filtros:
  - setModifyingFilter
  - setNullifyingFilter
  - setRenderingFilter
  - setRequiringFilter
  - setSearchQueryFilter

- Métodos para establecer valores:
  - setDefaultValue
  - setInitialValue
  - setMaxValue
  - setMinValue

- Métodos relacionados con la base de datos:
  - setSqlName

Método setDefaultDescription

El método **setDefaultDescription** de la meta-propiedad se utiliza para establecer la descripción de la propiedad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la propiedad. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la propiedad.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-propiedad se utiliza para establecer la descripción corta de la propiedad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la propiedad. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la propiedad.

Método setDefaultLabel

El método **setDefaultLabel** de la meta-propiedad se utiliza para establecer la etiqueta de la propiedad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultLabel** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultLabel(String defaultLabel)

Establece la etiqueta de la propiedad. Su único parámetro es:

- **defaultLabel**: sustantivo singular que se usa como etiqueta de la propiedad.

Método setDefaultShortLabel

El método **setDefaultShortLabel** de la meta-propiedad se utiliza para establecer la etiqueta corta de la propiedad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultShortLabel** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortLabel(String defaultLabel)

Establece la etiqueta corta de la propiedad. Su único parámetro es:

- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta de la propiedad.

Método setDefaultTooltip

El método **setDefaultTooltip** de la meta-propiedad se utiliza para establecer la descripción emergente (*tooltip*) de la propiedad que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultTooltip** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultTooltip(String defaultTooltip)

Establece la descripción emergente de la propiedad. Su único parámetro es:

- **defaultTooltip**: una o más oraciones que describen muy brevemente la propiedad.

Método setDefaultValue

El método **setDefaultValue** de la meta-propiedad se utiliza para establecer el valor por omisión de la propiedad. El valor por omisión se utiliza al almacenar el valor de la propiedad en la base de datos, dependiendo de la opción seleccionada para el elemento **defaultCondition** de la anotación PropertyField.

El método **setDefaultValue** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene varias firmas para cada clase de meta-propiedad (una para cada tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase de la meta-propiedad	Tipo de dato del parámetro value (<T>)
BooleanProperty	java.lang.Boolean SpecialBooleanValue BooleanExpression
CharacterProperty StringProperty	java.lang.String SpecialCharacterValue CharacterExpression
BigDecimalProperty BigIntegerProperty ByteProperty DoubleProperty FloatProperty IntegerProperty LongProperty ShortProperty	java.lang.Number SpecialNumericValue NumericExpression
DateProperty TimeProperty TimestampProperty	java.util.Date SpecialTemporalValue TemporalExpression
Entity (meta-entidad)	Entity SpecialEntityValue EntityExpression Instance

setDefaultValue(<T> value)

Establece el valor por omisión de la propiedad. Su único parámetro es:

- **value**: valor por omisión.

Método setInitialValue

El método **setInitialValue** de la meta-propiedad se utiliza para establecer el valor inicial de la propiedad. El valor inicial se utiliza para inicializar el valor de la propiedad en la operación **insert** de las vistas (páginas) de registro.

El método **setInitialValue** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene varias firmas para cada clase de meta-propiedad (una para cada tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase de la meta-propiedad	Tipo de dato del parámetro value (<T>)
BooleanProperty	java.lang.Boolean SpecialBooleanValue BooleanExpression
CharacterProperty StringProperty	java.lang.String SpecialCharacterValue CharacterExpression

Clase de la meta-propiedad	Tipo de dato del parámetro value (<T>)
BigDecimalProperty BigIntegerProperty ByteProperty DoubleProperty FloatProperty IntegerProperty LongProperty ShortProperty	java.lang.Number SpecialNumericValue NumericExpression
DateProperty TimeProperty TimestampProperty	java.util.Date SpecialTemporalValue TemporalExpression
Entity (meta-entidad)	Entity SpecialEntityValue EntityExpression Instance

setInitialValue(<T> value)

Establece el valor inicial de la propiedad. Su único parámetro es:

- **value**: valor inicial.

Método setMaxValue

El método **setMaxValue** de la meta-propiedad se utiliza para establecer el valor máximo de una propiedad numérica.

El método **setMaxValue** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene varias firmas para cada clase de meta-propiedad (una para cada tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase de la meta-propiedad	Tipo de dato del parámetro value (<T>)
BigDecimalProperty BigIntegerProperty ByteProperty DoubleProperty FloatProperty IntegerProperty LongProperty ShortProperty	Number NumericExpression

setMaxValue(<T> value)

Establece el valor máximo de la propiedad. Su único parámetro es:

- **value**: valor máximo.

Método setMinValue

El método **setMinValue** de la meta-propiedad se utiliza para establecer el valor mínimo de una propiedad numérica.

El método **setMinValue** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene varias firmas para cada clase de meta-propiedad (una para cada tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase de la meta-propiedad	Tipo de dato del parámetro value (<T>)
BigDecimalProperty BigIntegerProperty ByteProperty DoubleProperty FloatProperty IntegerProperty LongProperty ShortProperty	Number NumericExpression

setMinValue(<T> value)

Establece el valor mínimo de la propiedad. Su único parámetro es:

- **value**: valor mínimo.

Método setModifyingFilter

El método **setModifyingFilter** de la meta-propiedad se utiliza para establecer el filtro de modificación de la propiedad en las vistas (páginas) de registro de la entidad. En las instancias de la entidad que no cumplen con los criterios del filtro, la propiedad será de solo-lectura (*read-only*).

El método **setModifyingFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setModifyingFilter(BooleanExpression filter)

Establece el filtro modificación de la propiedad. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setNullifyingFilter

El método **setNullifyingFilter** de la meta-propiedad se utiliza para establecer el filtro de anulación de la propiedad en las vistas (páginas) de registro de la entidad. En las instancias de la entidad que cumplen con los criterios del filtro, la propiedad será anulada.

El método **setNullifyingFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setNullifyingFilter(BooleanExpression filter)

Establece el filtro anulación de la propiedad. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setRenderingFilter

El método **setRenderingFilter** de la meta-propiedad se utiliza para establecer el filtro de presentación de la propiedad en las vistas (páginas) de consulta y registro de la entidad. En las instancias de la entidad que no cumplen con los criterios del filtro, la propiedad será invisible.

El método **setRenderingFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setRenderingFilter(BooleanExpression filter)

Establece el filtro presentación de la propiedad. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setRequiringFilter

El método **setRequiringFilter** de la meta-propiedad se utiliza para establecer el filtro de obligatoriedad de la propiedad en las vistas (páginas) de registro de la entidad. En las instancias de la entidad que cumplen con los criterios del filtro, la propiedad será obligatoriamente requerida.

El método **setRequiringFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setRequiringFilter(BooleanExpression filter)

Establece el filtro de obligatoriedad de la propiedad. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setSearchQueryFilter

El método **setSearchQueryFilter** de la meta-propiedad se utiliza para establecer el filtro de búsqueda del valor de la referencia (propiedad que hace referencia a otra entidad) en las vistas (páginas) de registro de la entidad.

El método **setSearchQueryFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setSearchQueryFilter(BooleanExpression filter)

Establece el filtro de búsqueda de la propiedad. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Método setSqlName

El método **setSqlName** de la meta-propiedad se utiliza para establecer el nombre de la columna de la base de datos correspondiente a la propiedad. Si este método no es ejecutado, el nombre de la columna se determina a partir del nombre de la meta-propiedad, sustituyendo cada letra mayúscula por un guion bajo (*underscore*) seguido de la letra convertida en minúscula.

El método **setSqlName** debe ejecutarse en el método **settleProperties** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setSqlName(String sqlName)

Establece el nombre de la columna correspondiente a la propiedad. Su único parámetro es:

- **sqlName**: nombre de la columna.

Propiedades requeridas en entidades persistentes estándar

La meta-entidad de una entidad persistente estándar debe incluir una clave primaria de tipo **LongProperty** (vea Anotación PrimaryKey). También debe incluir una propiedad versión (vea Anotación VersionProperty).

Por otra parte, se recomienda (no es requerido) incluir una clave de negocio (vea Anotación BusinessKey) y además, siempre que tenga sentido, una propiedad nombre (vea Anotación NameProperty).

Propiedades requeridas en entidades persistentes para enumeraciones

La meta-entidad de una entidad persistente correspondiente a una enumeración debe incluir una clave primaria de tipo **IntegerProperty** (vea Anotación PrimaryKey). También debe incluir una clave de negocio (vea Anotación BusinessKey).

Definición de propiedades enlazadas

Indicador de Inactividad

El indicador de inactividad (eliminación lógica) de la entidad puede ser el indicador de inactividad de una de sus entidades referenciadas. Cuando éste sea el caso, en lugar de utilizar la anotación InactiveIndicator se utiliza el método **linkForeignInactiveIndicatorProperty**.

El método **linkForeignInactiveIndicatorProperty** de la meta-entidad debe ejecutarse en el método **settleLinks** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

linkForeignInactiveIndicatorProperty(BooleanProperty foreignInactiveIndicatorProperty)

Enlaza el indicador de inactividad al indicador de inactividad de una entidad referenciada. Su único parámetro es:

- **foreignInactiveIndicatorProperty**: indicador de inactividad de una entidad referenciada (vea Anotación InactiveIndicator).

Propiedad Propietario

La propiedad propietario de la entidad puede ser la propiedad propietario de una de sus entidades referenciadas. Cuando éste sea el caso, en lugar de utilizar la anotación OwnerProperty se utiliza el método **linkForeignOwnerProperty**.

El método **linkForeignOwnerProperty** de la meta-entidad debe ejecutarse en el método **settleLinks** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

linkForeignOwnerProperty(Entity foreignOwnerProperty)

Enlaza la propiedad propietario a la propiedad propietario de una entidad referenciada. Su único parámetro es:

- **foreignOwnerProperty**: propiedad propietario de una entidad referenciada (vea Anotación OwnerProperty).

Propiedad Segmento

La propiedad segmento de la entidad puede ser la propiedad segmento de una de sus entidades referenciadas. Cuando éste sea el caso, en lugar de utilizar la anotación SegmentProperty se utiliza el método **linkForeignSegmentProperty**.

El método **linkForeignSegmentProperty** de la meta-entidad debe ejecutarse en el método **settleLinks** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

linkForeignSegmentProperty(Entity foreignSegmentProperty)

Enlaza la propiedad segmento a la propiedad segmento de una entidad referenciada. Su único parámetro es:

- **foreignSegmentProperty**: propiedad segmento de una entidad referenciada (vea Anotación SegmentProperty).

Definición de grupos de propiedades

Cada uno de los grupos de propiedades de la entidad se debe definir como un campo de la meta-entidad. A los campos de la meta-entidad que definen grupos de propiedades se les conoce como **meta-grupos**.

Para definir meta-grupos se utiliza la clase **Tab** o una clase Java que extienda, directa o indirectamente, la clase **Tab**.

Los meta-grupos generan pestañas (*tabs*) en las vistas (páginas) de consulta y registro detallado de la entidad.

Todo meta-grupo se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-grupos.

```
public class Persona extends PersistentEntityBase {

    constructores

    propiedades

    Tab tab1, tab2;

    @Override
    protected void settleTabs() {
        super.settleTabs();
        tab1.setDefaultLabel("datos básicos");
        tab1.newTabField(cedula, nombre, fechaNacimiento, fechaDefuncion,
            sexo, limiteCredito);
        tab2.setDefaultLabel("información de contacto");
        tab2.setDefaultShortLabel("contacto");
        tab2.newTabField(telefono, direccion, estado, pais);
    }

}
```

Definición de atributos de grupos de propiedades

Los atributos de un grupo de propiedades se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** del meta-grupo se utiliza para establecer la descripción del grupo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleTabs** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción del grupo. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen el grupo.

Método setDefaultShortDescription

El método **setDefaultShortDescription** del meta-grupo se utiliza para establecer la descripción corta del grupo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleTabs** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta del grupo. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente el grupo.

Método setDefaultLabel

El método **setDefaultLabel** del meta-grupo se utiliza para establecer la etiqueta del grupo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultLabel** debe ejecutarse en el método **settleTabs** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultLabel(String defaultLabel)

Establece la etiqueta del grupo. Su único parámetro es:

- **defaultLabel**: sustantivo singular que se usa como etiqueta del grupo.

Método setDefaultShortLabel

El método **setDefaultShortLabel** del meta-grupo se utiliza para establecer la etiqueta corta del grupo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultShortLabel** debe ejecutarse en el método **settleTabs** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortLabel(String defaultLabel)

Establece la etiqueta corta del grupo. Su único parámetro es:

- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta del grupo.

Método setDefaultTooltip

El método **setDefaultTooltip** del meta-grupo se utiliza para establecer la descripción emergente (*tooltip*) del grupo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultTooltip** debe ejecutarse en el método **settleTabs** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultTooltip(String defaultTooltip)

Establece la descripción emergente del grupo. Su único parámetro es:

- **defaultTooltip**: una o más oraciones que describen muy brevemente el grupo.

Método setRenderingFilter

El método **setRenderingFilter** del meta-grupo se utiliza para establecer el filtro de presentación del grupo en las vistas (páginas) de consulta y registro de la entidad. En las instancias de la entidad que no cumplen con los criterios del filtro, el grupo será invisible.

El método **setRenderingFilter** debe ejecutarse en el método **settleFilters** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setRenderingFilter(BooleanExpression filter)

Establece el filtro presentación del grupo. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-entidad, vea la sección Definición de expresiones de este mismo documento.

Definición de propiedades de grupos

Las propiedades que integran un grupo de propiedades se establecen mediante el método **newTabField**, tal como se describe a continuación.

Método newTabField

El método **newTabField** del meta-grupo se utiliza para agregar propiedades al grupo. Los grupos pueden estar integrados por una o más propiedades.

El método **newTabField** debe ejecutarse en el método **settleTabs** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

newTabField(Property... property)

Agrega una o más propiedades al grupo. Su único parámetro es:

- **property**: una o más propiedades (separadas por una coma) de la entidad.

Definición de vistas

Cada una de las vistas de la entidad se debe definir como un campo de la meta-entidad. A los campos de la meta-entidad que definen vistas se les conoce como **meta-vistas**.

Para definir meta-vistas se utiliza la clase **View** o una clase Java que extienda, directa o indirectamente, la clase **View**.

Las meta-vistas se utilizan en la definición de operaciones de negocio para generar archivos e informes, tal como se explica en las anotaciones ExportOperationClass y ReportOperationClass, respectivamente.

Toda meta-vista se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-vistas.

```
public class Persona extends PersistentEntityBase {

    constructores

    propiedades

    View v1;

    @Override
    protected void settleViews() {
        super.settleViews();
        v1.newControlField(pais.codigo, SortOption.ASC);
        v1.newHeadingField(pais.nombre, pais.codigo);
        v1.newControlField(estado.codigo, SortOption.ASC);
        v1.newHeadingField(estado.nombre, estado.codigo);
        v1.newDetailField(cedula, SortOption.ASC);
        v1.newDetailField(nombre);
        v1.newDetailField(fechaNacimiento);
        v1.newDetailField(limiteCredito,
            ViewFieldAggregation.AVERAGE_DEVIATION_MINIMUM_MAXIMUM);
    }
}
```

Definición de atributos de vistas

Los atributos de una vista se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** de la meta-vista se utiliza para establecer la descripción de la vista que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la vista. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la vista.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-vista se utiliza para establecer la descripción corta de la vista que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la vista. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la vista.

Método setDefaultLabel

El método **setDefaultLabel** de la meta-vista se utiliza para establecer la etiqueta de la vista que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultLabel** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultLabel(String defaultLabel)

Establece la etiqueta de la vista. Su único parámetro es:

- **defaultLabel**: sustantivo singular que se usa como etiqueta de la vista.

Método setDefaultShortLabel

El método **setDefaultShortLabel** de la meta-vista se utiliza para establecer la etiqueta corta de la vista que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultShortLabel** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortLabel(String defaultLabel)

Establece la etiqueta corta de la vista. Su único parámetro es:

- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta de la vista.

Método setDefaultTooltip

El método **setDefaultTooltip** de la meta-vista se utiliza para establecer la descripción emergente (*tooltip*) de la vista que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultTooltip** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultTooltip(String defaultTooltip)

Establece la descripción emergente de la vista. Su único parámetro es:

- **defaultTooltip**: una o más oraciones que describen muy brevemente la vista.

Definición de propiedades de vistas

Las propiedades que integran una vista se establecen mediante los métodos que se describen a continuación.

Método newControlField

El método **newControlField** de la meta-vista se utiliza para agregar propiedades de control a la vista. Las propiedades de control definen los grupos de agregación de la vista. Las vistas pueden o no tener propiedades de control.

El método **newControlField** retorna el campo de la vista (objeto de la clase **ViewField**) correspondiente a la propiedad. Los atributos de los meta-campos se establecen mediante los métodos descritos en la sección Definición de atributos de campos de vistas.

El método **newControlField** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene varias firmas, las cuales se describen a continuación.

ViewField newControlField(Property column)

Agrega una propiedad de control a la vista. Su único parámetro es:

- **column**: una de las propiedades de la entidad; el orden será ascendente.

ViewField newControlField(Property column, SortOption sort)

Agrega una propiedad de control a la vista. Sus parámetros son:

- **column**: una de las propiedades de la entidad.
- **sort** {ASC, DESC}: especifica el orden de la propiedad en la vista. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para establecer el orden como ascendente o descendente, respectivamente.

Método newHeadingField

Inicialmente, los encabezados de los grupos de agregación de los informes generados a partir de la vista tienen una sola propiedad; ésta es la propiedad de control que se utilizó para definir el grupo. El método **newHeadingField** de la meta-vista se utiliza para agregar propiedades a los encabezados de los grupos de agregación.

El método **newHeadingField** retorna el campo de la vista (objeto de la clase **ViewField**) correspondiente a la propiedad. Los atributos de los meta-campos se establecen mediante los métodos descritos en la sección Definición de atributos de campos de vistas.

El método **newHeadingField** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

ViewField newHeadingField(Property column, Property group)

Agrega una propiedad a un encabezado de grupo de la vista. Sus parámetros son:

- **column**: una de las propiedades de la entidad.
- **group**: propiedad de control que se utilizó para definir el grupo al que se desea agregar la propiedad.

Método newDetailField

El método **newDetailField** de la meta-vista se utiliza para agregar propiedades de detalle a la vista. Las vistas pueden o no tener propiedades de detalle.

El método **newDetailField** retorna el campo de la vista (objeto de la clase **ViewField**) correspondiente a la propiedad. Los atributos de los meta-campos se establecen mediante los métodos descritos en la sección Definición de atributos de campos de vistas.

El método **newDetailField** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene varias firmas, las cuales se describen a continuación.

ViewField newDetailField(Property column)

Agrega una propiedad de detalle a la vista. Su único parámetro es:

- **column**: una de las propiedades de la entidad.

ViewField newDetailField(Property column, SortOption sort)

Agrega una propiedad de detalle a la vista. Sus parámetros son:

- **column**: una de las propiedades de la entidad.
- **sort** {ASC, DESC}: especifica el orden de la propiedad en la vista. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para establecer el orden como ascendente o descendente, respectivamente.

ViewField newDetailField(Property column, ViewFieldAggregation aggregation)

Agrega una propiedad de detalle a la vista. Sus parámetros son:

- **column**: una de las propiedades de la entidad.
- **aggregation**: especifica las funciones de agregación de la propiedad en la vista. Su valor es uno de los elementos de la enumeración **ViewFieldAggregation**. Estos son:

Elemento	Funciones de agregación
COUNT	Cuenta
MINIMUM	Mínimo
MAXIMUM	Máximo
SUM	Suma
AVERAGE	Promedio
DEVIATION	Desviación Estándar
COUNT_MINIMUM_MAXIMUM	Cuenta, Mínimo, Máximo
MINIMUM_MAXIMUM	Mínimo, Máximo
SUM_COUNT_AVERAGE	Suma, Cuenta, Promedio
SUM_COUNT_AVERAGE_DEVIATION_MINIMUM_MAXIMUM	Suma, Cuenta, Promedio, Desviación Estándar, Mínimo, Máximo
AVERAGE_DEVIATION	Promedio, Desviación Estándar
AVERAGE_DEVIATION_MINIMUM_MAXIMUM	Promedio, Desviación Estándar, Mínimo, Máximo

ViewField newDetailField(Property column, ViewFieldAggregation aggregation, SortOption sort)

Agrega una propiedad de detalle a la vista. Sus parámetros son:

- **column**: una de las propiedades de la entidad.
- **aggregation**: especifica las funciones de agregación de la propiedad en la vista. Su valor es uno de los elementos de la enumeración **ViewFieldAggregation**, indicados en la firma anterior.
- **sort** {ASC, DESC}: especifica el orden de la propiedad en la vista. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para establecer el orden como ascendente o descendente, respectivamente.

Definición de atributos de campos de vistas

Los métodos **newControlField**, **newHeadingField** y **newDetailField** de la meta-vista definen las propiedades que integran la vista y retornan el campo de la vista (objeto de la clase **ViewField**) correspondiente a la propiedad. Los atributos de un campo de una vista se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** del meta-campo se utiliza para establecer la descripción del campo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción del campo. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen el campo.

Método setDefaultShortDescription

El método **setDefaultShortDescription** del meta-campo se utiliza para establecer la descripción corta del campo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta del campo. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente el campo.

Método setDefaultLabel

El método **setDefaultLabel** del meta-campo se utiliza para establecer la etiqueta del campo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultLabel** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultLabel(String defaultLabel)

Establece la etiqueta del campo. Su único parámetro es:

- **defaultLabel**: sustantivo singular que se usa como etiqueta del campo.

Método setDefaultShortLabel

El método **setDefaultShortLabel** del meta-campo se utiliza para establecer la etiqueta corta del campo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultShortLabel** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortLabel(String defaultLabel)

Establece la etiqueta corta del campo. Su único parámetro es:

- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta del campo.

Método setDefaultTooltip

El método **setDefaultTooltip** del meta-campo se utiliza para establecer la descripción emergente (*tooltip*) del campo que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultTooltip** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultTooltip(String defaultTooltip)

Establece la descripción emergente del campo. Su único parámetro es:

- **defaultTooltip**: una o más oraciones que describen muy brevemente el campo.

Método setPixels

El método **setPixels** del meta-campo se utiliza para establecer el tamaño del campo en los informes, expresado en pixeles.

El método **setPixels** debe ejecutarse en el método **settleViews** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setPixels(Integer pixels)

Establece el tamaño del campo en los informes, expresado en pixeles. Su único parámetro es:

- **pixels** [0:960]: número entero entre 0 y 960. Si el valor es 0 entonces el campo no se muestra en los informes. Si el valor es nulo, menor que 0 o mayor que 960, entonces se utiliza el tamaño predeterminado.

Tamaño predeterminado de los campos en los informes

Si no se especifica el tamaño de algún campo, o si se especifica un valor fuera del rango permitido por el método **setPixels**, el tamaño del campo en el informe se determina en función la clase de la correspondiente meta-propiedad, como se ilustra en la siguiente tabla.

Clase de la meta-propiedad	Longitud	Tamaño del campo en pixeles
BooleanProperty	No aplica	48
CharacterProperty	No aplica	48
StringProperty	<= 10	48
	<= 20	64
	<= 50	128
	<= 100	192
	> 100	256
	Indefinida	384
BigDecimalProperty	No aplica	128
BigIntegerProperty	No aplica	128
ByteProperty	No aplica	64
DoubleProperty	No aplica	128
FloatProperty	No aplica	128
IntegerProperty	No aplica	64
LongProperty	No aplica	128
ShortProperty	No aplica	64
DateProperty	No aplica	64
TimeProperty	No aplica	64
TimestampProperty	No aplica	104

Definición de claves de acceso

Cada una de las claves de acceso de la entidad se debe definir como un campo de la meta-entidad. A los campos de la meta-entidad que definen las claves de acceso se les conoce como **meta-claves**.

