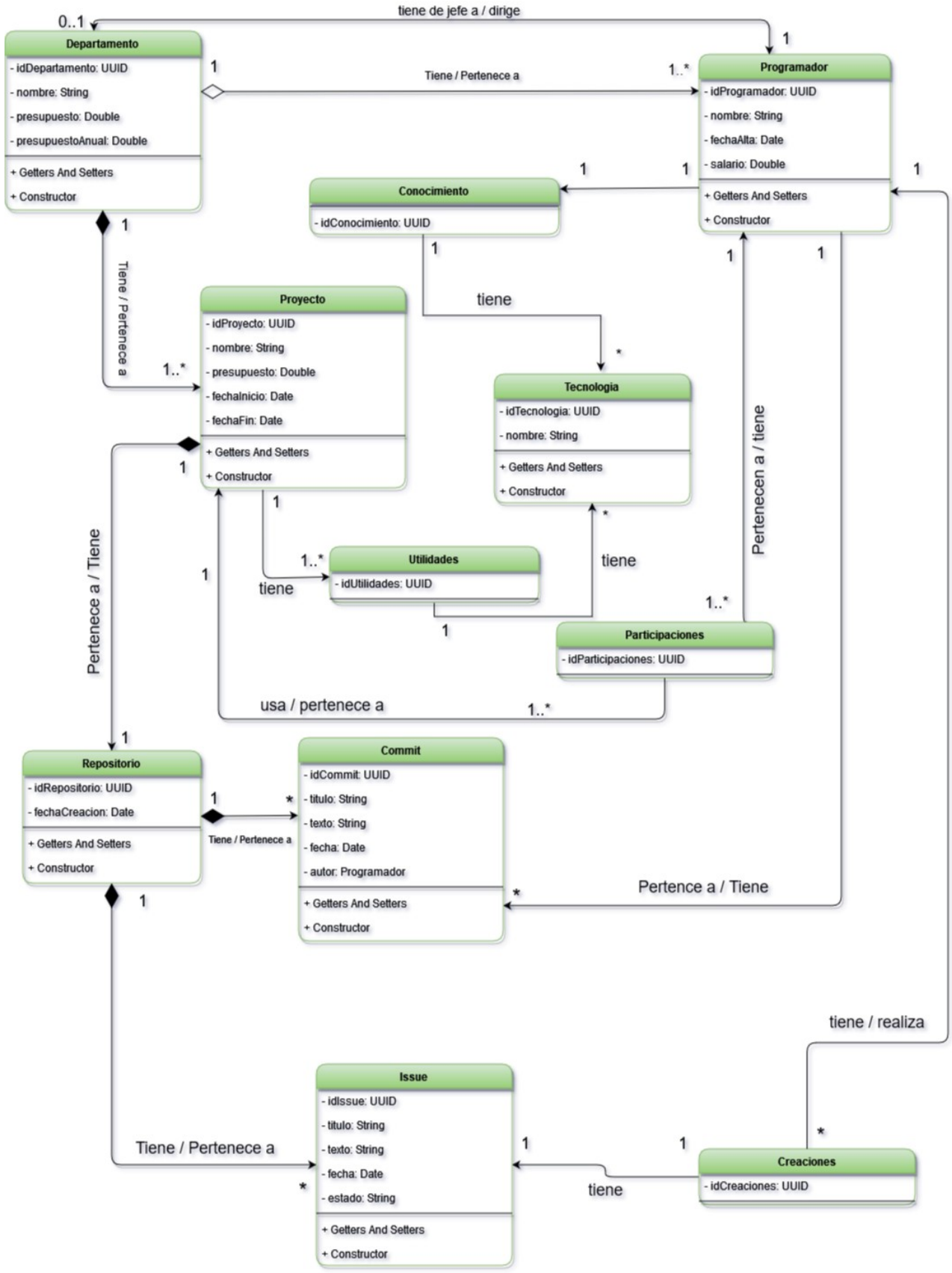
PRÁCTICA 3:
ACCESS DATA SL

ACCESO A DATOS

DYLAN HURTADO LÓPEZ Y JAVIER GONZÁLEZ MUÑOZ

Índice:

- Diagrama de clases y justificación



○ Relaciones:

■ Programador – Departamento:

Cada **Programador** tiene asociado un **departamento**.

Y cada **departamento** tiene varios **trabajadores**(**Programadores**),

un **jefe** de **departamento** que es un **Programador**,

el **historialJefes** son todos los programadores que han sido jefe de ese departamento.

Esta relación tiene una agregación de **Departamento** a **Programador**.

En cuanto a la **cardinalidad**:

Un **Programador** tiene un **departamento** y un **departamento** tiene muchos **programadores**.(**Uno a muchos**).

Un **departamento** tiene 0 o 1 jefe y el jefe(**Programador**) tiene un **departamento**.(**Uno a Uno**).