Para definir meta-claves se utiliza la clase **Key** o una clase Java que extienda, directa o indirectamente, la clase **Key**.

Las meta-claves generan índices en la base de datos para agilizar la búsqueda por las propiedades que la integran (las columnas indexadas).

Toda meta-clave se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-claves.

```
public class Persona extends PersistentEntityBase {

    constructores

    propiedades

    Key claveCedula, claveNombre, clavePaisEstado;

    @Override
    protected void settleKeys() {
        super.settleKeys();
        claveCedula.setUnique(true);
        claveCedula.newKeyField(cedula);
        claveNombre.setUnique(false);
        claveNombre.newKeyField(nombre, SortOption.ASC);
        clavePaisEstado.setUnique(false);
        clavePaisEstado.newKeyField(pais, estado);
    }

}
```

Definición de atributos de claves de acceso

Los atributos de una clave de acceso se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** de la meta-clave se utiliza para establecer la descripción de la clave que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleKeys** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la clave. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la clave.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-clave se utiliza para establecer la descripción corta de la clave que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleKeys** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la clave. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la clave.

Método setUnique

El método **setUnique** de la meta-clave se utiliza para establecer la unicidad de la clave. Las claves únicas no permiten valores duplicados en las propiedades que las integran.

El método **setUnique** debe ejecutarse en el método **settleKeys** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setUnique(boolean unique)

Establece la unicidad de la clave. Su único parámetro es:

- **unique**: `true`, si la clave es única; de lo contrario `false`.

Definición de propiedades de claves de acceso

Las propiedades que integran una clave de acceso se establecen mediante el método **newKeyField**, tal como se describe a continuación.

Método newKeyField

El método **newKeyField** de la meta-clave se utiliza para agregar propiedades a la clave. Las claves pueden estar integradas por una o más propiedades, y para cada propiedad se puede establecer el orden en el índice correspondiente. La posición relativa de cada propiedad en el índice está determinado por la secuencia en la que se agregan a la clave.

El método **newKeyField** debe ejecutarse en el método **settleKeys** de la meta-entidad y tiene varias firmas, las cuales se describen a continuación.

newKeyField(Property... property)

Agrega una o más propiedades a la clave. Su único parámetro es:

- **property**: una o más propiedades (separadas por una coma) de la entidad; el orden en el índice será ascendente.

newKeyField(Property property, SortOption sortOption)

Agrega una propiedad a la clave. Sus parámetros son:

- **property**: propiedad de la entidad.
- **sortOption** {ASC, DESC}: especifica el orden de la propiedad en el índice. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para establecer el orden como ascendente o descendente, respectivamente.

Definición de instancias

Normalmente solo las entidades que representan una enumeración tienen instancias definidas; más aún, toda enumeración debe tener al menos una instancia, ya que las instancias corresponden a los elementos de la enumeración.

Cada una de las instancias de la entidad se debe definir como un campo de la meta-entidad. A los campos de la meta-entidad que definen las instancias se les conoce como **meta-instancias**.

Para definir meta-instancias se utiliza la clase **Instance** o una clase Java que extienda, directa o indirectamente, la clase **Instance**.

Las meta-instancias generan filas en la tabla de la base de datos que corresponde a la entidad.

Toda meta-instancia se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

A continuación se ilustra la definición de instancias, utilizando como ejemplo una meta-entidad de nombre Sexo.

```
public class Sexo extends PersistentEnumerationEntityBase {

    constructores

    @ColumnField(nullable = Kleenean.FALSE)
    @ForeignKey(onDelete = OnDeleteAction.NONE)
    @OneToOne(navigability = Navigability.BIDIRECTIONAL)
    public Sexo opuesto;

    public Instance MASCULINO;

    public Instance FEMENINO;

    @Override
    protected void settleInstances() {
        super.settleInstances();
        MASCULINO.newInstanceField(opuesto, FEMENINO);
        FEMENINO.newInstanceField(opuesto, MASCULINO);
    }

}
```

Definición de atributos de instancias

Los atributos de una instancia se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** de la meta-instancia se utiliza para establecer la descripción de la instancia que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleInstances** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la instancia. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la instancia.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-instancia se utiliza para establecer la descripción corta de la instancia que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleInstances** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la instancia. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la instancia.

Definición de valores de propiedades de instancias

Los valores de las propiedades de cada instancia se establecen mediante el método **newInstanceField**, tal como se describe a continuación.

Normalmente no es necesario especificar los valores de la clave primaria y de la clave de negocio. Por omisión, a la clave primaria se le asigna un número secuencial en el orden en el que se definen las instancias en la entidad, comenzando a partir del número especificado en el elemento **startWith** de la anotación EntityClass; y a la clave de negocio se le asigna el nombre capitalizado (en minúsculas, con la primera letra en mayúscula) de la meta-instancia.

Método newInstanceField

El método **newInstanceField** de la meta-instancia se utiliza para especificar el valor de las propiedades de la instancia.

El método **newInstanceField** debe ejecutarse en el método **settleInstances** de la meta-entidad y tiene varias firmas, una para cada clase de meta-propiedad (y tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase de la meta-propiedad	Tipo de dato del parámetro value (<T>)
BooleanProperty	Boolean
CharacterProperty	Character
StringProperty	String
BigDecimalProperty	BigDecimal
BigIntegerProperty	BigInteger
ByteProperty	Byte
DoubleProperty	Double
FloatProperty	Float
IntegerProperty	Integer
LongProperty	Long
ShortProperty	Short
DateProperty	Date
TimeProperty	Time
TimestampProperty	Timestamp
Entity (meta-entidad)	Instance

newInstanceField(Property property, <T> value)

Establece el valor de la propiedad. Sus parámetros son:

- **property**: propiedad de la entidad.
- **value**: valor de la propiedad.

Definición de expresiones

Una expresión es una construcción compuesta de variables, operadores y llamadas a métodos, que se construye de acuerdo a la sintaxis del lenguaje Java y que, al evaluarla, produce un solo valor.

Cada una de las expresiones de la entidad se puede definir como un campo de la meta-entidad. A los campos de la meta-entidad que definen expresiones se les conoce como **meta-expresiones**.

Las meta-expresiones se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **Expression**.

La expresión de la meta-expresión, y el valor de sus atributos, se debe establecer en el método **settleExpressiones** de la meta-entidad. No todas las expresiones deben ser definidas como meta-expresiones; si la expresión solo es utilizada como parámetro de un método entonces no es necesario definirla como meta-expresión.

El tipo de dato de una expresión es el tipo de dato del valor que produce. En función del tipo de dato de la expresión que definen, las meta-expresiones se dividen en dos grupos:

- **Primitivas**: si la expresión es de un tipo primitivo o elemental, es decir, Boolean, Character, String, BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short, Date, Time o Timestamp.
- **Referencias**: si la expresión es una referencia a una entidad. Las meta-expresiones de este grupo implementan la interfaz **EntityExpression**, la cual extiende la interfaz **Expression**.

A su vez, las meta-expresiones primitivas se dividen en:

- **Booleanas**: si la expresión es de tipo Boolean. Las meta-expresiones booleanas se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **BooleanExpression** (la cual extiende la interfaz **Expression**) o alguna de sus sub-interfaces: **Check, Segment** o **State**.
- **Alfanuméricas**: si la expresión es de tipo Character o String. Las meta-expresiones alfanuméricas se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **CharacterExpression**, la cual extiende la interfaz **Expression**.
- **Numéricas**: si la expresión es de tipo BigDecimal, BigInteger, Byte, Double, Float, Integer, Long o Short. Las meta-expresiones numéricas se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **NumericExpression**, la cual extiende la interfaz **Expression**.
- **Temporales**: si la expresión es de tipo Date, Time o Timestamp. Las meta-expresiones temporales se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **TemporalExpression**, la cual extiende la interfaz **Expression**.

A las interfaces BooleanExpression, CharacterExpression, NumericExpression, TemporalExpression y EntityExpression se les conoce colectivamente como **interfaces de expresión por tipo**.

Toda meta-expresión, primitiva o referencia, se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Las meta-expresiones proveen un mecanismo sofisticado para definir valores y reglas de negocio. Específicamente, las expresiones se utilizan en métodos para definir valores de propiedades, filtros y límites:

- Métodos para definir valores de propiedades:
  - setDefaultValue
  - setInitialValue

- Métodos para definir filtros de operaciones:
  - setSelectFilter
  - setUpdateFilter
  - setDeleteFilter
  - setInsertFilter
  - setMasterDetailFilter

- Métodos para definir filtros de propiedades:
  - setModifyingFilter
  - setNullifyingFilter
  - setRenderingFilter
  - setRequiringFilter

- Métodos para definir filtros de grupos de propiedades:
  - setRenderingFilter

- Métodos para definir límites de propiedades:
  - setMaxValue
  - setMinValue

Además, las meta-expresiones booleanas se utilizan para definir restricciones, segmentos y estados de la entidad. Las restricciones son expresiones que deben ser verdaderas en todas las instancias de la entidad; por lo tanto usualmente se utilizan para generar validaciones en las vistas (páginas) de registro y en la base de datos. Los segmentos son expresiones que son verdaderas sólo en algunos casos; por lo tanto, usualmente se utilizan como parámetro de métodos para definir filtros y como productos intermedios en la construcción de expresiones complejas. Los estados son un tipo especial de segmentos, el tipo que le permite definir transiciones, que son caminos válidos entre dos estados; luego, al definir las operaciones de la entidad, se pueden especificar las transiciones realizadas por cada operación, y así se define una máquina de estados. Para definir restricciones se utilizan clases que implementen la interfaz **Check**; para definir segmentos, clases que implementen la interfaz **Segment**; y para definir estados, clases que implementen la interfaz **State**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-expresiones.

```
public class Persona extends PersistentEntityBase {

    constructores

    propiedades

    Check ck1, ck2, ck3;

    Segment varones, hembras;

    State soltero, casado, viudo;

    @Override
    protected void settleProperties() {
        super.settleProperties();
        limiteCredito.setDefaultValue(conyuge.isNull().then(10000).otherwise(15000));
        limiteCredito.setMaxValue(conyuge.isNull().then(200000).otherwise(300000));
    }

    @Override
    protected void settleExpressions() {
        super.settleExpressions();
        ck1 = fechaNacimiento.isLessOrEqualTo(SpecialTemporalValue.CURRENT_DATE);
        ck1.setDefaultErrorMessage("la fecha de nacimiento es una fecha futura");
        ck2 = fechaDefuncion.isNullOrLessOrEqualTo(SpecialTemporalValue.CURRENT_DATE);
        ck2.setDefaultErrorMessage("la fecha de defunción es una fecha futura");
        ck3 = fechaDefuncion.isNullOrGreaterOrEqualTo(fechaNacimiento);
        ck3.setDefaultErrorMessage("la fecha de defunción es menor que "
            + "la fecha de nacimiento");
        varones = sexo.isEqualTo(sexo.MASCULINO);
        varones.setDefaultErrorMessage("la persona no es de sexo masculino");
        hembras = sexo.isEqualTo(sexo.FEMENINO);
        hembras.setDefaultErrorMessage("la persona no es de sexo femenino");
        soltero = conyuge.isNull();
        soltero.setDefaultErrorMessage("la persona no está soltera");
        casado = conyuge.isNotNull();
        casado.setDefaultErrorMessage("la persona no está casada");
        viudo = casado.and(conyuge.fechaDefuncion.isNotNull());
        viudo.setDefaultErrorMessage("la persona no está viuda");
    }

}
```

Construcción de expresiones

La construcción de expresiones comienza invariablemente por la ejecución de un método de una meta-propiedad que produce una expresión; en caso de ser necesario, continúa con la ejecución de un método de la expresión obtenida, lo que produce una expresión más compleja; esto último se repite hasta construir la expresión completa.

En el ejemplo anterior, se establece el valor por omisión y el límite máximo de la meta-propiedad **limiteCredito** de tipo **BigDecimalProperty**, utilizando expresiones en las siguientes instrucciones:

```
limiteCredito.setDefaultValue(conyuge.isNull().then(10000).otherwise(15000));
limiteCredito.setMaxValue(conyuge.isNull().then(200000).otherwise(300000));
```

En ambas, primero se ejecuta el método **isNull** de la meta-propiedad **conyuge**, el cual produce una expresión de tipo **BooleanExpression**; luego se ejecuta el método **then** de esa **BooleanExpression**, el cual produce una primera expresión de tipo **NumericExpression**; finalmente, se ejecuta el método **otherwise** de esa primera **NumericExpression**, el cual produce una segunda expresión de tipo

NumericExpression, que se pasa como parámetro a los métodos **setDefaultValue** y **setMaxValue** de la meta-propiedad **limiteCredito**.

Para facilitar la construcción de expresiones, las clases que se utilizan para definir meta-entidades y meta-propiedades implementan una de las interfaces de expresión por tipo, como se muestra en la siguiente tabla.

Clase	Interfaz de expresión por tipo
BooleanProperty	BooleanExpression
CharacterProperty StringProperty	CharacterExpression
BigDecimalProperty BigIntegerProperty ByteProperty DoubleProperty FloatProperty IntegerProperty LongProperty ShortProperty	NumericExpression
DateProperty TimeProperty TimestampProperty	TemporalExpression
Entity (meta-entidad)	EntityExpression

Además de las clases que se utilizan para definir meta-entidades y meta-propiedades, existen otras clases que implementan las interfaces de expresión por tipo; las más relevantes se muestran en la siguiente tabla.

Interfaz	Clase
BooleanExpression	BooleanComparisonX BooleanConditionalX BooleanOrderedPairX BooleanScalarX
CharacterExpression	CharacterConditionalX CharacterOrderedPairX CharacterScalarX
NumericExpression	NumericConditionalX NumericOrderedPairX NumericScalarX
TemporalExpression	TemporalConditionalX TemporalOrderedPairX TemporalScalarX
EntityExpression	EntityConditionalX EntityOrderedPairX

Si en lugar de agruparlas por la clase de valor que producen lo hacemos por sus características funcionales, las expresiones se dividen en:

- **Expresiones de Comparación** (clases que implementan la interfaz **ComparisonX**)
- **Expresiones Condicionales** (clases que implementan la interfaz **ConditionalX**)
- **Expresiones Escalares** (clases que implementan la interfaz **ScalarX**)
- **Expresiones de Pares Ordenados** (clases que implementan la interfaz **OrderedPairX**)

La siguiente tabla muestra las clases de expresiones agrupadas por sus características funcionales.

Interfaz	Clase
ComaparisonX	BooleanComparisonX
ConditionalX	BooleanConditionalX CharacterConditionalX NumericConditionalX TemporalConditionalX EntityConditionalX
OrderedPairX	BooleanOrderedPairX CharacterOrderedPairX NumericOrderedPairX TemporalOrderedPairX EntityOrderedPairX
ScalarX	BooleanScalarX CharacterScalarX NumericScalarX TemporalScalarX

Expresiones de Comparación

Las expresiones de comparación son expresiones booleanas (implementan la interfaz BooleanExpression) que se utilizan para definir operaciones de comparación, es decir, operaciones que comparan dos valores, mediante un operador de comparación, y producen un valor booleano `true`, si la comparación se cumple, o un valor booleano `false`, si la comparación no se cumple. Salvo en el caso de los operadores que contemplan el valor nulo, estas operaciones producen un valor nulo si alguno de los operandos es nulo. Los operadores de comparación varían en función del tipo de dato de los valores comparados y, por lo tanto, las interfaces de expresión por tipo tienen diferentes métodos para producir expresiones de comparación. La enumeración **ComparisonOp** tiene un elemento por cada operador de comparación; los operadores utilizados por expresiones de comparación son:

- **IS_NULL**: compara el operando contra el valor nulo; la comparación se cumple si son iguales. Este operador funciona con un solo operando, ya que el segundo es siempre el valor nulo. Este es el operador utilizado por las expresiones producidas por los métodos **isNull** de las clases que implementan las interfaces de expresión por tipo.
- **IS_NOT_NULL**: compara el operando contra el valor nulo; la comparación se cumple si no son iguales. Este operador funciona con un solo operando, ya que el segundo es siempre el valor nulo. Este es el operador utilizado por las expresiones producidas por los métodos **isNotNull** de las clases que implementan las interfaces de expresión por tipo.
- **IS_TRUE**: compara el operando contra el valor booleano `true`; la comparación se cumple si son iguales. Este operador funciona con un solo operando, ya que el segundo es siempre el valor booleano `true`. Este es el operador utilizado por las expresiones producidas por los métodos **isTrue** de las clases que implementan la interfaz BooleanExpression.
- **IS_FALSE**: compara el operando contra el valor booleano `false`; la comparación se cumple si son iguales. Este operador funciona con un solo operando, ya que el segundo es siempre el valor booleano `false`. Este es el operador utilizado por las expresiones producidas por los métodos **isFalse** de las clases que implementan la interfaz BooleanExpression.
- **IS_NULL_OR_TRUE**: compara el operando contra el valor booleano `true`; la comparación se cumple si son iguales o si el primer valor es nulo. Este operador funciona con un solo operando, ya que el segundo es siempre el valor booleano `true`. Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrTrue** de las clases que implementan la interfaz BooleanExpression.
- **IS_NULL_OR_FALSE**: compara el operando contra el valor booleano `false`; la comparación se cumple si son iguales o si el primer valor es nulo. Este operador funciona con un solo operando, ya que el segundo es siempre el valor booleano `false`. Este es el operador utilizado por las

expresiones producidas por los métodos **isNullOrFalse** de las clases que implementan la interfaz BooleanExpression.

- **EQ**: compara los dos operandos; la comparación se cumple si son iguales. Este es el operador utilizado por las expresiones producidas por los métodos **isEqualTo** de las clases que implementan las interfaces de expresión por tipo.
- **NEQ**: compara los dos operandos; la comparación se cumple si no son iguales. Este es el operador utilizado por las expresiones producidas por los métodos **isNotEqualTo** de las clases que implementan las interfaces de expresión por tipo.
- **GT**: compara los dos operandos; la comparación se cumple si el primer valor es mayor que el segundo. Este es el operador utilizado por las expresiones producidas por los métodos **isGreaterThan** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **GTEQ**: compara los dos operandos; la comparación se cumple si el primer valor es mayor o igual al segundo. Este es el operador utilizado por las expresiones producidas por los métodos **isGreaterOrEqualTo** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **LT**: compara los dos operandos; la comparación se cumple si el primer valor es menor que el segundo. Este es el operador utilizado por las expresiones producidas por los métodos **isLessThan** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **LTEQ**: compara los dos operandos; la comparación se cumple si el primer valor es menor o igual al segundo. Este es el operador utilizado por las expresiones producidas por los métodos **isLessOrEqualTo** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **IS_NULL_OR_EQ**: compara los dos operandos; la comparación se cumple si son iguales o si el primer valor es nulo. Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrEqualTo** de las clases que implementan las interfaces de expresión por tipo.
- **IS_NULL_OR_NEQ**: compara los dos operandos; la comparación se cumple si no son iguales o si el primer valor es nulo. Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrNotEqualTo** de las clases que implementan las interfaces de expresión por tipo.
- **IS_NULL_OR_GT**: compara los dos operandos; la comparación se cumple si el primer valor es mayor que el segundo, o si el primer valor es nulo. Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrGreaterThan** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **IS_NULL_OR_GTEQ**: compara los dos operandos; la comparación se cumple si el primer valor es mayor o igual al segundo, o si el primer valor es nulo. Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrGreaterOrEqualTo** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **IS_NULL_OR_LT**: compara los dos operandos; la comparación se cumple si el primer valor es menor que el segundo, o si el primer valor es nulo. Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrLessThan** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **IS_NULL_OR_LTEQ**: compara los dos operandos; la comparación se cumple si el primer valor es menor o igual al segundo, o si el primer valor es nulo. Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrLessOrEqualTo** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **STARTS_WITH**: compara los dos operandos; la comparación se cumple si el primer valor comienza con una secuencia de caracteres igual al segundo valor. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **startsWith** de las clases que implementan la interfaz CharacterExpression.
- **NOT_STARTS_WITH**: compara los dos operandos; la comparación se cumple si el primer valor no comienza con una secuencia de caracteres igual al segundo valor. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por

las expresiones producidas por los métodos **notStartsWith** de las clases que implementan la interfaz CharacterExpression.

- **CONTAINS**: compara los dos operandos; la comparación se cumple si el primer valor contiene una secuencia de caracteres igual al segundo valor. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **contains** de las clases que implementan la interfaz CharacterExpression.
- **NOT_CONTAINS**: compara los dos operandos; la comparación se cumple si el primer valor no contiene una secuencia de caracteres igual al segundo valor. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **notContains** de las clases que implementan la interfaz CharacterExpression.
- **ENDS_WITH**: compara los dos operandos; la comparación se cumple si el primer valor termina con una secuencia de caracteres igual al segundo valor. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **endsWith** de las clases que implementan la interfaz CharacterExpression.
- **NOT_ENDS_WITH**: compara los dos operandos; la comparación se cumple si el primer valor no termina con una secuencia de caracteres igual al segundo valor. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*)... Este es el operador utilizado por las expresiones producidas por los métodos **notEndsWith** de las clases que implementan la interfaz CharacterExpression.
- **IS_NULL_OR_STARTS_WITH**: compara los dos operandos; la comparación se cumple si el primer valor comienza con una secuencia de caracteres igual al segundo valor, o si el primer valor es nulo. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrStartsWith** de las clases que implementan la interfaz CharacterExpression.
- **IS_NULL_OR_NOT_STARTS_WITH**: compara los dos operandos; la comparación se cumple si el primer valor no comienza con una secuencia de caracteres igual al segundo valor, o si el primer valor es nulo. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrNotStartsWith** de las clases que implementan la interfaz CharacterExpression.
- **IS_NULL_OR_CONTAINS**: compara los dos operandos; la comparación se cumple si el primer valor contiene una secuencia de caracteres igual al segundo valor, o si el primer valor es nulo. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrContains** de las clases que implementan la interfaz CharacterExpression.
- **IS_NULL_OR_NOT_CONTAINS**: compara los dos operandos; la comparación se cumple si el primer valor no contiene con una secuencia de caracteres igual al segundo valor, o si el primer valor es nulo. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrNotContains** de las clases que implementan la interfaz CharacterExpression.
- **IS_NULL_OR_ENDS_WITH**: compara los dos operandos; la comparación se cumple si el primer valor termina con una secuencia de caracteres igual al segundo valor, o si el primer valor es nulo. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrEndsWith** de las clases que implementan la interfaz CharacterExpression.
- **IS_NULL_OR_NOT_ENDS_WITH**: compara los dos operandos; la comparación se cumple si el primer valor no termina con una secuencia de caracteres igual al segundo valor, o si el primer valor es nulo. La comparación hace diferencia entre mayúsculas y minúsculas (es *case-sensitive*). Este es el operador utilizado por las expresiones producidas por los métodos **isNullOrNotEndsWith** de las clases que implementan la interfaz CharacterExpression.

Expresiones Condicionales

Las Expresiones Condicionales son expresiones que se utilizan para definir operaciones condicionales, es decir, operaciones que, mediante un operador condicional, evalúan una expresión booleana y producen uno de dos valores dependiendo de si esa expresión booleana se cumple o no. Las operaciones condicionales son muy similares a las operaciones que se construyen con el operador ternario **"?"** de Java, tales como (**`x?v1:v2`**), donde **`x`** representa una expresión booleana, **`v1`** el resultado que se produce si **`x`** es verdadera, y **`v2`** el resultado que se produce si **`x`** no es verdadera. A diferencia del operador ternario de Java, si el valor de **`x`** es nulo, las operaciones de expresiones condicionales aún producen un valor, el valor **`v2`**, en lugar de arrojar una excepción. En otras palabras, en las operaciones de expresiones condicionales la expresión booleana no es verdadera si es falsa o es nula. Para producir expresiones condicionales, la interfaz BooleanExpression tiene varias firmas del método **then**. El método **then** produce una expresión condicional con una operación de la forma (**`x?v1:null`**), es decir, una operación condicional que produce **`v1`** si la expresión booleana **`x`** es verdadera, o **`null`** en caso contrario. Para completar la expresión condicional, las clases que implementan la interfaz ConditionalX (BooleanConditionalX, CharacterConditionalX, NumericConditionalX, TemporalConditionalX y EntityConditionalX) tienen varias firmas del método **otherwise**. El método **otherwise** toma una expresión condicional producida por el método **then** y la convierte en otra, con una operación de la forma (**`x?v1:v2`**). La enumeración **ConditionalOp** tiene un elemento por cada operador condicional; los operadores utilizados por expresiones condicionales son:

- **IF_THEN**: corresponde a una operación condicional de la forma (**`x?v1:null`**). Este es el operador utilizado por las expresiones producidas por los métodos **then** de las clases que implementan la interfaz BooleanExpression.
- **IF_THEN_ELSE**: corresponde a una operación condicional de la forma (**`x?v1:v2`**). Este es el operador utilizado por las expresiones producidas por los métodos **otherwise** de las clases que implementan la interfaz ConditionalX**.**

Expresiones Escalares

Las expresiones escalares son expresiones que se utilizan para definir operaciones escalares, es decir, operaciones que, mediante un operador escalar y un valor, producen otro valor. Salvo en el caso de los operadores que contemplan el valor nulo, estas operaciones producen un valor nulo si el operando es nulo. Los operadores escalares varían en función del tipo de dato del operando y, por lo tanto, las interfaces de expresión por tipo tienen diferentes métodos para producir expresiones escalares. La enumeración **ScalarOp** tiene un elemento por cada operador escalar; los operadores utilizados por expresiones escalares son:

- **DEFAULT_WHEN_NULL**: compara el valor de una propiedad contra el valor nulo; si son iguales, la operación produce el valor por omisión de la propiedad; en caso contrario, el valor de la propiedad. Este es el operador utilizado por las expresiones producidas por los métodos **defaultWhenNull** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **NULL_WHEN_DEFAULT**: compara el valor de una propiedad contra el valor por omisión de esa misma propiedad; si son iguales, la operación produce un valor nulo; en caso contrario, el valor de la propiedad. Este es el operador utilizado por las expresiones producidas por los métodos **nullWhenDefault** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **NOT**: produce la negación lógica de un valor booleano. Este es el operador utilizado por las expresiones producidas por los métodos **not** de las clases que implementan la interfaz BooleanExpression.
- **LOWER**: convierte un valor alfanumérico a otro que tiene los mismos caracteres pero todos en letra minúscula. Este es el operador utilizado por las expresiones producidas por los métodos **lower** de las clases que implementan la interfaz CharacterExpression.

- **UPPER**: convierte un valor alfanumérico a otro que tiene los mismos caracteres pero todos en letra mayúscula. Este es el operador utilizado por las expresiones producidas por los métodos **upper** de las clases que implementan la interfaz CharacterExpression.
- **UNCAPITALIZE**: convierte un valor alfanumérico a otro que tiene los mismos caracteres pero con el primer carácter en letra minúscula. Este es el operador utilizado por las expresiones producidas por los métodos **uncapitalize** de las clases que implementan la interfaz CharacterExpression.
- **TRIM**: convierte un valor alfanumérico a otro, eliminando todos caracteres de control al comienzo y al final del valor. Este es el operador utilizado por las expresiones producidas por los métodos **trim** de las clases que implementan la interfaz CharacterExpression.
- **LTRIM**: convierte un valor alfanumérico a otro, eliminando todos caracteres de control al comienzo del valor. Este es el operador utilizado por las expresiones producidas por los métodos **ltrim** de las clases que implementan la interfaz CharacterExpression.
- **RTRIM**: convierte un valor alfanumérico a otro, eliminando todos caracteres de control al final del valor. Este es el operador utilizado por las expresiones producidas por los métodos **rtrim** de las clases que implementan la interfaz CharacterExpression.
- **MODULUS**: produce el valor absoluto de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **abs** de las clases que implementan la interfaz NumericExpression.
- **OPPOSITE**: produce el valor opuesto (inverso aditivo) de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **chs** de las clases que implementan la interfaz NumericExpression.
- **RECIPROCAL**: produce el valor recíproco (inverso multiplicativo) de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **inv** de las clases que implementan la interfaz NumericExpression.
- **TO_CHARACTER**: produce el primer carácter de un valor alfanumérico. Este es el operador utilizado por las expresiones producidas por los métodos **toChar** de las clases que implementan la interfaz CharacterExpression.
- **TO_STRING**: produce una cadena de caracteres a partir de un valor alfanumérico. Este es el operador utilizado por las expresiones producidas por los métodos **toCharString** de las clases que implementan las interfaces BooleanExpression, CharacterExpression, NumericExpression y TemporalExpression.
- **TO_BYTE**: produce un valor de tipo Byte a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toByte** de las clases que implementan la interfaz NumericExpression.
- **TO_SHORT**: produce un valor de tipo Short a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toShort** de las clases que implementan la interfaz NumericExpression.
- **TO_INTEGER**: produce un valor de tipo Integer a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toInteger** de las clases que implementan la interfaz NumericExpression.
- **TO_LONG**: produce un valor de tipo Long a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toLong** de las clases que implementan la interfaz NumericExpression.
- **TO_FLOAT**: produce un valor de tipo Float a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toFloat** de las clases que implementan la interfaz NumericExpression.
- **TO_DOUBLE**: produce un valor de tipo Double a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toDouble** de las clases que implementan la interfaz NumericExpression.
- **TO_BIG_INTEGER**: produce un valor de tipo BigInteger a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toBigInteger** de las clases que implementan la interfaz NumericExpression.
- **TO_BIG_DECIMAL**: produce un valor de tipo BigDecimal a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toBigDecimal** de las clases que implementan la interfaz NumericExpression.

- **TO_BYTE**: produce un valor de tipo Byte a partir de un valor numérico. Este es el operador utilizado por las expresiones producidas por los métodos **toByte** de las clases que implementan la interfaz NumericExpression.
- **TO_DATE**: produce un valor de tipo Date a partir de un valor de fecha-y-hora. Este es el operador utilizado por las expresiones producidas por los métodos **toDate** de las clases que implementan la interfaz NumericExpression.
- **TO_TIME**: produce un valor de tipo Time a partir de un valor de fecha-y-hora. Este es el operador utilizado por las expresiones producidas por los métodos **toTime** de las clases que implementan la interfaz NumericExpression.
- **TO_TIMESTAMP**: produce un valor de tipo Timestamp a partir de un valor de fecha-y-hora. Este es el operador utilizado por las expresiones producidas por los métodos **toTimestamp** de las clases que implementan la interfaz NumericExpression.

Expresiones de Pares Ordenados

Las expresiones de pares ordenados son expresiones que se utilizan para definir operaciones sobre pares ordenados, es decir, operaciones que, mediante un operador y dos operandos en un orden determinado (ya que del orden puede depender el resultado), producen un valor. Salvo en el caso de los operadores que contemplan el valor nulo, estas operaciones producen un valor nulo si alguno de los operandos es nulo. Los operadores varían en función del tipo de dato de los operandos y, por lo tanto, las interfaces de expresión por tipo tienen diferentes métodos para producir expresiones de pares ordenados. La enumeración **OrderedPairOp** tiene un elemento por cada operador de este tipo; los operadores utilizados por expresiones de pares ordenados son:

- **COALESCE**: produce el primero de los operandos cuyo valor no sea nulo. Este es el operador utilizado por las expresiones producidas por los métodos **coalesce** de las clases que implementan las interfaces CharacterExpression, NumericExpression, TemporalExpression y EntityExpression.
- **NULLIF**: compara los dos operandos; si son iguales, la operación produce un valor nulo; en caso contrario, el valor del primer operando. Este es el operador utilizado por las expresiones producidas por los métodos **nullIf** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **MAXIMUM**: produce el valor mayor entre los dos operandos. Este es el operador utilizado por las expresiones producidas por los métodos **max** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **MINIMUM**: produce el valor mayor entre los dos operandos. Este es el operador utilizado por las expresiones producidas por los métodos **min** de las clases que implementan las interfaces CharacterExpression, NumericExpression y TemporalExpression.
- **AND**: produce la conjunción de dos valores booleanos. Este es el operador utilizado por las expresiones producidas por los métodos **and** de las clases que implementan la interfaz BooleanExpression.
- **NAND**: produce la negación alternativa de dos valores booleanos. Es equivalente a la negación (NOT) de la conjunción (AND). Este es el operador utilizado por las expresiones producidas por los métodos **nand** de las clases que implementan la interfaz BooleanExpression.
- **OR**: produce la disyunción de dos valores booleanos. Este es el operador utilizado por las expresiones producidas por los métodos **or** de las clases que implementan la interfaz BooleanExpression.
- **NOR**: produce la negación conjunta de dos valores booleanos. Es equivalente a la negación (NOT) de la disyunción (OR). Este es el operador utilizado por las expresiones producidas por los métodos **nor** de las clases que implementan la interfaz BooleanExpression.
- **XOR**: produce la disyunción exclusiva de dos valores booleanos. Este es el operador utilizado por las expresiones producidas por los métodos **xor** de las clases que implementan la interfaz BooleanExpression.
- **XNOR**: produce la equivalencia de dos valores booleanos. Es equivalente a la negación (NOT) de la disyunción exclusiva (XOR). Este es el operador utilizado por las expresiones producidas por los métodos **xnor** de las clases que implementan la interfaz BooleanExpression.

- **X_IMPLIES_Y**: produce la condicional material de dos valores booleanos (resulta en falso solo si el primer operando es verdadero y el segundo es falso). Este es el operador utilizado por las expresiones producidas por los métodos **implies** de las clases que implementan la interfaz BooleanExpression.
- **CONCAT**: produce un valor alfanumérico concatenado los dos operandos. Este es el operador utilizado por las expresiones producidas por los métodos **concat** de las clases que implementan la interfaz CharacterExpression.
- **X_PLUS_Y**: produce el valor de la suma de los dos operandos. Este es el operador utilizado por las expresiones producidas por los métodos **plus** de las clases que implementan la interfaz NumericExpression.
- **X_MINUS_Y**: produce el valor de la diferencia entre los dos operandos, restando el segundo del primero. Este es el operador utilizado por las expresiones producidas por los métodos **minus** de las clases que implementan la interfaz NumericExpression.
- **X_MULTIPLIED_BY_Y**: produce el valor del producto de los dos operandos. Este es el operador utilizado por las expresiones producidas por los métodos **times** de las clases que implementan la interfaz NumericExpression.
- **X_DIVIDED_INTO_Y**: produce el valor de la división de los dos operandos, dividiendo el primero entre el segundo. Este es el operador utilizado por las expresiones producidas por los métodos **over** de las clases que implementan la interfaz NumericExpression.
- **X_RAISED_TO_THE_Y**: produce el valor del primer operando elevado a la potencia determinada por el valor del segundo. Este es el operador utilizado por las expresiones producidas por los métodos **toThe** de las clases que implementan la interfaz NumericExpression.

Definición de atributos de expresiones

Los atributos de una expresión se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** de la meta-expresión se utiliza para establecer la descripción de la expresión que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleExpressions** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la expresión. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la expresión.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-expresión se utiliza para establecer la descripción corta de la expresión que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleExpressions** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la expresión. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la expresión.

Método setDefaultErrorMessage

El método **setDefaultErrorMessage** de la meta-expresión se utiliza para establecer el mensaje de error asociado a la expresión que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultErrorMessage** debe ejecutarse en el método **settleExpressions** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultErrorMessage(String defaultErrorMessage)

Establece la descripción de la expresión. Su único parámetro es:

- **defaultErrorMessage**: mensaje de error asociado a la expresión.

Definición de transiciones

Las transiciones son caminos válidos entre dos estados. Cada una de las transiciones de la entidad se puede definir como un campo de la meta-entidad. A los campos de la meta-entidad que definen las transiciones se les conoce como **meta-transiciones**. Alternativamente, al definir las operaciones de la entidad se pueden especificar las transiciones realizadas por cada operación sin tener que definir meta-transiciones, tal como se describe en la sección Definición de transiciones de operaciones.

Para definir meta-transiciones se utiliza la clase **Transition** o una clase Java que extienda, directa o indirectamente, la clase **Transition**.

Toda meta-transición se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-transiciones.

```
public class Persona extends PersistentEntityBase {

    constructores

    propiedades

    expresiones (restricciones, segmentos y estados)

    Transition soltero_casado, casado_viudo;

    @Override
    protected void settleTransitions() {
        super.settleTransitions();
        soltero_casado.settle(soltero, casado);
        casado_viudo.settle(casado, viudo);
    }

}
```

Definición de atributos de transiciones

Los atributos de una transición se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** de la meta-transición se utiliza para establecer la descripción de la transición que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleTransitions** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la transición. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la transición.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-transición se utiliza para establecer la descripción corta de la transición que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleTransitions** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la transición. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la transición.

Método settle

El método **settle** de la meta-transición se utiliza para especificar el estado inicial y el estado final de la transición.

El método **settle** debe ejecutarse en el método **settleTransitions** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

settle(State x, State y)

Establece el estado inicial y el estado final de la transición. Sus parámetros son:

- **x**: estado inicial.
- **y**: estado final.

Definición de operaciones de negocio

Cada operación de negocio de cada entidad de su aplicación se debe definir mediante una clase Java que extienda, directa o indirectamente, la clase **Operation**. A las clases Java que definen las operaciones de negocio se les conoce como **meta-operaciones**.

Una meta-operación puede ser una clase anidada (*nested class*) en la entidad a la que pertenece o una clase de nivel superior (*top level class*). En cualquiera de los dos casos debe definirse con el modificador de acceso **public**.

Para más información sobre la definición de meta-operaciones, vea Meta-programación de operaciones de negocio en este mismo documento.

Además, en la meta-entidad se debe definir un campo para cada meta-operación de la entidad. Tales campos se deben definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-operaciones.

```
public class Persona extends PersistentEntityBase {

    constructores

    propiedades

    expresiones (restricciones, segmentos y estados)

    RegistrarMatrimonio registrarMatrimonio;

    @OperationClass(access = OperationAccess.RESTRICTED)
    public class RegistrarMatrimonio extends ProcessOperation {
    }

    EnviarFelicitacion enviarFelicitacion;

    @OperationClass(access = OperationAccess.RESTRICTED)
    public class EnviarFelicitacion extends ProcessOperation {
    }

    @Override
    protected void settleOperations() {
        super.settleOperations();
        registrarMatrimonio.addTransition(soltero, casado);
    }

}
```

Definición de transiciones de operaciones

Las transiciones que lleva a cabo de cada operación se establecen mediante el método **addTransition**, tal como se describe a continuación. Alternativamente, se pueden establecer mediante **meta-transiciones**, tal como se indica en la sección Definición de transiciones.

Método addTransition

El método **addTransition** de la meta-operación se utiliza para agregar una transición a la lista de transiciones que lleva a cabo la operación.

El método **addTransition** debe ejecutarse en el método **settleOperations** de la meta-entidad y tiene dos firmas, las cuales se describen a continuación.

addTransition(Transition transition)

Agrega una transición a la lista de transiciones de la operación. Su único parámetro es:

- **transition**: transición de la meta-entidad.

addTransition(State x, State y)

Define una transición y la agrega a la lista de transiciones de la operación. Sus parámetros son:

- **x**: estado inicial.
- **y**: estado final.

Definición de disparadores de operaciones

Los disparadores que ejecutan una operación se pueden establecer mediante el método **addTriggerOn**, tal como se describe a continuación. Alternativamente, se pueden establecer mediante **meta-disparadores**, tal como se indica en la sección Definición de disparadores.

Método addTriggerOn

El método **addTriggerOn** de la meta-operación se utiliza para agregar uno o más estados a la lista de estados en los cuales se debe disparar la operación.

El método **addTriggerOn** debe ejecutarse en el método **settleOperations** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

addTriggerOn(State… states)

Agrega uno o más estados a la lista de estados en los cuales se debe disparar la operación. Su único parámetro es:

- **states**: uno o más estados de la entidad.

Definición de disparadores (triggers)

Los disparadores son artefactos que permiten especificar las acciones que se deben tomar cuando una instancia de una entidad alcanza un estado determinado. Cada uno de los disparadores de la entidad se puede definir como un campo de la meta-entidad. A los campos de la meta-entidad que definen los disparadores se les conoce como **meta-disparadores**. Alternativamente, al definir las operaciones de la entidad se pueden especificar los disparadores que ejecutan cada operación sin tener que definir meta-disparadores, tal como se describe en la sección Definición de disparadores de operaciones.

Para definir meta-disparadores se utiliza la clase **Trigger** o una clase Java que extienda, directa o indirectamente, la clase **Trigger**.

Toda meta-disparador se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-disparadores.

```
public class Persona extends PersistentEntityBase {

    constructores

    propiedades

    expresiones (restricciones, segmentos y estados)

    operaciones

    Trigger casado_enviarFelicitacion;

    @Override
    protected void settleTriggers() {
        super.settleTriggers();
        casado_enviarFelicitacion.settle(casado, enviarFelicitacion);
    }

}
```

Definición de atributos de disparadores

Los atributos de un disparador se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-entidad, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** del meta-disparador se utiliza para establecer la descripción del disparador que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleTriggers** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción del disparador. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen el disparador.

Método setDefaultShortDescription

El método **setDefaultShortDescription** del meta-disparador se utiliza para establecer la descripción corta del disparador que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleTriggers** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta del disparador. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente el disparador.

Método settle

El método **settle** del meta-disparador se utiliza para especificar las acciones que se deben tomar cuando una instancia de una entidad alcanza un estado determinado.

El método **settle** debe ejecutarse en el método **settleTriggers** de la meta-entidad y tiene una sola firma, la cual se describe a continuación.

settle(State state, ProcessOperation operation)

Establece el estado y la operación del disparador. Sus parámetros son:

- **state**: estado de la entidad.
- **operation**: operación de negocio de la entidad.

Meta-programación de operaciones de negocio

Cada operación de negocio de cada entidad de su aplicación se debe definir mediante una clase Java que extienda, directa o indirectamente, la clase **Operation**. A las clases Java que definen las operaciones de negocio se les conoce como **meta-operaciones**.

Una meta-operación puede ser una clase anidada (*nested class*) en la entidad a la que pertenece o una clase de nivel superior (*top level class*). En cualquiera de los dos casos debe definirse con el modificador de acceso **public**.

Las operaciones de negocio para generar archivos se deben definir mediante una clase Java que extienda, directa o indirectamente, la clase **ExportOperation**, la cual a su vez extiende la clase **Operation**.

Las operaciones de negocio para generar informes se deben definir mediante una clase Java que extienda, directa o indirectamente, la clase **ReportOperation**, la cual a su vez extiende la clase **Operation**.

Todas las demás operaciones de negocio se deben definir mediante una clase Java que extienda, directa o indirectamente, la clase **ProcessOperation**, la cual a su vez extiende la clase **Operation**.

Continuando con el ejemplo de la meta-entidad Persona, a continuación se ilustra la definición de meta-operaciones.

```
@ProcessOperationClass(overloading = Kleenean.FALSE)
@OperationClass(access = OperationAccess.RESTRICTED)
public class RegistrarMatrimonio extends ProcessOperation {
}

@ProcessOperationClass(overloading = Kleenean.FALSE)
@OperationClass(access = OperationAccess.RESTRICTED)
public class EnviarFelicitacion extends ProcessOperation {
}

@ReportOperationClass()
@OperationClass(access = OperationAccess.RESTRICTED)
public class PersonasCasadas extends ReportOperation {
}
```

Definición de atributos de operaciones mediante anotaciones

Una parte de los atributos de una operación se establecen decorando la meta-operación con anotaciones, tal como se describe a continuación. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-operación, tal como se describe en la sección Definición de atributos de operaciones mediante métodos de este mismo documento.

Las anotaciones para establecer atributos de meta-operaciones son:

- OperationClass
- ExportOperationClass
- ReportOperationClass
- ProcessOperationClass

Anotación OperationClass

La anotación **OperationClass** se utiliza para establecer atributos básicos de la operación. Es válida para cualquier clase de operación de negocio. Los elementos de la anotación son**:**

- **asynchronous** {UNSPECIFIED, TRUE, FALSE}: indica si la operación se debe ejecutar de manera síncrona o asíncrona. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si se debe ejecutar de manera asíncrona; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **access** {UNSPECIFIED, PRIVATE, PUBLIC, PROTECTED, RESTRICTED}: especifica el tipo de control de acceso de la operación. Su valor es uno de los elementos de la enumeración **OperationAccess**. Seleccione PRIVATE, PUBLIC, PROTECTED o RESTRICTED si la operación es de acceso privado, público, protegido o restringido, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RESTRICTED. Las operaciones con acceso privado no pueden ser ejecutadas directamente por los usuarios del sistema. Son ejecutadas solo por otras operaciones, a través de la Interfaz de Programación (API). Las operaciones con acceso público, protegido y restringido si pueden ser ejecutadas directamente por los usuarios del sistema, a través de la Interfaz de Usuario (UI). Las operaciones con acceso público pueden ser ejecutadas por todos los usuarios del sistema, aun cuando no tengan autorización explícita para ello. Las operaciones con acceso protegido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Al igual que las operaciones con acceso protegido, las operaciones con acceso restringido pueden ser ejecutadas por usuarios designados como súper-usuario o por usuarios explícitamente autorizados. Además, a diferencia de las operaciones con acceso protegido, las operaciones personalizables con acceso restringido, también pueden ser ejecutadas por usuarios que no tengan autorización explícita, pero solo sobre las instancias de la entidad que sean propiedad del usuario.
- **logging** {UNSPECIFIED, SUCCESS, FAILURE, BOTH}: especifica cuando se deben registrar pistas de auditoría de la ejecución de la operación. Su valor es uno de los elementos de la enumeración **OperationLogging**. Seleccione SUCCESS, FAILURE o BOTH si las pistas se deben registrar cuando la operación se ejecute exitosamente, cuando se produzca un error al ejecutar la operación, o en ambos casos, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es SUCCESS.

Anotación ExportOperationClass

La anotación **ExportOperationClass** se utiliza para establecer atributos de meta-operaciones que extienden la clase **ExportOperation** (operaciones de negocio para generar archivos). Los elementos de la anotación son**:**

- **name**: nombre alterno del procedimiento de generación del archivo.
- **view**: nombre de la vista SQL que se utiliza para la generación del archivo.
- **viewField**: nombre de la meta-vista de la entidad que se utiliza para la generación del archivo. Si alguna de las propiedades de detalle de la meta-vista tiene una función de agregación, entonces la operación puede generar el archivo tanto detallado como resumido; en caso contrario, solo puede generar el archivo detallado.
- **type** {UNSPECIFIED, DYNAMIC, PARAMETERIZED}: especifica el tipo de consulta de la operación. Su valor es uno de los elementos de la enumeración **ExportQueryType**. Seleccione DYNAMIC o PARAMETERIZED si la consulta es dinámica o parametrizada, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es DYNAMIC. Las consultas dinámicas son aquellas que dinámicamente construyen la cláusula WHERE de la consulta a partir de los parámetros no nulos de la operación y de sus respectivos operadores escalares de comparación (vea Anotación

ParameterField). Las consultas parametrizadas son consultas estáticas (su cláusula WHERE está prestablecida) que utilizan todos los parámetros (aún los nulos) de la operación.

- **rowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben exportar. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.
- **detailRowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben exportar al generar archivos detallados. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.
- **summaryRowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben exportar al generar archivos resumidos. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.
- **chartRowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben exportar al generar archivos gráficos. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.
- **sortOption** {ASC, DESC}: especifica el criterio de ordenamiento por omisión de las filas exportadas. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para exportar las filas ordenadas por el valor de su clave primaria, de manera ascendente o descendente, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es ASC.

Anotación ReportOperationClass

La anotación **ReportOperationClass** se utiliza para establecer atributos de meta-operaciones que extienden la clase **ReportOperation** (operaciones de negocio para generar informes). Los elementos de la anotación son**:**

- **name**: nombre alterno del procedimiento de generación del informe.
- **view**: nombre de la vista SQL que se utiliza para la generación del informe.
- **viewField**: nombre de la meta-vista de la entidad que se utiliza para la generación del informe. Si alguna de las propiedades de detalle de la meta-vista tiene una función de agregación, entonces la operación puede generar el informe tanto detallado como resumido; si además tiene al menos una propiedad de control y, por ende, al menos un grupo de agregación, entonces la operación también puede generar el informe gráfico; si la meta-vista no tiene agregaciones ni grupos, entonces la operación solo puede generar el informe detallado.
- **type** {UNSPECIFIED, DYNAMIC, PARAMETERIZED}: especifica el tipo de consulta de la operación. Su valor es uno de los elementos de la enumeración **ReportQueryType**. Seleccione DYNAMIC o PARAMETERIZED si la consulta es dinámica o parametrizada, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es DYNAMIC. Las consultas dinámicas son aquellas que dinámicamente construyen la cláusula WHERE de la consulta a partir de los parámetros no nulos de la operación y de sus respectivos operadores escalares de comparación (vea Anotación ParameterField). Las consultas parametrizadas son consultas estáticas (su cláusula WHERE está prestablecida) que utilizan todos los parámetros (aún los nulos) de la operación.
- **rowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben reportar. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.
- **detailRowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben reportar al generar informes detallados. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.
- **summaryRowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben reportar al generar informes resumidos. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.
- **chartRowsLimit** [0:1.000.000]: específica el número de máximo de filas que se deben reportar al generar informes gráficos. Su valor debe ser un número entero entre 0 y 1.000.000. Utilice 0 cuando no exista límite. El valor predeterminado es 10.000.

- **sortOption** {ASC, DESC}: especifica el criterio de ordenamiento por omisión de las filas reportadas. Su valor es uno de los elementos de la enumeración **SortOption**. Seleccione ASC o DESC para exportar las filas ordenadas por el valor de su clave primaria, de manera ascendente o descendente, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es ASC.

Anotación ProcessOperationClass

La anotación **ProcessOperationClass** se utiliza para establecer atributos de meta-operaciones que extienden la clase **ProcessOperation**. Los elementos de la anotación son**:**

- **name**: nombre alterno del procedimiento.
- **overloading** {UNSPECIFIED, TRUE, FALSE}: indica si al generar procedimientos SQL de la operación se puede, o no, sobrecargar (*overload*) el nombre del procedimiento. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si se puede sobrecargar el nombre; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE. Para información sobre el uso del atributo **overloading**, vea Implementación de operaciones de negocio mediante funciones de base de datos.
- **processingGroup**: especifica el código que identifica el grupo de procesamiento al que pertenece la operación. Las aplicaciones generadas con la plataforma **jee1** no permiten la ejecución simultánea de operaciones de un mismo grupo.

Definición de atributos de operaciones mediante métodos

Una parte de los atributos de una operación se establecen decorando la meta-operación con anotaciones, tal como se describe en la sección Definición de atributos de operaciones mediante anotaciones de este mismo documento. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-operación, tal como se describe a continuación.

Los métodos para establecer atributos de meta-operaciones son:

- setDefaultDescription
- setDefaultShortDescription
- setDefaultLabel
- setDefaultShortLabel
- addChartColor
- clearChartColorList

Método setDefaultDescription

El método **setDefaultDescription** de la meta-operación se utiliza para establecer la descripción de la operación que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleAttributes** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la operación. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la operación.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-operación se utiliza para establecer la descripción corta de la operación que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleAttributes** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la operación. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la operación.

Método setDefaultLabel

El método **setDefaultLabel** de la meta-operación se utiliza para establecer la etiqueta de la operación que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultLabel** debe ejecutarse en el método **settleAttributes** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultLabel(String defaultLabel)

Establece la etiqueta de la operación. Su único parámetro es:

- **defaultLabel**: sustantivo singular que se usa como etiqueta de la operación.

Método setDefaultShortLabel

El método **setDefaultShortLabel** de la meta-operación se utiliza para establecer la etiqueta corta de la operación que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultShortLabel** debe ejecutarse en el método **settleAttributes** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultShortLabel(String defaultLabel)

Establece la etiqueta corta del de la operación. Su único parámetro es:

- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta de la operación.

Método addChartColor

El método **addChartColor** de la meta-operación se utiliza para agregar colores a la lista de colores utilizada para generar informes gráficos. Solo aplica a operaciones de la clase **ReportOperation**. Los colores se agregan al final de la lista. Para especificar todos los colores, primero se deben remover los colores almacenados inicialmente en la lista, ejecutando el método **clearChartColorList**.

El método **addChartColor** debe ejecutarse en el método **settleAttributes** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

addChartColor(Color... colors)

Agrega colores a la lista de colores utilizada para generar informes gráficos. Su único parámetro es:

- **colors**: uno o más objetos de la clase **java.awt.Color** (separados por una coma).

Método clearChartColorList

El método **clearChartColorList** de la meta-operación se utiliza para remover todos los colores almacenados en la lista de colores utilizada para generar informes gráficos. Solo aplica a operaciones de la clase **ReportOperation**.

El método **clearChartColorList** debe ejecutarse en el método **settleAttributes** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

clearChartColorList()

Remueve todos los colores almacenados en la lista de colores utilizada para generar informes gráficos. Este método no tiene parámetros.

Definición de parámetros

Cada uno de los parámetros de la operación se debe definir como un campo de la meta-operación. A los campos de la meta-operación que definen parámetros se les conoce como **meta-parámetros**.

Los meta-parámetros se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **Parameter**.

En función del tipo de dato del parámetro que definen, los meta-parámetros se dividen en dos grupos:

- **Primitivos**: si el parámetro es de un tipo primitivo o elemental, es decir, Boolean, Character, String, BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short, Date, Time o Timestamp.
- **Referenciales**: si el parámetro es una referencia a una entidad.

Para definir meta-parámetros primitivos se utilizan las siguientes clases:

- **BigDecimalParameter**: para definir números decimales, con signo y precisión variable.
- **BigIntegerParameter**: para definir números enteros, con signo y precisión variable.
- **BinaryParameter**: para definir “objetos binarios” (imágenes, vídeos, etc.).
- **BooleanParameter**: para definir booleanos (TRUE o FALSE).
- **ByteParameter**: para definir números enteros, con signo, entre -128 y 127.
- **CharacterParameter**: para definir un solo carácter (letra, dígito o carácter especial).
- **DateParameter**: para definir fechas.
- **DoubleParameter**: para definir una representación de coma flotante de números reales (*floating point*) en formato (precisión) doble.
- **FloatParameter**: para definir una representación de coma flotante de números reales (*floating point*) en formato (precisión) simple.
- **IntegerParameter**: para definir números enteros, con signo, entre -2.147.483.648 y 2.147.483.647.
- **LongParameter**: para definir números enteros, con signo, entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.
- **ShortParameter**: para definir números enteros, con signo, entre -32.768 y 32.767.
- **StringParameter**: para definir cadenas de caracteres de longitud variable.
- **TimeParameter**: para definir horas.
- **TimestampParameter**: para definir “sellos de tiempo” (*timestamps*),

Para definir meta-parámetros referenciales se utiliza cualquier meta-entidad. Tanto la clase **AbstractPersistentEntity** como la clase **AbstractPersistentEnumerationEntity**, utilizadas para definir meta-entidades, implementan la interfaz **Parameter**.

Todo meta-parámetro, primitivo o referencial, se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Continuando con el ejemplo de la meta-operación RegistrarMatrimonio de la meta-entidad Persona, a continuación se ilustra la definición de meta-parámetros.

```
@ProcessOperationClass(overloading = Kleenean.FALSE)
@OperationClass(access = OperationAccess.RESTRICTED)
public class RegistrarMatrimonio extends ProcessOperation {

    @InstanceReference
    @Allocation(maxDepth = 1, maxRound = 0)
    protected Persona persona;

    @ParameterField(required = Kleenean.TRUE)
    protected DateParameter fechaMatrimonio;

}
```

Definición de atributos de parámetros mediante anotaciones

Una parte de los atributos de un parámetro se establecen decorando el meta-parámetro con anotaciones, tal como se describe a continuación. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-operación, tal como se describe en la sección Definición de atributos de parámetros mediante métodos de este mismo documento.

Las anotaciones para establecer atributos de meta-parámetros se dividen en los siguientes grupos:

- Anotaciones básicas:
  - ParameterField
  - BigDecimalField
  - BooleanField
  - NumericField
  - StringField
  - TimeField
  - TimestampField
- Anotaciones para establecer el rol del parámetro en la operación:
  - InstanceReference
- Anotaciones para meta-parámetros referenciales:
  - Allocation
  - EntityReferenceSearch
  - Filter
- Otras anotaciones:
  - FileReference

Anotación Allocation

La anotación **Allocation** se utiliza para establecer límites a la instanciación de los meta-parámetros referenciales de la operación. Estos límites se deben ajustar en función de las expresiones programadas en la meta-operación, ya que éstas pueden utilizar propiedades de entidades referenciadas. Si una expresión utiliza una propiedad de una entidad referenciada que está fuera de alcance (*out-of-scope*) entonces, al generar la aplicación, se produce un **NullPointerException** en la instrucción correspondiente a la expresión. La solución a este problema es aumentar los límites establecidos (explícitamente o por omisión) para la instanciación del meta-parámetro referencial que está fuera de alcance.

El primero de los límites determina la profundidad máxima que se puede alcanzar al instanciar el meta-parámetro. Las propiedades de cada meta-parámetro tienen profundidad 1. Las propiedades de las entidades referenciadas en el meta-parámetro tienen profundidad 2. Las propiedades de las entidades referenciadas por las entidades referenciadas en el meta-parámetro tienen profundidad 3, y así sucesivamente, ad-infinitum.

El segundo de los límites determina la cantidad máxima de referencias circulares que se puede alcanzar al instanciar un meta-parámetro. Una referencia circular es una referencia a la misma entidad, hecha de forma directa o indirecta (en otras palabras, hecha a cualquier profundidad).

Los elementos de la anotación son**:**

- **maxDepth**: específica la profundidad máxima del meta-parámetro. Su valor debe ser un número entero mayor o igual a 1. El valor predeterminado es 1.
- **maxRound**: específica la cantidad máxima de referencias circulares del meta-parámetro. Su valor debe ser un número entero mayor o igual a 0. El valor predeterminado es 0.

Anotación BigDecimalField

La anotación **BigDecimalField** se utiliza para establecer atributos de parámetros **BigDecimalParameter**. Los elementos de la anotación son**:**

- **precision** [1:1.000]: específica la precisión o cantidad de dígitos significativos del parámetro. Su valor debe ser un número entero entre 1 y 1.000. El valor predeterminado es 16.
- **scale**: [0:1.000]: específica la escala o cantidad de decimales del parámetro. Su valor debe ser un número entero entre 0 y 1.000 y debe ser menor o igual a **precision**. El valor predeterminado es 0.

Anotación BooleanField

La anotación **BooleanField** se utiliza para establecer atributos de parámetros **BooleanParameter**. Esta anotación tiene un único elemento, el cual se describe a continuación**:**

- **displayType** {UNSPECIFIED, DROPDOWN, CHECKBOX}: indica el tipo de componente que se utiliza para mostrar el valor del parámetro en las vistas (páginas) de ejecución de operaciones de negocio. Su valor es uno de los elementos de la enumeración **BooleanDisplayType**. Seleccione DROPDOWN o CHECKBOX para utilizar una lista desplegable o una casilla de verificación, respectivamente. La opción CHECKBOX solo aplica si el parámetro es requerido (vea el elemento **required** de la Anotación ParameterField). Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es CHECKBOX si el parámetro es requerido; y DROPDOWN si no lo es.

Anotación EntityReferenceSearch

La anotación **EntityReferenceSearch** se utiliza para configurar la forma en que las vistas (páginas) implementan la búsqueda del valor de un meta-parámetro referencial. Los elementos de la anotación son**:**

- **searchType** {UNSPECIFIED, LIST, DISPLAY, NONE}: especifica el tipo de búsqueda. Su valor es uno de los elementos de la enumeración **SearchType**. Seleccione LIST, DISPLAY o NONE para buscar mediante una lista desplegable (*drop-down list*), una vista (página), o para no implementar ningún mecanismo de búsqueda, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es LIST, si la entidad corresponde a una enumeración; y DISPLAY, en los demás casos.
- **listStyle** {UNSPECIFIED, CHARACTER_KEY, NAME, CHARACTER_KEY_AND_NAME, PRIMARY_KEY_AND_CHARACTER_KEY, PRIMARY_KEY_AND_NAME}: especifica el tipo de lista desplegable que se utiliza para la búsqueda. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **searchType** es LIST. Su valor es uno de los elementos de la enumeración **ListStyle**. Seleccione CHARACTER_KEY, NAME, CHARACTER_KEY_AND_NAME, PRIMARY_KEY_AND_CHARACTER_KEY o PRIMARY_KEY_AND_NAME para que la lista desplegable muestre la clave alfanumérica (o de negocio), el nombre, la clave alfanumérica y el nombre, la clave primaria y la clave alfanumérica, o la clave primaria y el nombre, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es CHARACTER_KEY.
- **displayMode** {UNSPECIFIED, READING, WRITING}: especifica el tipo de página que se utiliza para la búsqueda. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **searchType** es DISPLAY. Su valor es uno de los elementos de la enumeración **DisplayMode**. Seleccione READING o WRITING para utilizar una vista (página) de solo consulta o una vista de registro, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es READING.

Anotación FileReference

La anotación **FileReference** se utiliza para designar parámetros **StringParameter** como referencias a archivos cargados en el servidor. Los elementos de la anotación son**:**

- **max**: específica el tamaño máximo (en bytes). Su valor debe ser un número entero, mayor o igual a 0. Utilice 0 para permitir la carga de archivos de cualquier tamaño. Alternativamente, omita el elemento para utilizar el valor predeterminado del atributo. El valor predeterminado es 1.000.000 (1 MB).
- **types** {APPLICATION, AUDIO, DRAWING, IMAGE, MUSIC, TEXT, VIDEO}: lista de extensiones MIME (*Multipurpose Internet Mail Extensions*) válidas. Su valor es una lista de elementos de la enumeración **MimeType**. Alternativamente, omita el elemento permitir la carga de archivos de cualquier tipo.
- **storage** {UNSPECIFIED, FILE, ROW, ROW_AND_FILE}: especifica el tipo de almacenamiento de los archivos. Su valor es uno de los elementos de la enumeración **UploadStorageOption**. Seleccione FILE, ROW o ROW_AND_FILE para almacenar el archivo en el servidor de aplicaciones (web), en la base de datos, o en ambos, respectivamente. Alternativamente, omita el elemento para utilizar el valor predeterminado del atributo. El valor predeterminado es ROW_AND_FILE.
- **joinField**: nombre de la propiedad (de la entidad a la que pertenece la operación) que hace referencia a la tabla de la base de datos donde se almacena el archivo. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **storage** es ROW o ROW_AND_FILE.

Anotación Filter

La anotación **Filter** se utiliza para especificar los filtros automáticos que las vistas (páginas) implementan en la búsqueda del valor de un meta-parámetro referencial. Los elementos de la anotación son**:**

- **inactive** {UNSPECIFIED, TRUE, FALSE}: indica si se deben filtrar (ignorar), o no, las instancias de la entidad referenciada que se encuentran inactivas (eliminadas lógicamente). Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para filtrar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **owner** {UNSPECIFIED, TRUE, FALSE}: indica si se deben filtrar (ignorar), o no, las instancias de la entidad referenciada que son propiedad de un usuario diferente al que realiza la búsqueda (vea Anotación OwnerProperty). Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para filtrar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **segment** {UNSPECIFIED, TRUE, FALSE}: indica si se deben filtrar (ignorar), o no, las instancias de la entidad referenciada que no pertenecen a uno de los segmentos autorizados al usuario que realiza la búsqueda (vea Anotación SegmentProperty). Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para filtrar; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.

Anotación InstanceReference

La anotación **InstanceReference** se utiliza para designar un meta-parámetro como referencia a la instancia de la entidad de la operación. Por lo tanto, solo se puede designar como referencia a la instancia a meta-parámetros de la clase de la meta-entidad a la que corresponde la operación. Cada operación puede tener una o referencia a la instancia. Si no tiene referencia a la instancia, la operación es una operación de clase, es decir, una operación que, al ejecutarse, afecta un número indeterminado de instancias de la clase. Esta anotación no tiene elementos.

Anotación NumericField

La anotación **NumericField** se utiliza para establecer atributos de parámetros numéricos. Los elementos de la anotación son**:**

- **divisor** [1:N]: específica el divisor para la regla **divisorRule**. Su valor debe ser un número entero entre 1 y otro número que depende de la clase de parámetro: 100, para **ByteParameter**; 10.000, para **ShortParameter**; y 1.000.000, para las demás clases. El valor predeterminado es 100.
- **divisorRule** {UNSPECIFIED, CHECK, CEILING, FLOOR, ROUND}: especifica la regla que debe aplicar al ejecutar la operación. Su valor es uno de los elementos de la enumeración **DivisorRule**. Seleccione CHECK para comprobar que el valor del parámetro sea múltiplo de **divisor**, Seleccione CEILING para ajustar el valor de la propiedad al menor múltiplo de **divisor** que sea mayor o igual al valor suministrado. Seleccione FLOOR para ajustar el valor del parámetro al mayor múltiplo de **divisor** que sea menor o igual al valor suministrado. Seleccione ROUND para ajustar el valor del parámetro al múltiplo de **divisor** más cercano al valor suministrado. Alternativamente, omita el elemento o seleccione UNSPECIFIED para no ejecutar acción alguna.

Anotación ParameterField

La anotación **ParameterField** se utiliza para establecer atributos básicos del parámetro. Los elementos de la anotación son**:**

- **auditable** {UNSPECIFIED, TRUE, FALSE}: indica si el parámetro se debe incluir, o no, en las pistas de auditoría de la operación. Su valor es uno de los elementos de la enumeración **Kleenean**.

Seleccione TRUE para incluir el parámetro; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE para parámetros que corresponden a "objetos binarios" o a contraseñas; y TRUE para las demás parámetros.

- **password** {UNSPECIFIED, TRUE, FALSE}: indica si el parámetro es, o no, una contraseña. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si el parámetro es una contraseña; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **hidden** {UNSPECIFIED, TRUE, FALSE}: indica si el parámetro permanece, o no, oculto en las vistas (páginas) de ejecución de operaciones de negocio. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si el parámetro permanece oculta; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **required** {UNSPECIFIED, TRUE, FALSE}: indica si el parámetro es, o no, obligatoriamente requerido por las vistas (páginas) de ejecución de operaciones de negocio. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si el parámetro es obligatoriamente requerido; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE. Este elemento es irrelevante cuando el parámetro es la referencia a la instancia de la entidad de la operación (vea Anotación InstanceReference); en este caso, el parámetro siempre es requerido.
- **submit** {UNSPECIFIED, TRUE, FALSE}: indica si parámetro es un disparador, es decir, si las vistas (páginas) de ejecución de operaciones de negocio envían la información al servidor de aplicaciones inmediatamente que el valor de este parámetro es modificado. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si el parámetro es un disparador; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **linkedField**: nombre de la meta-propiedad que corresponde a este parámetro. Este elemento es relevante solo para parámetros de operaciones para generar archivos e informes con consulta dinámica (vea Anotación ExportOperationClass y Anotación ReportOperationClass). El nombre SQL de esta propiedad (vea Método setSqlName) es utilizado como nombre de columna para agregar la correspondiente comparación a la cláusula WHERE de la operación, a menos que también se especifique el elemento **linkedColumn** de esta misma anotación.
- **linkedColumn**: nombre de la columna de la tabla que corresponde a este parámetro. Este elemento es relevante solo para parámetros de operaciones para generar archivos e informes con consulta dinámica (vea Anotación ExportOperationClass y Anotación ReportOperationClass). Este nombre es utilizado para agregar la correspondiente comparación a la cláusula WHERE de la operación.
- **operator** {EQ, NEQ, GT, GTEQ, LT, LTEQ, LIKE, NOT_LIKE, IS_NULL_OR_EQ, IS_NULL_OR_NEQ, IS_NULL_OR_GT, IS_NULL_OR_GTEQ, IS_NULL_OR_LT, IS_NULL_OR_LTEQ, IS_NULL_OR_LIKE, IS_NULL_OR_NOT_LIKE}: especifica el operador escalar de comparación a utilizar. Su valor es uno de los elementos de la enumeración **StandardRelationalOp**. El apéndice 2 muestra la comparación correspondiente a cada elemento de la enumeración. El valor predeterminado es EQ. Este elemento es relevante solo para parámetros de operaciones para generar archivos e informes con consulta dinámica (vea Anotación ExportOperationClass y Anotación ReportOperationClass).
- **sequence** [**0:2.147.483.647**]: específica el número de secuencia o posición relativa en la que se muestra el parámetro en las vistas (páginas) de ejecución de operaciones de negocio. Su valor debe ser un número entero entre 0 y 2.147.483.647. Alternativamente, omita el elemento para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es 0. Si todas los parámetros tienen el mismo número de secuencia (0 o cualquier otro), entonces las vistas las muestran en el orden en el mismo orden en el que las meta-parámetros están definidos en la meta-operación.

Anotación StringField

La anotación **StringField** se utiliza para establecer atributos de parámetros **StringParameter**. Los elementos de la anotación son**:**

- **minLength** [0:8.000]: específica la cantidad mínima de caracteres que deben tener los valores del parámetro. Su valor debe ser un número entero entre 0 y 8.000. El valor predeterminado es 0.
- **maxLength**: [1:8.000]: específica la cantidad máxima de caracteres que pueden tener los valores del parámetro. Su valor debe ser un número entero entre 0 y 8.000. El valor predeterminado es 8.000.
- **regex**: especifica la expresión regular que deben satisfacer los valores del parámetro. Para más información sobre expresiones regulares consulte la documentación de Java (la página *Regular Expressions* es un buen punto de partida).
- **letterCase** {UNSPECIFIED, LOWER, UPPER, CAPITALIZED}: especifica la conversión que se debe realizar al almacenar valores del parámetro en la base de datos. Su valor es uno de los elementos de la enumeración **LetterCase**. Seleccione LOWER, UPPER o CAPITALIZED para convertir todos los caracteres a minúsculas, todos a mayúsculas, o para capitalizar (convertir el primer carácter de cada palabra a mayúscula y el resto a minúsculas), respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para no ejecutar conversión alguna.
- **allowDiacritics** {UNSPECIFIED, TRUE, FALSE}: indica si se permiten, o no, signos diacríticos al almacenar valores del parámetro en la base de datos. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para permitir signos diacríticos; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es TRUE.

Anotación TimeField

La anotación **TimeField** se utiliza para establecer atributos de parámetros **TimeParameter**. Esta anotación tiene un único elemento, el cual se describe a continuación:

- **precision** [0:6]: específica la precisión o cantidad de decimales (en los segundos) del parámetro. Su valor debe ser un número entero entre 0 y 6. El valor predeterminado dependerá de cada plataforma.

Anotación TimestampField

La anotación **TimestampField** se utiliza para establecer atributos de parámetros **TimestampParameter**. Esta anotación tiene un único elemento, el cual se describe a continuación:

- **precision** [0:6]: específica la precisión o cantidad de decimales (en los segundos) del parámetro. Su valor debe ser un número entero entre 0 y 6. El valor predeterminado dependerá de cada plataforma.

Definición de atributos de parámetros mediante métodos

Una parte de los atributos de un parámetro se establecen decorando la meta-parámetro con anotaciones, tal como se describe en la sección Definición de atributos de parámetros mediante anotaciones de este mismo documento. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-operación, tal como se describe a continuación.

Los métodos para establecer atributos de meta-parámetros se dividen en los siguientes grupos:

- Métodos relacionados con la interfaz de la aplicación:
  - setDefaultLabel
  - setDefaultShortLabel

- setDefaultDescription
- setDefaultShortDescription
- setDefaultTooltip

- Métodos para establecer filtros:
  - setSearchQueryFilter

- Métodos para establecer valores:
  - setDefaultValue
  - setInitialValue
  - setMaxValue
  - setMinValue

Método setDefaultDescription

El método **setDefaultDescription** de la meta-parámetro se utiliza para establecer la descripción del parámetro que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción del parámetro. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen el parámetro.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-parámetro se utiliza para establecer la descripción corta del parámetro que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta del parámetro. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente el parámetro.

Método setDefaultLabel

El método **setDefaultLabel** de la meta-parámetro se utiliza para establecer la etiqueta del parámetro que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultLabel** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultLabel(String defaultLabel)

Establece la etiqueta del parámetro. Su único parámetro es:

- **defaultLabel**: sustantivo singular que se usa como etiqueta del parámetro.

Método setDefaultShortLabel

El método **setDefaultShortLabel** de la meta-parámetro se utiliza para establecer la etiqueta corta del parámetro que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultShortLabel** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultShortLabel(String defaultLabel)

Establece la etiqueta corta del parámetro. Su único parámetro es:

- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta del parámetro.

Método setDefaultTooltip

El método **setDefaultTooltip** de la meta-parámetro se utiliza para establecer la descripción emergente (*tooltip*) del parámetro que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultTooltip** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultTooltip(String defaultTooltip)

Establece la descripción emergente del parámetro. Su único parámetro es:

- **defaultTooltip**: una o más oraciones que describen muy brevemente el parámetro.

Método setDefaultValue

El método **setDefaultValue** de la meta-parámetro se utiliza para establecer el valor por omisión del parámetro.

El método **setDefaultValue** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene varias firmas para cada clase de meta-parámetro (una para cada tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase del meta-parámetro	Tipo de dato del parámetro value (<T>)
BooleanParameter	java.lang.Boolean SpecialBooleanValue BooleanExpression
CharacterParameter StringParameter	java.lang.String SpecialCharacterValue CharacterExpression

Clase del meta-parámetro	Tipo de dato del parámetro value (<T>)
BigDecimalParameter BigIntegerParameter ByteParameter DoubleParameter FloatParameter IntegerParameter LongParameter ShortParameter	java.lang.Number SpecialNumericValue NumericExpression
DateParameter TimeParameter TimestampParameter	java.util.Date SpecialTemporalValue TemporalExpression
Entity (meta-entidad)	Entity SpecialEntityValue EntityExpression Instance

setDefaultValue(<T> value)

Establece el valor por omisión del parámetro. Su único parámetro es:

- **value**: valor por omisión.

Método setInitialValue

El método **setInitialValue** de la meta-parámetro se utiliza para establecer el valor inicial del parámetro. El valor inicial se utiliza para inicializar el valor del parámetro en las vistas (páginas) de ejecución de operaciones de negocio.

El método **setInitialValue** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene varias firmas para cada clase de meta-parámetro (una para cada tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase del meta-parámetro	Tipo de dato del parámetro value (<T>)
BooleanParameter	java.lang.Boolean SpecialBooleanValue
CharacterParameter StringParameter	java.lang.String SpecialCharacterValue
BigDecimalParameter BigIntegerParameter ByteParameter DoubleParameter FloatParameter IntegerParameter LongParameter ShortParameter	java.lang.Number SpecialNumericValue
DateParameter TimeParameter TimestampParameter	java.util.Date SpecialTemporalValue
Entity (meta-entidad)	Instance

El método tiene otras firmas, pero éstas no deben ser utilizadas. Si se utilizan, se produce un error al generar la aplicación. La siguiente tabla muestra las firmas de uso restringido.

Clase de la meta- parámetro	Tipo de dato del parámetro value (<T>)
BooleanProperty	BooleanExpression
CharacterProperty StringProperty	CharacterExpression
BigDecimalProperty BigIntegerProperty ByteProperty DoubleProperty FloatProperty IntegerProperty LongProperty ShortProperty	NumericExpression
DateProperty TimeProperty TimestampProperty	TemporalExpression
Entity (meta-entidad)	Entity SpecialEntityValue EntityExpression

setInitialValue(<T> value)

Establece el valor inicial del parámetro. Su único parámetro es:

- **value**: valor inicial.

Método setMaxValue

El método **setMaxValue** de la meta-parámetro se utiliza para establecer el valor máximo de un parámetro numérico.

El método **setMaxValue** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene varias firmas para cada clase de meta-parámetro (una para cada tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase de la meta-parámetro	Tipo de dato del parámetro value (<T>)
BigDecimalParameter BigIntegerParameter ByteParameter DoubleParameter FloatParameter IntegerParameter LongParameter ShortParameter	Number NumericExpression

setMaxValue(<T> value)

Establece el valor máximo del parámetro. Su único parámetro es:

- **value**: valor máximo.

Método setMinValue

El método **setMinValue** de la meta-parámetro se utiliza para establecer el valor mínimo de un parámetro numérico.

El método **setMinValue** debe ejecutarse en el método **settleParameters** de la meta-operación y tiene varias firmas para cada clase de meta-parámetro (una para cada tipo de dato del parámetro **value**), como se muestra en la siguiente tabla.

Clase de la meta-parámetro	Tipo de dato del parámetro value (<T>)
BigDecimalParameter BigIntegerParameter ByteParameter DoubleParameter FloatParameter IntegerParameter LongParameter ShortParameter	Number NumericExpression

setMinValue(<T> value)

Establece el valor mínimo del parámetro. Su único parámetro es:

- **value**: valor mínimo.

Método setRequiringFilter

El método **setRequiringFilter** del meta-parámetro se utiliza para establecer el filtro de obligatoriedad del parámetro en las vistas (páginas) de ejecución de operaciones de negocio.

El método **setRequiringFilter** debe ejecutarse en el método **settleFilters** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setRequiringFilter(BooleanExpression filter)

Establece el filtro de obligatoriedad del parámetro. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-operación, vea la sección Definición de expresiones de este mismo documento.

Método setSearchQueryFilter

El método **setSearchQueryFilter** del meta-parámetro se utiliza para establecer el filtro de búsqueda del valor de la referencia (parámetro que hace referencia a una entidad) en las vistas (páginas) de ejecución de operaciones de negocio.

El método **setSearchQueryFilter** debe ejecutarse en el método **settleFilters** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setSearchQueryFilter(BooleanExpression filter)

Establece el filtro de búsqueda del parámetro. Su único parámetro es:

- **filter**: expresión booleana que se utiliza como filtro. Para más información sobre la definición de expresiones en la meta-operación, vea la sección Definición de expresiones de este mismo documento.

Definición de expresiones

Una expresión es una construcción compuesta de variables, operadores y llamadas a métodos, que se construye de acuerdo a la sintaxis del lenguaje Java y que, al evaluarla, produce un solo valor.

Cada una de las expresiones de la operación se puede definir como un campo de la meta-operación. A los campos de la meta-operación que definen expresiones se les conoce como **meta-expresiones**.

Las meta-expresiones se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **Expression**.

La expresión de la meta-expresión, y el valor de sus atributos, se debe establecer en el método **settleExpressiones** de la meta-operación. No todas las expresiones deben ser definidas como meta-expresiones; si la expresión solo es utilizada como parámetro de un método entonces no es necesario definirla como meta-expresión.

El tipo de dato de una expresión es el tipo de dato del valor que produce. En función del tipo de dato de la expresión que definen, las meta-expresiones se dividen en dos grupos:

- **Primitivas**: si la expresión es de un tipo primitivo o elemental, es decir, Boolean, Character, String, BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short, Date, Time o Timestamp.
- **Referencias**: si la expresión es una referencia a una operación. Las meta-expresiones de este grupo implementan la interfaz **EntityExpression**, la cual extiende la interfaz **Expression**.

A su vez, las meta-expresiones primitivas se dividen en:

- **Booleanas**: si la expresión es de tipo Boolean. Las meta-expresiones booleanas se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **BooleanExpression** (la cual extiende la interfaz **Expression**) o la sub-interfaz **Check**.
- **Alfanuméricas**: si la expresión es de tipo Character o String. Las meta-expresiones alfanuméricas se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **CharacterExpression**, la cual extiende la interfaz **Expression**.
- **Numéricas**: si la expresión es de tipo BigDecimal, BigInteger, Byte, Double, Float, Integer, Long o Short. Las meta-expresiones numéricas se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **NumericExpression**, la cual extiende la interfaz **Expression**.
- **Temporales**: si la expresión es de tipo Date, Time o Timestamp. Las meta-expresiones temporales se deben definir mediante una clase Java que implemente, directa o indirectamente, la interfaz **TemporalExpression**, la cual extiende la interfaz **Expression**.

A las interfaces BooleanExpression, CharacterExpression, NumericExpression, TemporalExpression y EntityExpression se les conoce colectivamente como **interfaces de expresión por tipo**.

Toda meta-expresión, primitiva o referencia, se debe definir sin modificador de acceso (*access modifier*), con el modificador **public** o con el modificador **protected**.

Las meta-expresiones proveen un mecanismo sofisticado para definir valores y reglas de negocio. Específicamente, las expresiones se utilizan en métodos para definir valores de parámetros, filtros y límites:

- Métodos para definir valores de parámetros:
  - setDefaultValue
  - setInitialValue

- Métodos para definir filtros de parámetros:
  - setRequiringFilter
  - setSearchQueryFilter

- Métodos para definir límites de parámetros:
  - setMaxValue
  - setMinValue

Además, las meta-expresiones booleanas se utilizan para definir restricciones de la operación. Las restricciones son expresiones que deben ser verdaderas en todas las ejecuciones de la operación; por lo tanto usualmente se utilizan para generar validaciones en las vistas (páginas) de ejecución de operaciones de negocio.

Construcción de expresiones

La construcción de expresiones de meta-operaciones cumple las mismas reglas que la construcción de expresiones de meta-entidades (vea Construcción de expresiones en el capítulo Meta-programación de entidades).

Definición de atributos de expresiones

Los atributos de una expresión se establecen incluyendo el *setter* correspondiente en el método apropiado de la meta-operación, tal como se describe a continuación.

Método setDefaultDescription

El método **setDefaultDescription** de la meta-expresión se utiliza para establecer la descripción de la expresión que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleExpressions** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción de la expresión. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen la expresión.

Método setDefaultShortDescription

El método **setDefaultShortDescription** de la meta-expresión se utiliza para establecer la descripción corta de la expresión que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleExpressions** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta de la expresión. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente la expresión.

Método setDefaultErrorMessage

El método **setDefaultErrorMessage** de la meta-expresión se utiliza para establecer el mensaje de error asociado a la expresión que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultErrorMessage** debe ejecutarse en el método **settleExpressions** de la meta-operación y tiene una sola firma, la cual se describe a continuación.

setDefaultErrorMessage(String defaultErrorMessage)

Establece la descripción de la expresión. Su único parámetro es:

- **defaultErrorMessage**: mensaje de error asociado a la expresión.

Meta-programación de proyectos

Cada proyecto de su aplicación se debe definir mediante una clase Java que extienda, directa o indirectamente, la clase **Project**. A las clases Java que definen los proyectos de su aplicación se les conoce como **meta-proyectos**.

Los meta-proyectos se dividen en:

- **Módulos**: proyectos que sirven para agrupar entidades funcionalmente relacionadas.
- **Maestros**: proyectos que sirven para generar la aplicación. Si utiliza la plataforma **jee1**, se debe definir extendiendo **ProyectoBase**, que a su vez extiende **Project**.

Definición de módulos

Un módulo es un meta-proyecto que sirve para agrupar entidades funcionalmente relacionadas.

En los módulos se debe definir un campo por cada entidad que integra el módulo, como se muestra en el siguiente ejemplo.

```
public class Modulo1 extends Project {

    @Override
    protected void settleAttributes() {
        super.settleAttributes();
        setDefaultLabel("Personal");
        setDefaultDescription("Gestión de Personal");
    }

    Persona persona;
    Jurisdiccion jurisdiccion;

}
```

Un módulo también puede estar integrado otros módulos; se debe definir un campo para cada sub-módulo que integra el módulo.

Definición de proyectos maestros

Un proyecto maestro es un meta-proyecto que sirve para generar la aplicación.

En los proyectos maestros se debe definir un campo por cada módulo que integra la aplicación. Si utiliza la plataforma **jee1**, puede incluir los módulos predefinidos de esta plataforma. Los módulos predefinidos de **jee1** son:

- **Auditoria**: módulo de auditoría; este módulo mantiene un registro de la ejecución de todas las funciones, procesos e informes de la aplicación y de todos los archivos cargados (*uploaded*) al servidor.
- **ControlAcceso**: módulo de control de acceso; este módulo está basado en el modelo de control de acceso por roles, donde se asignan funciones (operaciones) y usuarios a roles, y los permisos de un usuario son la suma de los permisos de los roles a los que se ha asignado el usuario.
- **ControlProcesos**: módulo de control de procesos; este módulo permite consultar los grupos de procesamiento y cancelar procesos que se encuentran en ejecución.

- **ControlPruebas**: módulo de control de pruebas; este módulo permite registrar casos, escenarios, programas y paquetes (suites) de prueba, y llevar control sobre la ejecución de las pruebas y sus resultados.
- **ControlTareas**: módulo de control de tareas; este módulo permite consultar y llevar el control de las tareas que cada usuario debe realizar .

Para generar la aplicación, el proyecto maestro debe incluir un método **main** (público, estático, de tipo void y con un arreglo de tipo String como único parámetro), en el cual se ejecuten los métodos **build** y **generate**, como se muestra en el siguiente ejemplo.

```
public class Maestro extends ProyectoBase {
    public static void main(String[] args) {
        Maestro.setLocale(Locale.forLanguageTag("es"));
        Maestro.setAlertLoggingLevel(LoggingLevel.OFF);
        Maestro maestro = new Maestro();
        if (maestro.build()) {
            maestro.putEnvironmentVariable(VERSION_JAVA, "1.8.0_25");
            maestro.putEnvironmentVariable(VERSION_GLASSFISH, "4.1");
            maestro.putEnvironmentVariable(VERSION_POSTGRESQL, "9.3");
            maestro.setSecurityRealmType(SecurityRealmType.LDAP);
            maestro.setRoleBasedAccessControllerName("LDAP");
            maestro.setInternetAccessAllowed(true);
            maestro.attachAddAttributesMethods(
                meta.postgresql.PostgreSQLAttributes.class);
            maestro.setAlias("xyz1ap101");
            maestro.generate(PLATAFORMA_NETBEANS_POSTGRESQL_GLASSFISH);
            maestro.clearAddAttributesMethods();
        }
    }
    @Override
    protected void settleAttributes() {
        super.settleAttributes();
        setDefaultLabel("Gestión de Recursos Empresariales");
        setDefaultDescription("Gestión de Recursos Empresariales");
    }
    @ProjectModule(menu = Kleenean.TRUE, role = Kleenean.TRUE)
    Modulo1 mod1;
    @ProjectModule(menu = Kleenean.TRUE, role = Kleenean.TRUE)
    Modulo2 mod2;
    @ProjectModule(menu = Kleenean.TRUE, role = Kleenean.TRUE)
    Auditoria modx1;
    @ProjectModule(menu = Kleenean.TRUE, role = Kleenean.TRUE)
    ControlAcceso modx2;
    @ProjectModule(menu = Kleenean.TRUE, role = Kleenean.TRUE)
    ControlProcesos modx3;
    @ProjectModule(menu = Kleenean.TRUE, role = Kleenean.TRUE)
    ControlPruebas modx4;
    @ProjectModule(menu = Kleenean.FALSE, role = Kleenean.TRUE)
    ControlTareas modx5;
}
```

El método **build** analiza los componentes del proyecto y de sus módulos, y construye el PIM (*Platform Independent Model*) de la aplicación. El método **generate**, que recibe como parámetro el nombre de la plataforma para la cual generar la aplicación, transforma el modelo PIM, producido por el método **build**, en PSM (*Platform Specific Model*) y finalmente genera los archivos de la aplicación.

Si utiliza la plataforma **jee1**, el alias del proyecto maestro es el nombre del directorio raíz de los archivos generados; por lo tanto, se debe establecer un alias diferente antes de cada ejecución del método **generate**, utilizando el método setAlias.

La plataforma **jee1** ofrece cuatro variantes. Sus respectivos nombres indican sus principales componentes. Las variantes de **jee1** son:

- PLATAFORMA_NETBEANS_POSTGRESQL_GLASSFISH
- PLATAFORMA_NETBEANS_POSTGRESQL_JBOSS
- PLATAFORMA_ECLIPSE_POSTGRESQL_GLASSFISH
- PLATAFORMA_ECLIPSE_POSTGRESQL_JBOSS

Existe una gran cantidad de archivos comunes a estas variantes; esos archivos se producen generando la seudo-plataforma PLATAFORMA_BASE antes de generar cualquiera de las variantes.

El siguiente ejemplo muestra el método **main** de un proyecto maestro que genera la aplicación para cada una de las variantes de la plataforma **jee1**.

```
public static void main(String[] args) {
    Maestro maestro = new Maestro();
    if (maestro.build()) {
        maestro.setAlias("xyz1ap101");
        maestro.generate(PLATAFORMA_NETBEANS_POSTGRESQL_GLASSFISH);
        maestro.setAlias("xyz1ap102");
        maestro.generate(PLATAFORMA_NETBEANS_POSTGRESQL_JBOSS);
        maestro.setAlias("xyz1ap201");
        maestro.generate(PLATAFORMA_ECLIPSE_POSTGRESQL_GLASSFISH);
        maestro.setAlias("xyz1ap202");
        maestro.generate(PLATAFORMA_ECLIPSE_POSTGRESQL_JBOSS);
    }
}
```

Definición de atributos de proyectos mediante anotaciones

Una parte de los atributos de un proyecto se establecen decorando el meta-proyecto con anotaciones, tal como se describe a continuación. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado del meta-proyecto, tal como se describe en la sección Definición de atributos de proyectos mediante métodos de este mismo documento.

Anotación ProjectModule

La anotación **ProjectModule** se utiliza para establecer atributos de módulos en un proyecto maestro. Los elementos de la anotación son**:**

- **menu** {UNSPECIFIED, TRUE, FALSE}: indica si las vistas (páginas) generadas para las entidades que integran el módulo deben ser, o no, accesibles desde el menú principal de la aplicación. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE si las vistas (páginas) deben ser accesibles desde el menú; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.
- **role** {UNSPECIFIED, TRUE, FALSE}: indica si se deben generar, o no, roles específicos para las operaciones de las entidades que integran el módulo. Su valor es uno de los elementos de la enumeración **Kleenean**. Seleccione TRUE para generar roles específicos; en caso contrario, seleccione FALSE. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es FALSE.

Definición de atributos de proyectos mediante métodos

Una parte de los atributos de un proyecto se establecen decorando el meta-proyecto con anotaciones, tal como se describe en la sección Definición de atributos de proyectos mediante anotaciones de este mismo documento. El resto de los atributos se establecen incluyendo el *setter* correspondiente en el método apropiado del meta-proyecto, tal como se describe a continuación.

Los métodos para establecer atributos de meta-proyectos son:

- attachAddAttributesMethods
- clearAddAttributesMethods
- putEnvironmentVariable
- setAlertLoggingLevel
- setAlias
- setDefaultDescription
- setDefaultShortDescription
- setDefaultLabel
- setDefaultShortLabel
- setDetailLoggingLevel
- setInternetAccessAllowed
- setLocale
- setRoleBasedAccessControllerName
- setSecurityRealmType
- setTrackingLoggingLevel
- setUserEntityClass

Método attachAddAttributesMethods

El método **attachAddAttributesMethods** del meta-proyecto se utiliza para agregar una clase a la lista de clases que contienen métodos **AddAttributes**. El método puede utilizarse repetidamente para agregar varias clases. Para posteriormente borrar la lista se utiliza el método clearAddAttributesMethods.

Todo método **AddAttributes** debe ser público, estático y sin valor de retorno; con un único parámetro que implemente, directa o indirectamente, la interfaz **Artifact**; y decorado con la anotación **AddAttributesMethod**. El nombre del método puede ser cualquiera que cumpla con las reglas de Java. Por ejemplo:

```
@AddAttributesMethod(110)
public static void addAttributes(PersistentEntity entity) {
    entity.addAttribute("TABLESPACE", "TS01");
    entity.addAttribute("WITH",
        KVP.join("fillfactor", 80),
        KVP.join("autovacuum_enabled", true)
    );
}
@AddAttributesMethod(120)
public static void addAttributes(TareaUsuario entity) {
    entity.addAttribute("TABLESPACE", "TS02");
}
@AddAttributesMethod
public static void addAttributes(Key key) {
    key.addAttribute("WITH", KVP.join("fillfactor", 85));    }
```

Típicamente, un método **AddAttributes** agrega uno o más **atributos extraordinarios**, es decir, atributos propios del PSM (*Platform Specific Model*), al artefacto que recibe como parámetro, y/o a sus artefactos relacionados, ejecutando repetidamente el método AddAttribute, el cual se describe más adelante.

Al generar el proyecto, los métodos **AddAttributes** de las clases agregadas son ejecutados automáticamente, para cada uno de los artefactos del proyecto que sean asignables a partir de la clase o interfaz del parámetro. En el ejemplo anterior, el primer método se ejecutará para cada una de las entidades persistentes del proyecto; el segundo, para la entidad TareaUsuario y las entidades que extienden TareaUsuario; y el tercero, para cada una de las claves de acceso (de cada una de las entidades del proyecto). El orden de ejecución de los métodos está determinado, en primer lugar, por el orden en el que se agregan las clases a la lista y, en segundo lugar, por el valor especificado en la anotación **AddAttributesMethod**. Si el orden de ejecución dentro de la clase no es relevante, tal valor puede ser omitido. En el ejemplo anterior, el orden es relevante para los dos primeros métodos porque TareaUsuario implementa PersistentEntity y, por lo tanto, para la entidad TareaUsuario y las entidades que extienden TareaUsuario, aplican ambos métodos. El valor del atributo TABLESPACE de tales entidades será el último valor asignado, es decir, "TS02". Nótese que valor del atributo WITH (especificado en el primer método) será el mismo para todas las entidades persistentes, incluyendo TareaUsuario y las entidades que extienden TareaUsuario.

El método **attachAddAttributesMethods** tiene una sola firma, la cual se describe a continuación.

attachAddAttributesMethods(Class<?> clazz)

Agrega una clase a la lista de clases que contienen métodos **AddAttributes** del proyecto. Su único parámetro es:

- **clazz**: clase que contiene métodos **AddAttributes**.

Método addAttribute

El método **AddAttribute** es un método de todo artefacto (objeto que implementa la interfaz **Artifact**), que permite agregar un atributo a la lista de atributos extraordinarios del artefacto.

Los atributos extraordinarios son parejas clave/valor, de modo que si se agregan varios atributos con la misma clave a un artefacto, el valor de tal atributo será el último valor agregado.

El método **AddAttribute** tiene las siguientes firmas:

Object addAttribute(String name, Object value)

Agrega un atributo a la lista de atributos extraordinarios del artefacto. Sus parámetros son:

- **name**: clave del atributo.
- **value**: valor del atributo.

Object addAttribute(String name, Object... value)

Agrega un atributo a la lista de atributos extraordinarios del artefacto. Sus parámetros son:

- **name**: clave del atributo.
- **value**: valor del atributo. A diferencia de la firma anterior, el valor es una lista de valores.

Object addAttribute(Property property, String name, Object value)

Agrega un atributo a la lista de atributos extraordinarios del artefacto. Esta firma solo aplica a meta-entidades (artefactos que implementan la interfaz **Entity**). Sus parámetros son:

- **property**: meta-propiedad (de la meta-entidad) a la que corresponde el atributo.
- **name**: clave del atributo.
- **value**: valor del atributo.

Método clearAddAttributesMethods

El método **clearAddAttributesMethods** del meta-proyecto se utiliza para borrar la lista de clases que contienen métodos **AddAttributes**.

El método **clearAddAttributesMethods** tiene una sola firma, la cual se describe a continuación.

clearAddAttributesMethods()

Borra la lista de clases que contienen métodos **AddAttributes**.

Método putEnvironmentVariable

El método **putEnvironmentVariable** del meta-proyecto se utiliza para agregar una variable a la lista de variables de ambiente del proyecto. El método puede utilizarse repetidamente para agregar varias variables.

Las variables de ambiente son parejas clave/valor, de modo que si se agregan varias variables con la misma clave, el valor de tal variable será el último valor agregado.

El método **putEnvironmentVariable** tiene una sola firma, la cual se describe a continuación.

putEnvironmentVariable(String key, String value)

Agrega una variable a la lista de variables de ambiente del proyecto. Sus parámetros son:

- **key**: clave de la variable.
- **value**: valor de la variable.

Método setAlertLoggingLevel

El método **setAlertLoggingLevel** del meta-proyecto se utiliza para establecer el nivel de severidad de los mensajes de alerta que se deben mostrar al generar el proyecto. El valor predeterminado de esta propiedad es WARN (mostrar los mensajes de alerta cuyo nivel de severidad sea WARN, ERROR o FATAL).

El método **setAlertLoggingLevel** tiene una sola firma, la cual se describe a continuación.

setAlertLoggingLevel(LoggingLevel logginglevel)

Establece el nivel de severidad de los mensajes de alerta. Su único parámetro es:

- **logginglevel** {ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF}: elemento de la enumeración **LoggingLevel** que determina el nivel de severidad de los mensajes de alerta que se deben mostrar al generar el proyecto. Especifique ALL para mostrar todos los mensajes de alerta. Especifique TRACE, DEBUG, INFO, WARN, ERROR o FATAL para mostrar los mensajes de alerta cuyo nivel de severidad sea mayor o igual al nivel especificado. Especifique OFF para no mostrar ningún mensaje de alerta.

Método setAlias

El método **setAlias** del meta-proyecto se utiliza para establecer el alias (código alterno) del proyecto. El alias solo puede contener letras minúsculas y números, debe comenzar por una letra, y no puede ser **jee1ap101**, **meta** o **workspace**. Se recomienda utilizar un alias que tenga el nombre de su proyecto como prefijo.

Si utiliza la plataforma **jee1**, el alias del proyecto maestro es el nombre del directorio raíz de los archivos generados; por lo tanto, se debe establecer un alias diferente antes de cada ejecución del método **generate** (Vea Definición de proyectos maestros en este mismo documento).

El método **setAlias** tiene una sola firma, la cual se describe a continuación.

setAlias(String alias)

Establece el alias del proyecto. Su único parámetro es:

- **alias**: código alterno del proyecto.

Método setDefaultDescription

El método **setDefaultDescription** del meta-proyecto se utiliza para establecer la descripción del proyecto que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultDescription** debe ejecutarse en el método **settleAttributes** del meta-proyecto y tiene una sola firma, la cual se describe a continuación.

setDefaultDescription(String defaultDescription)

Establece la descripción del proyecto. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen el proyecto.

Método setDefaultShortDescription

El método **setDefaultShortDescription** del meta-proyecto se utiliza para establecer la descripción corta del proyecto que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la descripción.

El método **setDefaultShortDescription** debe ejecutarse en el método **settleAttributes** del meta-proyecto y tiene una sola firma, la cual se describe a continuación.

setDefaultShortDescription(String defaultDescription)

Establece la descripción corta del proyecto. Su único parámetro es:

- **defaultDescription**: una o más oraciones que describen brevemente el proyecto.

Método setDefaultLabel

El método **setDefaultLabel** del meta-proyecto se utiliza para establecer la etiqueta del proyecto que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultLabel** debe ejecutarse en el método **settleAttributes** del meta-proyecto y tiene una sola firma, la cual se describe a continuación.

setDefaultLabel(String defaultLabel)

Establece la etiqueta del proyecto. Su único parámetro es:

- **defaultLabel**: sustantivo singular que se usa como etiqueta del proyecto.

Método setDefaultShortLabel

El método **setDefaultShortLabel** del meta-proyecto se utiliza para establecer la etiqueta corta del proyecto que se almacena en el archivo de recursos por defecto. En caso de que el archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo de recursos por defecto para obtener el valor de la etiqueta.

El método **setDefaultShortLabel** debe ejecutarse en el método **settleAttributes** del meta-proyecto y tiene una sola firma, la cual se describe a continuación.

setDefaultShortLabel(String defaultLabel)

Establece la etiqueta corta del proyecto. Su único parámetro es:

- **defaultLabel**: sustantivo singular, preferiblemente simple, que se usa como etiqueta corta del proyecto.

Método setDetailLoggingLevel

El método **setDetailLoggingLevel** del meta-proyecto se utiliza para establecer el nivel de severidad de los mensajes de detalle que se deben mostrar al generar el proyecto. El valor predeterminado de esta propiedad es OFF (no mostrar ningún mensaje de detalle).

El método **setDetailLoggingLevel** tiene una sola firma, la cual se describe a continuación.

setDetailLoggingLevel(LoggingLevel logginglevel)

Establece el nivel de severidad de los mensajes de detalle. Su único parámetro es:

- **logginglevel** {ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF}: elemento de la enumeración **LoggingLevel** que determina el nivel de severidad de los mensajes de detalle que se deben mostrar al generar el proyecto. Especifique ALL para mostrar todos los mensajes de detalle. Especifique TRACE, DEBUG, INFO, WARN, ERROR o FATAL para mostrar los mensajes de detalle cuyo nivel de severidad sea mayor o igual al nivel especificado. Especifique OFF para no mostrar ningún mensaje de detalle.

Método setInternetAccessAllowed

El método **setInternetAccessAllowed** del meta-proyecto se utiliza para especificar si el proyecto generado incluye, o no, características que requieren acceso a internet para su funcionamiento (por ejemplo, si la página **Inicio de Sesión** incluye, o no, un elemento Google reCaptcha). El valor predeterminado de esta propiedad es `false` (no se incluyen características que requieren acceso a internet).

El método **setInternetAccessAllowed** tiene una sola firma, la cual se describe a continuación.

setInternetAccessAllowed(boolean internetAccessAllowed)

Especifica si el proyecto generado incluye, o no, características que requieren acceso a internet. Su único parámetro es:

- **internetAccessAllowed**: `true`, si el proyecto generado incluye características que requieren acceso a internet; de lo contrario `false`.

Método setLocale

El método **setLocale** del meta-proyecto se utiliza para establecer el objeto *Locale* predeterminado del proyecto (un objeto *Locale* representa una o región geográfica, política, cultural, etc.). El objeto *Locale* determina, entre otras cosas, el idioma del archivo de recursos por defecto y el formato predeterminado de fecha y hora. El valor predeterminado de esta propiedad es el *Locale* predeterminado de la JVM.

El método **setLocale** tiene una sola firma, la cual se describe a continuación.

setLocale(Locale locale)

Establece *Locale* predeterminado del proyecto. Su único parámetro es:

- **locale**: objeto *Locale*.

Método setRoleBasedAccessControllerName

El método **setRoleBasedAccessControllerName** del meta-proyecto se utiliza para establecer el nombre del controlador de seguridad del proyecto. El valor predeterminado de esta propiedad es el alias del proyecto (vea Método setAlias).

El método **setRoleBasedAccessControllerName** debe ejecutarse en el método **settleAttributes** del meta-proyecto y tiene una sola firma, la cual se describe a continuación.

setRoleBasedAccessControllerName(String roleBasedAccessControllerName)

Establece el nombre del controlador de seguridad del proyecto. Su único parámetro es:

- **roleBasedAccessControllerName**: nombre del controlador de seguridad del proyecto. Especifique LDAP si el controlador de seguridad del proyecto cumple el protocolo LDAP. En caso contrario especifique el alias del proyecto.

Método setSecurityRealmType

El método **setSecurityRealmType** del meta-proyecto se utiliza para establecer el tipo de dominio de seguridad del proyecto. El valor predeterminado de esta propiedad es JDBC.

El método **setSecurityRealmType** debe ejecutarse en el método **settleAttributes** del meta-proyecto y tiene una sola firma, la cual se describe a continuación.

setSecurityRealmType(SecurityRealmType securityRealmType)

Establece el tipo de dominio de seguridad del proyecto. Su único parámetro es:

- **securityRealmType** {JDBC, LDAP}: elemento de la enumeración **SecurityRealmType** que corresponde el tipo de dominio de seguridad del proyecto. Especifique LDAP si el controlador de seguridad del proyecto cumple el protocolo LDAP. En caso contrario especifique JDBC.

Método setTrackingLoggingLevel

El método **setTrackingLoggingLevel** del meta-proyecto se utiliza para establecer el nivel de severidad de los mensajes de seguimiento que se deben mostrar al generar el proyecto. El valor predeterminado de esta propiedad es OFF (no mostrar ningún mensaje de detalle).

El método **setTrackingLoggingLevel** tiene una sola firma, la cual se describe a continuación.

setTrackingLoggingLevel(LoggingLevel logginglevel)

Establece el nivel de severidad de los mensajes de seguimiento. Su único parámetro es:

- **logginglevel** {ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF}: elemento de la enumeración **LoggingLevel** que determina el nivel de severidad de los mensajes de seguimiento que se deben mostrar al generar el proyecto. Especifique ALL para mostrar todos los mensajes de seguimiento. Especifique TRACE, DEBUG, INFO, WARN, ERROR o FATAL para mostrar los mensajes de seguimiento cuyo nivel de severidad sea mayor o igual al nivel especificado. Especifique OFF para no mostrar ningún mensaje de seguimiento.

Método setUserEntityClass

El método **setUserEntityClass** del meta-proyecto se utiliza para establecer la clase de la meta-entidad que corresponde a la entidad **Usuario** del proyecto.

El método **setUserEntityClass** debe ejecutarse en el método **settleAttributes** del meta-proyecto y tiene una sola firma, la cual se describe a continuación.

setUserEntityClass(Class<? extends Entity> clazz)

Establece la clase de la meta-entidad que corresponde a la entidad **Usuario** del proyecto. Su único parámetro es:

- **clazz**: clase de la meta-entidad que corresponde a la entidad **Usuario** del proyecto.

Si utiliza la plataforma **jee1**, la meta-entidad que corresponde a la entidad **Usuario** es **meta.entidad.comun.control.acceso.Usuario**. Si su proyecto maestro extiende la clase **ProyectoBase** ya tiene establecida la clase de la meta-entidad que corresponde a la entidad **Usuario** porque en el constructor de **ProyectoBase** se ejecuta el método **setUserEntityClass**.

Generación de datos

Como parte de la meta-programación, se pueden especificar diferentes opciones para generar datos que le ayudarán a probar y demostrar su aplicación. Estas opciones se especifican decorando meta-entidades y meta-propiedades con anotaciones específicas para este propósito, como se explica en la siguiente sección.

Definición de opciones para generar datos mediante anotaciones

Para generar datos para una entidad, decore la correspondiente meta-entidad con la anotación **@EntityDataGen**. Los elementos de la anotación son:

- **start** [1:10.000]: especifica el primer número de la serie. Debe ser un número entero del 1 al 10.000 y debe ser menor o igual que el valor especificado, o determinado, para el elemento **stop**. El valor predeterminado es 1.
- **stop** [1:10.000]: especifica el último número de la serie. Debe ser un número entero del 1 al 10.000 y debe ser mayor o igual que el valor especificado, o determinado, para el elemento **start**. El valor predeterminado es 100.
- **step** [0:1]: especifica el intervalo entre los números de la serie. Debe ser 0 o 1. El valor predeterminado es 1.

Solo se generarán datos si **start** es menor o igual que **stop** y **step** es igual a 1. El número de filas generadas será: **stop - start + 1**.

Generación de referencias a otras entidades

Decore las propiedades que hacen referencia a otras entidades con la anotación **@EntityReferenceDataGen**. Los elementos de la anotación son:

- **type** {UNSPECIFIED, DEFAULT, SERIES, RANDOM}: especifica el método para generar datos para esta propiedad. Su valor es uno de los elementos de la enumeración **DataGenType**. Seleccione DEFAULT para generar el valor por omisión de la propiedad (para dar un valor por omisión a una propiedad se utiliza el método setDefaultValue). Seleccione SERIES para generar un valor en serie. Seleccione RANDOM para generar un valor aleatorio. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RANDOM.
- **nullable** [0:100]: especifica el porcentaje de valores nulos que se generarán. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM. El valor de **nullable** debe ser un número entero del 0 al 100. El valor predeterminado es 0. Si el valor de **nullable** es 100, solo se generarán valores nulos.

Generación de datos booleanos

Decore las propiedades de tipo **BooleanProperty** con la anotación **@BooleanDataGen**. Los elementos de la anotación son:

- **type** {UNSPECIFIED, DEFAULT, SERIES, RANDOM}: especifica el método para generar datos para esta propiedad. Su valor es uno de los elementos de la enumeración **DataGenType**. Seleccione DEFAULT para generar el valor por omisión de la propiedad (para dar un valor por omisión a una propiedad se utiliza el método setDefaultValue). Seleccione SERIES para generar un valor en serie. Seleccione RANDOM para generar un valor aleatorio. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RANDOM.

- **nullable** [0:100]: especifica el porcentaje de valores nulos que se generarán. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM. El valor de **nullable** debe ser un número entero del 0 al 100. El valor predeterminado es 0. Si el valor de **nullable** es 100, solo se generarán valores nulos. Si **nullable + trueable > 100**, entonces se ajustará el valor de **nullable** a **100 – trueable**, y no se generarán valores FALSE.
- **trueable** [0:100]: especifica el porcentaje de valores TRUE que se generarán. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM. El valor de **trueable** debe ser un número entero del 0 al 100. El valor predeterminado es 0. Si el valor de **trueable** es 0, todos los valores no nulos serán FALSE. Si el valor de **nullable** es 100, solo se generarán valores TRUE. Si **nullable + trueable > 100**, entonces se ajustará el valor de **nullable** a **100 – trueable**, y no se generarán valores FALSE.
- **function**: especifica el nombre de la función definida por el usuario que se utilizará para generar los datos. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. La función no se ejecuta si el valor generado por el método RANDOM o SERIES es nulo. Los parámetros que recibe la función son:
  - **tabla**: nombre de la tabla; su tipo de dato corresponde a java.lang.String.
  - **columna**: nombre de la columna; su tipo de dato corresponde a java.lang.String.
  - **clave**: clave primaria de la fila; su tipo de dato corresponde a java.lang.Integer.
  - **secuencia**: número de secuencia de la fila; su tipo de dato corresponde a java.lang.Integer.
  - **valor**: valor calculado por el método RANDOM o SERIES; su tipo de dato corresponde a java.lang.Boolean.

Generación de datos alfanuméricos

Decore las propiedades de tipo **StringProperty** con la anotación **@CharacterDataGen**. Los elementos de la anotación son:

- **type** {UNSPECIFIED, DEFAULT, SERIES, RANDOM}: especifica el método para generar datos para esta propiedad. Su valor es uno de los elementos de la enumeración **DataGenType**. Seleccione DEFAULT para generar el valor por omisión de la propiedad (para dar un valor por omisión a una propiedad se utiliza el método setDefaultValue). Seleccione SERIES para generar un valor en serie, concatenando **prefix**, **secuencia** y **suffix**, donde el valor de **prefix** y **suffix** viene dado por sus correspondientes elementos, descritos más adelante; y el valor de **secuencia** se obtiene de una serie cíclica que se define utilizando los elementos **start**, **stop** y **step**, también descritos más adelante. Seleccione RANDOM para generar un valor aleatorio. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RANDOM.
- **start** [1:10.000]: especifica el primer número de la serie. Debe ser un número entero del 1 al 10.000 y debe ser menor o igual que el valor especificado, o determinado, para el elemento **stop**. El valor predeterminado es 1.
- **stop** [1:10.000]: especifica el último número de la serie. Debe ser un número entero del 1 al 10.000 y debe ser mayor o igual que el valor especificado, o determinado, para el elemento **start**. El valor predeterminado es 10.000.
- **step** [1:10.000]: especifica el intervalo entre los números de la serie. Debe ser un número entero del 1 al 10.000. El valor predeterminado es 1.
- **nullable** [0:100]: especifica el porcentaje de valores nulos que se generarán. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM. El valor de **nullable** debe ser un número entero del 0 al 100. El valor predeterminado es 0. Si el valor de **nullable** es 100, solo se generarán valores nulos.
- **pattern**: especifica la cadena de caracteres que se utiliza como patrón en él método RANDOM. Los caracteres del patrón se copian al resultado o se sustituyen por caracteres generados aleatoriamente, de la siguiente manera: cada letra **a** minúscula se sustituye por una letra minúscula; cada letra **A** mayúscula, por una letra mayúscula; cada signo de interrogación (**?**), por una letra, minúscula o mayúscula; cada dígito **0**, por un dígito; cada letra **x** minúscula**,** por un dígito o una letra minúscula; cada letra **X** mayúscula, por un dígito o una letra mayúscula; y cada

asterisco (*), por un dígito o una letra, minúscula o mayúscula; los demás caracteres son copiados. Para evitar la sustitución de alguno de los caracteres especiales previamente mencionados (**a, A, ?, 0, x, X, ***), éste debe ser precedido por una barra oblicua inversa (****). Cuando un carácter es precedido por una barra oblicua inversa se omite la barra y se copia el carácter al resultado; por lo tanto, para copiar una barra oblicua inversa debe escribirla dos veces seguidas (****).

- **prefix**: especifica la cadena de caracteres que se utiliza como prefijo de los valores generados por los métodos RANDOM y SERIES.
- **suffix**: especifica la cadena de caracteres que se utiliza como sufijo de los valores generados por los métodos RANDOM y SERIES.
- **function**: especifica el nombre de la función definida por el usuario que se utilizará para generar los datos. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. La función no se ejecuta si el valor generado por el método RANDOM o SERIES es nulo. Los parámetros que recibe la función son:
  - **tabla**: nombre de la tabla; su tipo de dato corresponde a java.lang.String.
  - **columna**: nombre de la columna; su tipo de dato corresponde a java.lang.String.
  - **clave**: clave primaria de la fila; su tipo de dato corresponde a java.lang.Integer.
  - **secuencia**: número de secuencia de la fila; su tipo de dato corresponde a java.lang.Integer.
  - **valor**: valor calculado por el método RANDOM o SERIES; su tipo de dato corresponde a java.lang.String.

Generación de datos numéricos

Decore las propiedades de tipo **BigDecimalProperty, BigIntegerProperty, ByteProperty, DoubleProperty, FloatProperty, IntegerProperty, LongProperty** y **ShortProperty**, con la anotación **@NumericDataGen**. Los elementos de la anotación son:

- **type** {UNSPECIFIED, DEFAULT, SERIES, RANDOM}: especifica el método para generar datos para esta propiedad. Su valor es uno de los elementos de la enumeración **DataGenType**. Seleccione DEFAULT para generar el valor por omisión de la propiedad (para dar un valor por omisión a una propiedad se utiliza el método setDefaultValue). Seleccione SERIES para generar un valor en serie, mediante la fórmula **min + factor × secuencia**, donde el valor de **min** y **factor** viene dado por sus correspondientes elementos, descritos más adelante; y el valor de **secuencia** se obtiene de una serie cíclica que se define utilizando los elementos **start**, **stop** y **step**, también descritos más adelante. Seleccione RANDOM para generar un valor aleatorio, mediante la fórmula **min + factor × f(random),** donde el valor de **min** y **factor** viene dado por sus correspondientes elementos, descritos más adelante; la acción que realiza la función **f** viene dada por el elemento **action**, también descrito más adelante; y el valor de **random** se obtiene de manera aleatoria entre **min** y **max**, donde el valor de **min** y **max** viene dado por sus correspondientes elementos, también descritos más adelante. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RANDOM.
- **start** [1:10.000]: especifica el primer número de la serie. Debe ser un número entero del 1 al 10.000 y debe ser menor o igual que el valor especificado, o determinado, para el elemento **stop**. El valor predeterminado es 1.
- **stop** [1:10.000]: especifica el último número de la serie. Debe ser un número entero del 1 al 10.000 y debe ser mayor o igual que el valor especificado, o determinado, para el elemento **start**. El valor predeterminado es 10.000.
- **step** [1:10.000]: especifica el intervalo entre los números de la serie. Debe ser un número entero del 1 al 10.000. El valor predeterminado es 1.
- **nullable** [0:100]: especifica el porcentaje de valores nulos que se generarán. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM. El valor de **nullable** debe ser un número entero del 0 al 100. El valor predeterminado es 0. Si el valor de **nullable** es 100, solo se generarán valores nulos.

- **min**: especifica el mínimo número **random** generado. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. Debe ser un número entero del -2147483648 al 2147483647. El valor predeterminado es el valor mínimo de la propiedad (para dar un valor mínimo a una propiedad se utiliza el método setMinValue). El número **random** generado es un resultado intermedio que puede ser redondeado o truncado, multiplicado por **factor** y comparado contra el valor mínimo y máximo de la propiedad para obtener el resultado final. En ningún caso el resultado final será menor que el valor mínimo de la propiedad.
- **max**: especifica el máximo número **random** generado. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. Debe ser un número entero del -2147483648 al 2147483647. El valor predeterminado es el valor máximo de la propiedad (para dar un valor máximo a una propiedad se utiliza el método setMaxValue). El número **random** generado es un resultado intermedio que puede ser redondeado o truncado, multiplicado por **factor** y comparado contra el valor mínimo y máximo de la propiedad para obtener el resultado final. En ningún caso el valor generado será mayor que el valor máximo de la propiedad.
- **action** [UNSPECIFIED, ROUND, TRUNCATE]: especifica la acción que se realiza sobre el número **random** generado. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM. Su valor es uno de los elementos de la enumeración **DataGenNumericAction**. Seleccione ROUND o TRUNCATE para redondear o truncar el número **random** generado antes de multiplicarlo por **factor**. Alternativamente, omita el elemento o seleccione UNSPECIFIED para no ejecutar ninguna acción previa a la multiplicación por **factor**.
- **factor**: especifica el factor que multiplica al número **random** generado. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. Puede ser cualquier número decimal. El valor predeterminado es 1.
- **function**: especifica el nombre de la función definida por el usuario que se utilizará para generar los datos. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. La función no se ejecuta si el valor generado por el método RANDOM o SERIES es nulo. Los parámetros que recibe la función son:
  - **tabla**: nombre de la tabla; su tipo de dato corresponde a java.lang.String.
  - **columna**: nombre de la columna; su tipo de dato corresponde a java.lang.String.
  - **clave**: clave primaria de la fila; su tipo de dato corresponde a java.lang.Integer.
  - **secuencia**: número de secuencia de la fila; su tipo de dato corresponde a java.lang.Integer.
  - **valor**: valor calculado por el método RANDOM o SERIES; su tipo de dato corresponde a java.lang.Number.

Generación de datos temporales

Decore las propiedades de tipo **DateProperty**, **TimeProperty** y **TimestampProperty**, con la anotación **@TemporalDataGen**. Los elementos de **@TemporalDataGen** son:

- **type** {UNSPECIFIED, DEFAULT, SERIES, RANDOM}: especifica el método para generar datos para esta propiedad. Su valor es uno de los elementos de la enumeración **DataGenType**. Seleccione DEFAULT para generar el valor por omisión de la propiedad (para dar un valor por omisión a una propiedad se utiliza el método setDefaultValue). Seleccione SERIES para generar un valor en serie, mediante la fórmula **min + secuencia**, donde el valor de **min** viene dado por su correspondiente elemento, descrito más adelante; el valor de **secuencia** se obtiene de una serie cíclica que se define utilizando los elementos **start**, **stop** y **step**, también descritos más adelante; y el intervalo al cual se suma el valor de **secuencia** viene dado por el valor del elemento **interval**, también descrito más adelante. Seleccione RANDOM para generar un valor aleatorio, mediante la fórmula **min + random,** donde el valor de **min** viene dado por su correspondiente elemento, descrito más adelante; y el valor de **random** es un número de días, minutos o segundos, para **DateProperty**, **TimeProperty** o **TimestampProperty**, respectivamente, que se obtiene de manera aleatoria entre **min** y **max**. Alternativamente, omita el elemento o seleccione

UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado del atributo es RANDOM.

- **start** [1:10.000]: especifica el primer número de la serie. Debe ser un número entero del 1 al 10.000 y debe ser menor o igual que el valor especificado, o determinado, para el elemento **stop**. El valor predeterminado es 1.
- **stop** [1:10.000]: especifica el último número de la serie. Debe ser un número entero del 1 al 10.000 y debe ser mayor o igual que el valor especificado, o determinado, para el elemento **start**. El valor predeterminado es 10.000.
- **step** [1:10.000]: especifica el intervalo entre los números de la serie. Debe ser un número entero del 1 al 10.000. El valor predeterminado es 1.
- **nullable** [0:100]: especifica el porcentaje de valores nulos que se generarán. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM. El valor de **nullable** debe ser un número entero del 0 al 100. El valor predeterminado es 0. Si el valor de **nullable** es 100, solo se generarán valores nulos.
- **min**: especifica el mínimo valor generado. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. Puede ser un valor constante, acorde con el tipo de dato. El formato de una constante de tipo **Date** es **yyyy-mm-dd**. El formato de una constante de tipo **Time** es **hh:mm:ss**. El formato de una constante de tipo **Timestamp** es **yyyy-mm-dd hh:mm:ss**. También puede ser una expresión para calcular un valor relativo a la fecha y hora de ejecución. Escriba un número entero, positivo o negativo, seguido de una letra que identifique el intervalo de tiempo al cual desea sumar el número. La letra **D** mayúscula corresponde a día; la **M** mayúscula, a mes; la **Y** mayúscula, a año; la **h** minúscula, a hora; la **m** minúscula, a minuto; y la **s** minúscula, a segundo. El valor predeterminado para las propiedades de tipo **DateProperty** es la fecha actual menos 70 años. El valor predeterminado para las propiedades de tipo **DateProperty** es la 00:00. El valor predeterminado para las propiedades de tipo **TimestampProperty** es la fecha actual menos 70 años y la hora actual
- **max**: especifica el máximo valor generado. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. Puede ser un valor constante, acorde con el tipo de dato. El formato de una constante de tipo **Date** es **yyyy-mm-dd**. El formato de una constante de tipo **Time** es **hh:mm:ss**. El formato de una constante de tipo **Timestamp** es **yyyy-mm-dd hh:mm:ss**. También puede ser una expresión para calcular un valor relativo a la fecha y hora de ejecución. Escriba un número entero, positivo o negativo, seguido de una letra que identifique el intervalo de tiempo al cual desea sumar el número. La letra **D** mayúscula corresponde a día; la **M** mayúscula, a mes; la **Y** mayúscula, a año; la **h** minúscula, a hora; la **m** minúscula, a minuto; y la **s** minúscula, a segundo. El valor predeterminado para las propiedades de tipo **DateProperty** es la fecha actual más 30 años. El valor predeterminado para las propiedades de tipo **DateProperty** es la 23:59. El valor predeterminado para las propiedades de tipo **TimestampProperty** es la fecha actual más 30 años y la hora actual.
- **interval** {UNSPECIFIED, YEAR, MONTH, DAY, HOUR, MINUTE, SECOND}: especifica el intervalo que se incrementa al calcular valores en serie. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es SERIES. Su valor es uno de los elementos de la enumeración **DataGenTemporalInterval**. Seleccione YEAR, MONTH, DAY, HOUR, MINUTE o SECOND para sumar el valor de **secuencia** al año, mes, día, hora, minuto o segundo, respectivamente. Alternativamente, omita el elemento o seleccione UNSPECIFIED para utilizar el valor predeterminado del atributo. El valor predeterminado para las propiedades de tipo **DateProperty** y **TimestampProperty** es DAY. El valor omisión para las propiedades de tipo **TimeProperty** es HOUR.

- **function**: especifica el nombre de la función definida por el usuario que se utilizará para generar los datos. Este elemento es relevante solo si el valor especificado, o determinado, para el elemento **type** es RANDOM o SERIES. La función no se ejecuta si el valor generado por el método RANDOM o SERIES es nulo. Los parámetros que recibe la función son:
    - **tabla**: nombre de la tabla; su tipo de dato corresponde a java.lang.String.
    - **columna**: nombre de la columna; su tipo de dato corresponde a java.lang.String.
    - **clave**: clave primaria de la fila; su tipo de dato corresponde a java.lang.Integer.
    - **secuencia**: número de secuencia de la fila; su tipo de dato corresponde a java.lang.Integer.
    - **valor**: valor calculado por el método RANDOM o SERIES; su tipo de dato corresponde a java.sql.Date o java.sql.Time o java.sql.Timestamp, según corresponda.

Apéndice 1: Meta-entidades de la plataforma jee1

1. AmbientePrueba
2. Aplicacion
3. ArchivoAdjunto
4. CasoPrueba
5. ClaseRecurso
6. CondicionEjeFun
7. CondicionTarea
8. ConjuntoSegmento
9. DocumentoPrueba
10. DocumentoPruebaX1
11. DocumentoPruebaX2
12. DocumentoPruebaX3
13. DocumentoPruebaX4
14. DocumentoPruebaX5
15. DocumentoPruebaX6
16. Dominio
17. DominioParametro
18. EjecucionLineaPrueba
19. EjecucionPrueba
20. ElementoSegmento
21. EscenarioPrueba
22. FiltroFuncion
23. FiltroFuncionPar
24. Funcion
25. FuncionParametro
26. GrupoProceso
27. LineaPrueba
28. NivelOpcionMenu
29. OpcionMenu
30. OperadorCom
31. Pagina
32. PaginaUsuario
33. PaquetePrueba
34. Parametro
35. ParametroLineaPrueba
36. ParteAmbientePrueba
37. PiezaAmbientePrueba
38. ProgramaPrueba
39. RastroFuncion
40. RastroFuncionPar
41. RastroInforme
42. RastroProceso
43. RecursoValor
44. Rol
45. RolFiltroFuncion
46. RolFuncion
47. RolFuncionPar
48. RolPagina
49. RolUsuario
50. Segmento
51. Tarea
52. TareaUsuario
53. TipoClaseRecurso
54. TipoComparacion
55. TipoDatoPar
56. TipoDocumentoPrueba
57. TipoDominio
58. TipoFuncion
59. TipoNodo
60. TipoPagina
61. TipoParametro
62. TipoParametroDom
63. TipoPiezaPrueba
64. TipoRastroFun
65. TipoRecurso
66. TipoResultadoPrueba
67. TipoRol
68. TipoValor
69. Usuario

Apéndice 2: Comparaciones con StandardRelationalOp

La siguiente tabla muestra la comparación con operadores SQL correspondiente a cada elemento de la enumeración **StandardRelationalOp**. En cada caso, "*columna*" corresponde al nombre de la columna que se compara; y el signo "**?**", al valor que se recibe como parámetro para construir la comparación.

Elemento	Comparación
EQ	*columna* = **?**
NEQ	*columna* <> **?**
GT	*columna* > **?**
GTEQ	*columna* >= **?**
LT	*columna* < **?**
LTEQ	*columna* <= **?**
LIKE	*columna* LIKE **?**
NOT_LIKE	*columna* NOT LIKE **?**
IS_NULL_OR_EQ	*columna* IS NULL OR *columna* = **?**
IS_NULL_OR_NEQ	*columna* IS NULL OR *columna* <> **?**
IS_NULL_OR_GT	*columna* IS NULL OR *columna* > **?**
IS_NULL_OR_GTEQ	*columna* IS NULL OR *columna* >= **?**
IS_NULL_OR_LT	*columna* IS NULL OR *columna* < **?**
IS_NULL_OR_LTEQ	*columna* IS NULL OR *columna* <= **?**
IS_NULL_OR_LIKE	*columna* IS NULL OR *columna* LIKE **?**
IS_NULL_OR_NOT_LIKE	*columna* IS NULL OR *columna* NOT LIKE **?**

Apéndice 3: Creación de la base de datos de aplicaciones jee1

La plataforma **jee1** incluye plantillas de scripts para la creación y administración de la base de datos de la aplicación en el servidor de base de datos PostgreSQL. Los scripts para Windows se generan en el subdirectorio **source\management\resources\scripts\windows\postgresql** del directorio raíz de la aplicación. La siguiente tabla muestra los scripts generados.

Script	Función
cleanup	Elimina los rastros de auditoria con más de un mes de antigüedad
copy-former	Crea copias de tablas en archivos de texto
createdb	Crea la base de datos de la aplicación
dropdb	Elimina la base de datos de la aplicación
dropmakedb	Elimina y crea la base de datos y los objetos de la aplicación
dump	Crea un archivo respaldo de toda la base de datos
makedb	Crea los objetos de la aplicación en la base de datos
maketestdb	Elimina y crea la base de datos y los objetos de la aplicación; adicionalmente carga las tablas de la aplicación con datos generados
migrate-former	Copia los datos de una versión anterior de la base de datos
migratedb	Copia los datos de una versión anterior de la base de datos
restore	Restaura la base de datos a partir de un archivo respaldo
vacuumdb	Reorganiza y comprime la base de datos

Estos scripts también se generan para Linux en el subdirectorio **source\management\resources\scripts\linux\postgresql**.

Para crear la base de datos por primera vez, se debe ejecutar primero el script **createdb** y luego el script **makedb**. Alternativamente, si se desea crear la base de datos de la aplicación con datos generados, se puede ejecutar solo el script **maketestdb**. Para más información sobre la generación de datos vea el capítulo Generación de datos.

Si después de crear la base de datos se hacen cambios al modelo de la aplicación, entonces se debe crear nuevamente la base de datos. En este caso se debe ejecutar el script **dropmakedb**. Alternativamente, si se desea crear la base de datos de la aplicación con datos generados, se puede ejecutar solo el script **maketestdb**. Alternativamente, si se desea crear la base de datos de la aplicación y copiar los datos de la versión anterior, se puede ejecutar el script **makedb** y luego el script **migratedb**.

El script **makedb** primero renombra el esquema **public** como **former**; luego crea nuevamente el esquema **public**, donde se crean todos los objetos de la aplicación; de esta manera, los datos anteriores quedan almacenados en los objetos del esquema **former**.

El script **migratedb** copia los datos almacenados en los objetos del esquema **former** a los objetos del esquema **public**. La copia se lleva a cabo ejecutando las funciones definidas en alguno de los archivos del subdirectorio **source\management\resources\database\postgresql\data-migration\custom-made**. Cada vez que la aplicación se genera, se produce un nuevo archivo con las funciones para copiar los datos, pero se almacena en otro subdirectorio, específicamente en **source\management\resources\database\postgresql\data-migration\base**. El nombre del archivo generado es **insert-select-past-data-*timestamp*.sql**, donde ***timestamp*** es el “sello de tiempo” del momento en el que se produce el archivo. Antes de ejecutar el script **migratedb** se debe copiar el archivo **insert-select-past-data-*timestamp*.sql** que corresponde a la revisión del modelo que generó los objetos del esquema **former**, del subdirectorio **source\management\resources\database\postgresql\data-migration\base** al subdirectorio **source\management\resources\database\postgresql\data-migration\custom-made**. Por esta razón se recomienda, al

concluir cada revisión importante del modelo, copiar el archivo **insert-select-past-data-*timestamp*.sql** al subdirectorio **source\management\resources\database\postgresql\data-migration\custom-made** y darle un nombre más representativo de la revisión realizada, pero sin cambiar la extensión **sql**. Por otra parte, dependiendo de los cambios realizados al modelo, las funciones generadas para copiar datos podrían necesitar ajustes manuales, previos a la ejecución del script **migratedb**. Entre los cambios que obligan a ajustar estas funciones se encuentran:

- Entidades eliminadas.
- Entidades renombradas.
- Propiedades de entidades eliminadas.
- Propiedades de entidades renombradas.
- Tipos de datos de propiedades incompatibles con el tipo anterior; por ejemplo, al cambiar una propiedad de DateProperty a IntegerProperty o viceversa.
- Nuevos límites y/o restricciones a los datos; por ejemplo, al reducir la cantidad máxima de caracteres (longitud) de una propiedad StringProperty o la cantidad de dígitos significativos (precisión) de una propiedad BigDecimalProperty.

Nota: antes de ejecutar por primera vez uno cualquiera de los scripts generados, se debe ejecutar el script **home-setup** que se encuentra en el directorio raíz de la aplicación. Luego se debe revisar y, en caso de ser necesario, ajustar las variables de entorno en los siguientes archivos, que el script **home-setup** crea en el directorio raíz de la aplicación:

- variables-conf
- variables-home
- variables-glassfish
- variables-postgresql
- asadmin.password

Normalmente solo las variables definidas en el script **variables-home** necesitan ser ajustadas a su ambiente de trabajo.

Método alternativo para la migración de datos

El script **migratedb** es apropiado para la migración de datos de prueba, ya que normalmente el volumen de estos datos es bajo. Existe otro método para la migración, que utiliza los scripts **copy-former** y **migrate-former** en lugar del **migratedb**, y que resulta más apropiado para migrar altos volúmenes de datos. La secuencia de pasos de este método alternativo es la siguiente:

- Crear el subdirectorio **source\management\backup\postgresql\former**.
- Dar autorización read/write al usuario de PostgreSQL sobre el subdirectorio creado en el paso anterior.
- Ejecutar el script **makedb**.
- Ejecutar el archivo `meta.util.SQLMerger` del proyecto meta; esto genera los siguientes archivos en el paquete `meta.sql.postgresql` del directorio test del proyecto:
  - **copy-former-*yyyymmdd*.sql** (***yyyymmdd*** es la fecha de creación del archivo)
  - **migrate-former-public-*yyyymmdd*.sql**
- Revisar y modificar (en caso de que sea necesario) el archivo **migrate-former-public-*yyyymmdd*.sql**. Al igual que para **insert-select-past-data-*timestamp*.sql**, dependiendo de los cambios realizados al modelo, las funciones generadas en **migrate-former-public-*yyyymmdd*.sql** podrían necesitar ajustes manuales. Los posibles ajustes se pueden ubicar fácilmente buscando la cadena WATCH OUT. A veces es solo una advertencia, otras veces es un error; el comentario contiguo indica cuál de los dos es. Aun cuando sea un error se podrá ejecutar si se está seguro de que la tabla en la que aparece tal error está vacía o, dependiendo del error, si se está seguro de que la columna a la que se refiere el

error no tiene valores nulos. Si no es así, se debe tomar la correspondiente acción para corregir el error (probablemente modificar el modelo y regenerar).

- Después de corregir los errores, copiar los archivos **copy-former-*yyyymmdd*.sql** y **migrate-former-public-*yyyymmdd*.sql** al subdirectorio **source\management\resources\database\postgresql\data-migration\custom-made**.
- Ejecutar el script **copy-former**; este script permite seleccionar un archivo **copy-former-*yyyymmdd*.sql** y lo ejecuta; esto crea copias de todas las tablas que se puedan copiar con un comando COPY, el cual es menos flexible pero más eficiente que el comando INSERT/SELECT. Las copias se crean en archivos de texto, en el directorio creado en el primer paso.
- Ejecutar el script **migrate-former**; este script permite seleccionar un archivo **migrate-former-*yyyymmdd*.sql** y lo ejecuta; esto copia los datos de los archivos de texto creados por **copy-former** y los datos del resto de las tablas mediante comandos INSERT/SELECT.

Apéndice 4: Configuración de GlassFish para aplicaciones jee1

La plataforma **jee1** incluye plantillas de scripts para la configuración y administración del servidor de aplicaciones GlassFish. Los scripts para Windows se generan en el subdirectorio **source\management\resources\scripts\windows\glassfish** del directorio raíz de la aplicación. La siguiente tabla muestra los scripts generados.

Script	Función
add-libraries	Agrega archivos JAR de librerías
create-jdbc	Crea el dominio de seguridad, los conjuntos de conexiones y demás recursos JDBC
create-jms	Crea las fábricas de conexiones, los conjuntos de conexiones y demás recursos JMS
create-jvm-options	Crea opciones JVM del sistema
create-jvm-options.en_US	Crea opciones JVM del sistema para el idioma inglés
create-jvm-options.es_VE	Crea opciones JVM del sistema para el idioma español
create-system-properties	Crea propiedades del sistema
delete-jdbc	Elimina el dominio de seguridad, los conjuntos de conexiones y demás recursos JDBC
delete-jms	Elimina las fábricas de conexiones, los conjuntos de conexiones y demás recursos JMS
delete-jvm-options	Elimina opciones JVM del sistema
delete-jvm-options.en_US	Elimina opciones JVM del sistema para el idioma inglés
delete-jvm-options.es_VE	Elimina opciones JVM del sistema para el idioma español
delete-system-properties	Elimina propiedades del sistema
deploy	Despliega la aplicación de empresa
domain-start	Inicia la ejecución del dominio
domain-stop	Detiene la ejecución del dominio
list-components	Produce una lista de los componentes desplegados
remove-libraries	Elimina archivos JAR de librerías
server-config	Establece parámetros de configuración del servidor de aplicaciones
undeploy	Anula el despliegue de la aplicación de empresa

Estos scripts también se generan para Linux en el subdirectorio **source\management\resources\scripts\linux\glassfish**.

Antes de ejecutar la aplicación por primera vez, se debe configurar el servidor GlassFish, tal como se describe en la sección Configuración de GlassFish para proyectos generados con jee1 de la guía de instalación.

Nota: antes de ejecutar por primera vez uno cualquiera de los scripts generados, se debe ejecutar el script **home-setup** que se encuentra en el directorio raíz de la aplicación. Luego se debe revisar y, en caso de ser necesario, ajustar las variables de entorno en los siguientes archivos, que el script **home-setup** crea en el directorio raíz de la aplicación:

- variables-conf
- variables-home
- variables-glassfish
- variables-postgresql
- asadmin.password

Normalmente solo las variables definidas en el script **variables-home** necesitan ser ajustadas a su ambiente de trabajo.

Apéndice 5: Refinación de aplicaciones jee1

Para refinar la aplicación se agregan los artefactos necesarios, teniendo el cuidado de no modificar ni eliminar los artefactos generados por Adalid (salvo los indicados más adelante, en la sección Archivos generados que pueden ser modificados), de manera que posteriormente se puedan incorporar cambios al modelo y regenerar la aplicación sin perder la refinación.

La plataforma **jee1** genera una aplicación de empresa (*enterprise application*) JEE. El nombre de esta aplicación, y por lo tanto el nombre del directorio donde se genera, es el **alias** del proyecto generado (vea Método setAlias). La aplicación de empresa está integrada por dos módulos Java EE (un módulo EJB y un módulo Web) y dos librerías Java SE. El nombre del módulo EJB, y por lo tanto el nombre del directorio donde se genera, es el nombre de la aplicación de empresa con el sufijo **–ejb**. El nombre del módulo Web, y por lo tanto el nombre del directorio donde se genera, es el nombre de la aplicación de empresa con el sufijo **–war**. El nombre de la primera librería, y por lo tanto el nombre del directorio donde se genera, es el nombre de la aplicación de empresa con el sufijo **–lib**. Por último, el nombre de la segunda librería, y por lo tanto el nombre del directorio donde se genera, es el nombre de la aplicación de empresa con el sufijo **–resources**. A diferencia de la primera, la cual es una librería de clases estándar, la segunda librería no contiene clases Java; es una librería de recursos que solo contiene archivos de configuración y archivos de propiedades para la internacionalización de la aplicación. En cada uno de los módulos y librerías hay subdirectorios específicos donde, en caso de ser necesario, se pueden crear paquetes y archivos adicionales sin el riesgo de que sean modificados o eliminados por Adalid al regenerar la aplicación, tal como se muestra en la siguiente tabla.

Módulo/Librería	Subdirectorio
alias-ejb	src\java\code
alias-war	src\java\code
	web\code
alias-lib	src\java\code
alias-resources	src\code

Además de aplicación de empresa, la plataforma **jee1** genera otros dos directorios: **development** y **management**. El primero contiene scripts y otros archivos utilizados en la fase de desarrollo; y el segundo, scripts, archivos de configuración, plantillas y otros archivos utilizados en las fases de desarrollo, prueba y producción. Tanto en el directorio **development** como en el directorio **management** se pueden crear subdirectorios y archivos adicionales, teniendo el cuidado de no modificar ni eliminar los generados por Adalid, salvo en algunos subdirectorios, los cuales están reservados para los archivos generados por la plataforma **jee1** o por la ejecución de scripts generados por la plataforma **jee1**. Los subdirectorios reservados son:

- management/backup
- management/dist
- management/logs
- management/resources/bootstrapping
- management/resources/config
- management/resources/database/postgresql/base
- management/resources/database/postgresql/data/base
- management/resources/database/postgresql/data-generation/base
- management/resources/database/postgresql/data-migration/base
- management/resources/database/postgresql/functions/base
- management/resources/database/postgresql/sequences/base
- management/resources/database/postgresql/tables/base
- management/resources/database/postgresql/triggers/base

- management/resources/database/postgresql/util/base
- management/resources/database/postgresql/views/base
- management/resources/database/postgresql/views/data-provider
- management/resources/database/postgresql/views/ jasper-report
- management/resources/database/postgresql/views/system
- management/resources/database/postgresql/views/zymurgy/base
- management/resources/database/postgresql/wrap-up/base
- management/resources/reporting/jasper/reports/entity-list
- management/resources/velocity
- management/sql

Archivos generados que pueden ser modificados

Existen archivos generados que si pueden ser modificados. Estos archivos solo se generan si el archivo no existe, de modo que las modificaciones no se pierden al regenerar la aplicación. La siguiente tabla muestra los archivos que pueden ser modificados.

Subdirectorio	Archivos
alias-war\web	ayuda.html
alias-war\web\resources	Hojas de estilo (*.css)
	Imágenes (*.gif, *.jpg, *.png, etc.)
development\resources\reporting\jasper\templates	Hojas de estilo (*.jrtx)
	Imágenes (*.gif, *.jpg, *.png, etc.)
management\resources\reporting\jasper\reports	Hojas de estilo (*.jrtx)
	Imágenes (*.gif, *.jpg, *.png, etc.)

Por ejemplo, es común modificar los archivos del subdirectorio ***alias***-war\web\resources que corresponden a las imágenes que aparecen en las páginas **Menú Principal** e **Inicio de Sesión**. Estos archivos son:

- mastbody.png
- masthead_left.png
- masthead_right.png

También es común modificar las hojas de estilo (*.jrtx) y las imágenes (*.gif, *.jpg, *.png, etc.) utilizadas en los informes, las cuales se encuentran en los siguientes subdirectorios:

- development\resources\reporting\jasper\templates\resources
- management\resources\reporting\jasper\reports\custom-made\\resources
- management\resources\reporting\jasper\reports\custom-made\subreports\resources
- management\resources\reporting\jasper\reports\ entity-list\\resources
- management\resources\reporting\jasper\reports\ entity-list\subreports\resources

Implementación de operaciones de negocio mediante funciones de base de datos

La manera más sencilla, dinámica y eficiente de implementar las operaciones de negocio es mediante funciones de PostgreSQL. Para cada operación se genera una función *stub* (sustituto temporal de código aún no desarrollado) que no hace nada; esto permite ejecutar la aplicación antes de terminar la codificación de todas las operaciones de negocio definidas en el modelo. Todas las funciones *stub* se generan en el archivo **create-business-function.sql**, ubicado en el subdirectorio **source\management\sql\postgresql** del directorio raíz de la aplicación.

Además de las funciones *stub*, el archivo también contiene las funciones de control de las operaciones, las cuales ejecutan, entre otras, a las funciones *stub*. Los nombres de las funciones de control tienen la forma ***entidad*$*operacion***, en donde ***entidad*** corresponde al nombre SQL de

la entidad a la cual pertenece la operación; y ***operacion*** corresponde al nombre SQL de la operación. El nombre de las funciones *stub* depende del valor del atributo **overloading** de la operación (vea Anotación ProcessOperationClass). Si el valor de **overloading** es TRUE, el nombre de la función *stub* será igual al de la función de control; para diferenciar la firma, la función *stub* pierde el parámetro **_super$**, que es el primero de los parámetros de la función de control y corresponde al número de identificación del rastro de la ejecución de la operación. Si el valor de **overloading** es FALSE, el nombre de la función *stub* será el nombre de la función de control más el sufijo **$biz**; en este caso, la lista de parámetros de la función *stub* y la de la función de control son iguales.

Continuando con el ejemplo de la entidad Persona, a continuación se muestra una de sus funciones *stub*.

```
create or replace function
persona$enviar_felicitacion$biz(_super$ bigint, _persona$ bigint)
returns integer as $$
begin
    return 0;
end;
$$ language plpgsql;
```

Para implementar una operación de negocio mediante una función de PostgreSQL, se debe sobrescribir la correspondiente función *stub*. Los scripts que se utilizan para crear la base de datos (vea el apéndice 3) ejecutan el archivo **create-business-function.sql** antes de ejecutar los archivos con extensión **sql** que se encuentran en el subdirectorio **source\management\resources\database\postgresql\functions**. Por lo tanto, para implementar una operación de negocio mediante una función de PostgreSQL, sin modificar el archivo **create-business-function.sql** generado, se deben seguir los siguientes pasos:

- Crear un subdirectorio, preferiblemente con el nombre de la entidad a la cual pertenece la operación, en el subdirectorio **source\management\resources\database\postgresql\functions\custom-made**.
- Crear un archivo con extensión **sql**, preferiblemente con el nombre de la operación, en el subdirectorio correspondiente a la entidad, creado en el paso anterior.
- Codificar la función en el archivo correspondiente a la operación, creado en el paso anterior. Se puede comenzar copiando la función *stub*. Note que se debe utilizar **create or replace function** para sobrescribir la función *stub*.

Implementación de operaciones de negocio mediante métodos de clases EJB

La manera de implementar las operaciones de negocio con independencia del gestor de base de datos es mediante métodos de clases EJB. Para cada entidad que tiene operaciones de negocio definidas en el modelo se genera una clase EJB en el paquete ***alias*.ejb.business.process.logic** del módulo EJB de la aplicación, donde ***alias*** corresponde al alias del proyecto generado. En esa clase se genera un método para cada operación de negocio de la entidad, el cual ejecuta la función *stub* de PostgreSQL; esto permite ejecutar la aplicación antes de terminar la codificación de todas las operaciones de negocio definidas en el modelo.

Los nombres de estas clases EJB tienen la forma ***Entidad*BusinessProcessLogicBean**, donde ***Entidad*** corresponde al nombre de la entidad. Los nombres de los métodos son iguales al de sus correspondientes operaciones.

Para implementar las operaciones de negocio de una entidad mediante métodos de una clase EJB, sin modificar la clase ***Entidad*BusinessProcessLogicBean** generada, se deben seguir los siguientes pasos:

- Crear el paquete ***alias*.ejb.business.process.logic.impl** en el subdirectorio **src\java\code** del módulo EJB de la aplicación.
- Crear una clase EJB con el nombre ***Entidad*BusinessProcessLogicImpl**, donde ***Entidad*** corresponde al nombre de la entidad cuyas operaciones se desea implementar, en el paquete creado en el paso anterior. Alternativamente, se puede copiar la clase ***Entidad*BusinessProcessLogicImpl** que se genera en el paquete ***alias*.ejb.business.process.logic.impl** del subdirectorio **test** del módulo EJB de la aplicación.
- Codificar los métodos correspondientes a las operaciones de negocio en la clase correspondiente a la entidad, creada o copiada en el paso anterior. Si no se copió la clase ***Entidad*BusinessProcessLogicImpl** del subdirectorio **test**, se puede comenzar copiando los métodos de la clase ***Entidad*BusinessProcessLogicBean**.

Implementación de operaciones de negocio calendarizadas

Las aplicaciones generadas con la plataforma **jee1** permiten ejecutar operaciones de negocio de acuerdo a un calendario predeterminado. El calendario se define mediante la clase **SchedulerBean** del paquete ***alias*.ejb.core.scheduler**, el cual se encuentra en el módulo EJB de la aplicación.

Para implementar las operaciones calendarizadas mediante métodos de una clase EJB se deben seguir los siguientes pasos:

- Crear el paquete ***alias*.ejb.business.process.logic.impl** en el subdirectorio **src\java\code** del módulo EJB de la aplicación.
- Crear una clase EJB con el nombre **BusinessProcessLogicSchedulerImpl**, en el paquete creado en el paso anterior. La clase debe ser un *bean* de sesión sin estado, por lo que debe decorarse con la anotación **@Stateless**, y debe implementar la interfaz ***alias*.ejb.business.process.logic.BusinessProcessLogicSchedulerLocal**.
- Codificar los métodos de la interfaz **BusinessProcessLogicSchedulerLocal** en la clase creada en el paso anterior.

Si existe la clase **BusinessProcessLogicSchedulerImpl**, el método **diario()** se ejecuta cada día, a las 12:00 AM; el método **semanal()** se ejecuta los días lunes, a las 2:00 AM; y el método **mensual()** se ejecuta el primer día de cada mes, a las 4:00 AM.

Si no existe la clase **BusinessProcessLogicSchedulerImpl**, entonces las operaciones pueden ser implementadas mediante funciones de PostgreSQL. Si en la base de datos existe la función ***alias*$diario()**, ésta se ejecuta cada día, a las 12:00 AM. Si en la base de datos existe la función ***alias*$semanal()**, ésta se ejecuta los días lunes, a las 2:00 AM. Si en la base de datos existe la función ***alias*$mensual()**, ésta se ejecuta el primer día de cada mes, a las 4:00 AM.

Implementación de operaciones de negocio para generar informes

La plataforma **jee1** utiliza JasperReports para generar informes. Para cada una de las operaciones de negocio para generar informes se debe codificar un archivo **jrxml** de JasperReports, utilizando iReport Designer u otra herramienta funcionalmente equivalente. La plataforma **jee1** genera plantillas de informes en el subdirectorio **development\resources\reporting\jasper\templates**, que pueden ser utilizadas con iReport Designer. Los archivos **jrxml** codificados se deben colocar en el subdirectorio **source\management\resources\reporting\jasper\reports\custom-made** del directorio raíz de la aplicación. Los nombres de estos archivos deben tener la forma ***entidad*$*operacion*.jrxml**, en donde ***entidad*** corresponde al nombre SQL de la entidad a la cual pertenece la operación; y ***operacion*** corresponde al nombre SQL de la operación para generar informes.

Por regla general, los informes utilizan las vistas generadas. En caso de ser necesario, se pueden definir vistas adicionales para los informes, creando un archivo con extensión **sql** para cada vista, y

colocándolo en el subdirectorio **source\management\resources\database\postgresql\views\custom-made** del directorio raíz de la aplicación. Los scripts que se utilizan para crear la base de datos (vea el apéndice 3) ejecutan todos los archivos con extensión **sql** que se encuentran en el subdirectorio **source\management\resources\database\postgresql\views**.

Si la consulta de la operación es dinámica (vea Anotación ReportOperationClass) los nombres de los parámetros de la operación deben ser nombres de columnas de la vista que utiliza el informe. Si la consulta de la operación es estática, los nombres de los parámetros de la operación deben ser nombres de parámetros definidos en el informe.

Por otra parte, para personalizar la apariencia de los informes, se pueden modificar las hojas de estilo y los archivos de imágenes generados que se encuentran en los siguientes subdirectorios:

- source\management\resources\reporting\jasper\reports\custom-made
- source\management\resources\reporting\jasper\reports\custom-made\subreports\resources
- source\management\resources\reporting\jasper\reports\entity-list
- source\management\resources\reporting\jasper\reports\entity-list\subreports\resources

Implementación de reglas de negocio mediante disparadores de base de datos

La manera más sencilla, dinámica y eficiente de implementar las reglas de negocio, que no pueden ser definidas en el modelo, es mediante disparadores (*triggers*) y funciones de PostgreSQL. Para las operaciones **insert**, **update** y **delete** sobre las tablas que corresponden a las entidades del modelo se generan los disparadores **before row** y **after row**, y sus respectivas funciones; estas funciones ejecutan otras funciones que implementan las reglas definidas en el modelo, tales como asignaciones de valores por omisión, conversiones y restricciones **Check**. Todos los disparadores se generan en el archivo **create-trigger.sql** y todas las funciones se generan en el archivo **create-trigger-function.sql**, ambos ubicados en el subdirectorio **source\management\sql\postgresql** del directorio raíz de la aplicación.

Los nombres de los disparadores y de sus respectivas funciones tienen la forma ***tabla*$*tipo*_*operacion*_row**, en donde ***tabla*** corresponde al nombre de la tabla que corresponde a la entidad; ***tipo*** corresponde al tipo de disparador (**before** o **after**); y ***operacion*** corresponde al nombre SQL de la operación (**insert**, **update** o **delete**).

Continuando con el ejemplo de la entidad Persona, a continuación se muestran sus disparadores y las respectivas funciones.

```
drop trigger if exists persona$before_insert_row on persona cascade;
create trigger persona$before_insert_row before insert on persona
    for each row execute procedure persona$before_insert_row();

drop trigger if exists persona$before_update_row on persona cascade;
create trigger persona$before_update_row before update on persona
    for each row execute procedure persona$before_update_row();

drop trigger if exists persona$before_delete_row on persona cascade;
create trigger persona$before_delete_row before delete on persona
    for each row execute procedure persona$before_delete_row();

drop trigger if exists persona$after_insert_row on persona cascade;
create trigger persona$after_insert_row after insert on persona
    for each row execute procedure persona$after_insert_row();
```

```
drop trigger if exists persona$after_update_row on persona cascade;
create trigger persona$after_update_row after update on persona
    for each row execute procedure persona$after_update_row();

drop trigger if exists persona$after_delete_row on persona cascade;
create trigger persona$after_delete_row after delete on persona
    for each row execute procedure persona$after_delete_row();

create or replace function persona$before_insert_row() returns trigger
as $$
begin
    raise notice 'persona$before_insert_row()';
--  new = persona$before_value(new);
    new = persona$default_value(new);
--  new = persona$after_value(new);
    new = persona$convert(new);
--  perform persona$before_check(new);
    perform persona$check(new);
--  perform persona$after_check(new);
    return new;
end;
$$ language plpgsql;

create or replace function persona$before_update_row() returns trigger
as $$
begin
    raise notice 'persona$before_update_row()';
--  new = persona$before_value(new, old);
    new = persona$default_value(new, old);
--  new = persona$after_value(new, old);
    new = persona$convert(new);
--  perform persona$before_check(new, old);
    perform persona$check(new);
--  perform persona$after_check(new, old);
    return new;
end;
$$ language plpgsql;

create or replace function persona$before_delete_row() returns trigger
as $$
begin
    raise notice 'persona$before_delete_row()';
    return old;
end;
$$ language plpgsql;

create or replace function persona$after_insert_row() returns trigger
as $$
begin
    raise notice 'persona$after_insert_row()';
    return null;
end;
$$ language plpgsql;
```

```
create or replace function persona$after_update_row() returns trigger
as $$
begin
    raise notice 'persona$after_update_row()';
    return null;
end;
$$ language plpgsql;

create or replace function persona$after_delete_row() returns trigger
as $$
begin
    raise notice 'persona$after_delete_row()';
    perform persona$delete$job(old.id);
    return null;
end;
$$ language plpgsql;
```

Existen al menos dos maneras de implementar reglas de negocio mediante disparadores de base de datos sin modificar los archivos generados (**create-trigger.sql** y **create-trigger-function.sql**). La primera es definir disparadores y funciones adicionales para las tablas que corresponden a las entidades. PostgreSQL permite definir varios disparadores para un mismo evento de una misma tabla, y los ejecuta ordenados alfabéticamente por nombre. Para definir disparadores y funciones adicionales para una entidad, se deben seguir los siguientes pasos:

- Crear un archivo con extensión **sql**, preferiblemente con el nombre de la entidad, en el subdirectorio **source\management\resources\database\postgresql\triggers\custom-made**.
- Codificar la definición de los disparadores de la tabla que corresponde a la entidad en el archivo creado en el paso anterior.
- Crear un subdirectorio, preferiblemente con el nombre de la entidad, en el subdirectorio **source\management\resources\database\postgresql\functions\custom-made**.
- Crear archivos con extensión **sql**, preferiblemente con el nombre del disparador, para cada disparador de la tabla que corresponde a la entidad, en el subdirectorio creado en el paso anterior.
- Codificar las respectivas funciones en los archivos correspondientes a los disparadores, creados en el paso anterior.

La segunda manera de implementar reglas de negocio mediante disparadores es codificar funciones para complementar las funciones generadas; pero, en esta versión, esto está limitado a las funciones **before_insert_row** y **before_update_row**. Para cada función **before_insert_row** y **before_update_row** se pueden codificar las siguientes funciones complementarias:

- before_value
- after_value
- before_check
- after_check

En el ejemplo anterior se puede ver el orden en el que son ejecutadas estas funciones. Cabe resaltar que las funciones complementarias que se ejecutan en **before_insert_row** y **before_update_row** tienen el mismo nombre pero diferente lista de parámetros: **(new)** para **before_insert_row** y **(new, old)** para **before_update_row**, ya que la variable **old** invariablemente tiene el valor NULL en disparadores **before_insert_row**. De modo que no son 4 sino 8 funciones complementarias; esto permite codificar reglas diferentes para cada operación. Para definir una función complementaria para una entidad, se deben seguir los siguientes pasos:

- Crear un subdirectorio, preferiblemente con el nombre de la entidad, en el subdirectorio **source\management\resources\database\postgresql\functions\custom-made**.
- Crear un archivo con extensión **sql**, preferiblemente con el nombre corto de la función complementaria (**before_value**, **after_value**, **before_check**, **after_check**), en el subdirectorio correspondiente a la entidad, creado en el paso anterior.
- Codificar las dos funciones (mismo nombre pero diferente lista de parámetros) en el archivo creado en el paso anterior.
- Habilitar la ejecución de las funciones complementarias, decorando la meta-entidad con la anotación EntityTriggers. El valor predeterminado de los atributos de esa anotación es FALSE, lo que significa que no se debe ejecutar ninguna de las funciones complementarias. En el ejemplo anterior se puede ver que las líneas que ejecutan las funciones complementarias son comentarios. Esos comentarios se convierten en instrucciones cuando el valor del correspondiente elemento de la anotación es TRUE.

Refinación de la interfaz de usuario Web

La mayor parte de la funcionalidad de las páginas web generadas por la plataforma **jee1** se delega en tres de los objetos que las páginas instancian al ejecutar. Estos son: el gestor, el asistente y el proveedor de datos. Para refinar las páginas de la interfaz Web pueden extenderse las clases que utilizan las páginas para instanciar estos objetos.

No todas las páginas instancian los tres objetos. Los objetos son instanciados dependiendo del tipo de página, como se muestra en la siguiente tabla.

Tipo de página	Gestor	Asistente	Proveedor de datos
Tablas de consulta y/o registro	Sí	Sí	Sí
Detalles de consulta y/o registro	Sí	Sí	Sí
Árboles de consulta y/o registro	Sí	No	Sí
Consolas de procesamiento	Sí	No	No

La plataforma **jee1** incluye cuatro clases de gestores, una para cada uno de los tipos de página, como se muestra en la siguiente tabla.

Tipo de página	Clase del Gestor
Tablas de consulta y/o registro	GestorPaginaActualizacionConTabla
Detalles de consulta y/o registro	GestorPaginaActualizacion
Árboles de consulta y/o registro	GestorPaginaActualizacionConArbol
Consolas de procesamiento	GestorPaginaBasica

Todas las clases de gestores se generan en el paquete ***alias*.lib.core.web.app** de la librería de clases estándar de la aplicación, donde ***alias*** corresponde al alias del proyecto generado. Los gestores proveen la funcionalidad común a páginas de un mismo tipo.

Para cada entidad definida en el modelo se genera una clase de asistente en el paquete ***alias*.web.commons.assistants** del módulo Web de la aplicación. Los nombres de las clases tienen la forma **AsistentePaginaActualizacion*Entidad***, donde ***Entidad*** corresponde al nombre de la entidad que genera la clase. Los asistentes proveen la funcionalidad común a las tablas y detalles de consulta y/o registro de la entidad.

Para cada entidad definida en el modelo también se genera una clase de proveedor de datos en el paquete ***alias*.lib.data.provider.xdp2** de la librería de clases estándar de la aplicación. Los nombres de las clases tienen la forma ***Entidad*CachedRowSetDataProvider2**, donde ***Entidad*** corresponde al nombre de la entidad que genera la clase. Los proveedores de datos proveen la funcionalidad que necesitan las páginas para interactuar con el gestor de base de datos. Todas las páginas de consulta y/o registro de la entidad utilizan, al menos, la clase de proveedor de datos de

la entidad. Si la página es de formato Maestro/Detalle, entonces también utiliza la clase de proveedor de datos de la entidad correspondiente al maestro.

El mecanismo de instanciación para todos estos objetos (gestores, asistentes y proveedores de datos) permite detectar dinámicamente si una página debe utilizar la clase generada o una extensión de la clase generada. Para que este mecanismo utilice una extensión de la clase generada, se deben cumplir los siguientes requisitos:

- El paquete de la extensión debe ser ***paquete*.ext**, donde ***paquete*** corresponde al nombre del paquete de la clase extendida. Tal como se mencionó al inicio de este apéndice, los paquetes adicionales deben crearse en el subdirectorio **src\java\code** de la librería de clases estándar o del módulo Web, según corresponda.
- El nombre de la extensión debe ser ***Clase*X**, donde ***Clase*** corresponde al nombre de la clase extendida.

Las siguientes tablas muestran el paquete y el nombre de clases extendidas.

Objeto	Paquete de la extensión
Gestor	***alias***.lib.core.web.app.ext
Asistente	***alias***.web.commons.assistants.ext
Proveedor de datos	***alias***.lib.data.provider.xdp2.ext

Nombre de la clase extendida	Nombre de la extensión
GestorPaginaActualizacionConTabla	GestorPaginaActualizacionConTablaX
GestorPaginaActualizacion	GestorPaginaActualizacionX
GestorPaginaActualizacionConArbol	GestorPaginaActualizacionConArbolX
GestorPaginaBasica	GestorPaginaBasicaX

Continuando con el ejemplo de la entidad Persona, la siguiente tabla muestra el paquete y el nombre de las extensiones del asistente y del proveedor de datos de esa entidad.

Objeto	Nombre de la extensión
Asistente	AsistentePaginaActualizacionPersonaX
Proveedor de datos	PersonaCachedRowSetDataProvider2X

Por otra parte, para personalizar la apariencia de las páginas, se pueden modificar las hojas de estilo generadas; estas se encuentran en el subdirectorio **web\resources** del módulo Web de la aplicación. También se pueden modificar los archivos de imágenes que se encuentran en ese mismo subdirectorio.

Finalmente, para que la aplicación disponga de información de ayuda en línea, se puede modificar el archivo **ayuda.html** que se encuentra en el subdirectorio **web** del módulo Web de la aplicación.

Internacionalización de la aplicación

La plataforma **jee1** genera archivos de recursos por defecto, también conocidos como **archivos base** a partir del modelo de la aplicación. En caso de que un archivo de recursos para el idioma seleccionado por el usuario no esté disponible, la interfaz de la aplicación utiliza el archivo base para obtener los recursos necesarios.

Los archivos base son generados en el paquete ***alias*.lib.base.bundle.crop** del subdirectorio **src\crop** de la librería de recursos de la aplicación, donde ***alias*** corresponde al alias del proyecto generado (vea Método setAlias). Los archivos generados son:

- BundleDominios.properties

- BundleEnums.properties
- BundleFunciones.properties
- BundleMensajes.properties
- BundleMenus.properties
- BundleParametros.properties
- BundleWebui.properties

Los archivos de recursos son manejados con objetos de la clase ResourceBundle de Java, de manera que el nombre de los archivos debe cumplir con las reglas de nomenclatura correspondientes. Los archivos de recursos generados son archivos de propiedades y, por lo tanto, tienen extensión **properties**.

Para agregar un idioma, se deben seguir los siguientes pasos:

- Copiar los archivos base al paquete ***alias*.lib.base.bundle.code** del subdirectorio **src\code** de la librería de recursos de la aplicación.
- Renombrar los archivos copiados en el punto anterior, colocándoles el sufijo correspondiente al idioma que se desea agregar. El sufijo debe tener al menos el código del idioma y, opcionalmente, el código de país y la variante del archivo; por ejemplo, el sufijo **_en_GB** debe utilizarse para los archivos correspondientes al inglés (en) de Gran Bretaña (GB).
- Traducir los valores de las propiedades de los archivos renombrados en el punto anterior.

La interfaz de la aplicación primero busca los archivos de recursos en el paquete ***alias*.lib.base.bundle.code**, luego en el paquete ***alias*.lib.base.bundle.copy**, y por último en el paquete ***alias*.lib.base.bundle.crop**. Por lo tanto, siguiendo un procedimiento similar al anterior, también es posible personalizar los archivos base generados, siguiendo los siguientes pasos:

- Copiar los archivos base que se desea personalizar al paquete ***alias*.lib.base.bundle.code** del subdirectorio **src\code** de la librería de recursos de la aplicación.
- Modificar los valores de las propiedades de los archivos copiados en el punto anterior.

Además de los archivos base, la plataforma **jee1** también genera archivos de recursos para el idioma inglés. Estos archivos contienen la traducción al inglés de las propiedades de los módulos predefinidos de **jee1**; solo hace falta agregarles la traducción de las propiedades específicas de la aplicación. Estos archivos se generan en el subdirectorio **src\i18n\en*alias*\lib\base\bundle\crop** de la librería de recursos y tienen el sufijo **_en**.

Para información complementaria sobre la internacionalización de aplicaciones, consulte la documentación de Java (el capítulo Internationalization de The Java Tutorials es un buen punto de partida).

Refinación de las plantillas para generar la aplicación

A diferencia de la mayoría de los generadores de código actualmente disponibles en el mercado, Adalid ha sido diseñado para permitirle construir sus propios generadores, dándole total control sobre el proceso de generación y los documentos producidos. Además, también es posible personalizar las plantillas utilizadas para generar la aplicación de manera individual para cada proyecto **jee1**. Adalid primero busca las plantillas en el subdirectorio **resources\velocity\templates** del proyecto **jee1** que se genera y luego en el subdirectorio **source\development\resources\velocity\templates** de Adalid. Por lo tanto, para personalizar una plantilla se deben seguir los siguientes pasos:

- Copiar la plantilla que se desea personalizar al directorio **resources\velocity\templates** del proyecto **jee1**. Al copiar la plantilla, se debe copiar la estructura de directorio a partir de **source\development\resources\velocity\templates**. Por ejemplo, para personalizar la

plantilla **source\development\resources\velocity\templates\database\postgresql\drop-everything-else.vm**, ésta se debe copiar al subdirectorio **resources\velocity\templates\database\postgresql** del proyecto **jee1**.

- Modificar la plantilla copiada en el paso anterior.

Adalid utiliza *Apache Velocity Engine* como motor de plantillas, de manera que las plantillas deben ser elaboradas utilizando el lenguaje de plantillas de Velocity. Para más información sobre Velocity, consulte la página The Apache Velocity Project.