Programador – Tecnología:

Cada **Programador** tiene asociado unas **Tecnologías.**

Y cada **Tecnología** tiene varios **Programadores**,

Como es una relación **muchos a muchos hay que romperla con una tabla/clase** intermedia en este caso **Conocimiento.**

Un **programador** tiene un **conocimiento** y un **conocimiento** tiene un **programador (Uno a uno).**

Varias Tecnologías pertenece a un **conocimiento**, un **conocimiento** tiene varias **tecnologías (Muchos a uno)**

Programador – Proyecto:

Cada **Programador** tiene asociado unos **Proyectos.**

Y cada **Proyecto** tiene varios **Programadores**,

Como es una relación **muchos a muchos** hay que romperla con una tabla/clase intermedia en este caso **Participaciones.**

Un **programador** tiene unas **participaciones** y una **participación** tiene un **programador (Uno a muchos).**

Un proyecto tiene a unas **participaciones**, unas **participaciones** tiene un **proyecto (Uno a muchos).**

Programador – Commit:

Cada **Programador** tiene asociado unos **Commits.**

Y cada **Commit** tiene un **Programador**,

Es una relación **muchos a uno y no es bidireccional** porque considero que cada commit es particular de cada programador y la empresa**.**

Programador – Issue:

Cada **Programador** tiene asociado unos **Issues.**

Y cada **Issue** tiene varios **Programadores**,

Como es una relación **muchos a muchos** hay que romperla con una tabla/clase intermedia en este caso **Creaciones.**

Un **programador** tiene unas **creaciones** y unas **creaciones** tiene un **programador (Muchos a uno).**

Un Issue tiene a una **creación**, unas **creaciones** tiene un **Issue (Uno a Uno).**

Departamento – Proyecto:

Cada **Departamento** tiene asociado unos **Proyectos.**

Y cada **Proyecto** tiene asociado un **Departamento**,

Por lo tanto es una relación **muchos a uno.** Además existe una **composición** de **Departamento** a **proyecto**

Proyecto – Repositorio:

Cada **Proyecto** tiene asociado un **Repositorio.**

Y cada **Repositorio** tiene asociado un **Proyecto**,

Por lo tanto es una relación **uno a uno.** Además existe una **composición** de **proyecto** a **repositorio.**

Proyecto – Tecnología:

Cada **Proyecto** tiene asociado unas **Tecnologías.**

Y cada **Tecnología** esta asociada en unos **Proyectos**,

Es una relación **muchos a muchos** por lo tanto hay que romperla con una tabla/clase intermedia en este caso **Utilidades.**

Un **Proyecto** tiene unas **Utilidades** y unas **Utilidades** tiene un **Proyecto (Muchos a uno).**

Una Tecnología tiene a unas **Utilidades**, unas **Utilidades** tiene varias **Tecnología (Uno a muchos).**

Repositorio – Commit:

Cada **Repositorio** tiene asociado unos **Commits.**

Y cada **Commit** tiene asociado un **Repositorio**,

Por lo tanto es una relación muchos a uno. Además existe una **composición** de **Repositorio** a **Commit.**

Repositorio – Issues:

Cada **Repositorio** tiene asociado unos **Issues.**

Y cada **Issue** tiene asociado un **Repositorio**,

Por lo tanto es una relación muchos a uno. Además existe una **composición** de **Repositorio** a **Issue.**

Implementación del sistema y relación Mapeo Objeto Relacional de forma manual + implementación operaciones CRUD.

- **Clases por paquetes**
    - App - Contiene las llamadas a los métodos principales de funcionamiento. Asignados en la clase Facade. Podemos ver las propiedades asignadas en “application.properties” de resources.
    - Facade - Oculta los métodos de funcionamiento para limpiar el Main (App). Sigue una estructura de patrón “Fachada”. Ordena los métodos de inicio de la base de datos y contiene las funciones de uso para generar JSON y XML.
    - Model - Clases POJO de las entidades y tablas principales como intermedias.Commit, Conocimiento, Creaciones, Departamento, Issue, Participaciones, Programador, Proyecto, Repositorio, Tecnología, Utilidades.
    - Service - Cuenta con una clase Base Service de la que heredan todas las demás clases (clase de cada entidad) como commits, programadores, departamentos… En cada una de ellas se tienen los métodos que devuelven “mapper.toDTO” dependiendo del funcionamiento del método. Mandamos información al mapper para que lo use. Es la relación intermedia entre repositorio y mapper.

    - Repository - El repositorio se comunica con la base de datos. Partiendo de CrudRepository como interfaz, el resto de clases creadas para cada entidad y tablas intermedias debe cumplir los requisitos implementados por la interfaz. Nuevamente se crea una clase para cada una llamada “Repo_____” (commit, departamento, programador…)

    - Mapper - Cuenta con una clase BaseMapper de la que hereda el resto de clases del paquete. Trata de trasformar la información de cada campo (commits, programadores,issues...) y tratarlos como items para en un futuro con ayuda de la creación de dto, poder exponer la información en un formato como puede ser xml o json.

- DTO - Cuenta con una clase para cada entidad/tabla y trata de transmitir la información recibida por una entrada dada a una salida y así crear un formato organizado de datos gracias a su patrón. Cada clase implementada "_____DTO" se encarga de la información del tipo que indica su nombre.

- DatabaseController - Se encarga de los atributos requeridos para la conexión y el control de la base de datos, como puede ser el puerto, la url, el usuario, la contraseña… Obligando a que no sean nulos. En ella se inicia la configuración y se añaden métodos de control sobe la base de datos.

- Controller - Clase encargada de generar los métodos y crear la instancia del controlador. Hace de conexión entre los datos en bruto y el paso final a lo que el usuario pide por tener el "control". Creamos para cada tabla métodos de traducción a JSON y XML.

- **Métodos** (Métodos individualmente comentados en JDOC)
  - ApplicationProperties - readProperty() se usa en el Main para leer propiedades al igual que para cargarlas.
  - Facade
    - Base de datos - getInstance() crea una instancia siguiendo un patrón Singleton, checkService() comprueba si el servicio está activo, initDatabase() inicia la base de datos introduciendole "sqlFile" para cargarlo.

    - Usuario - selecJsonOrXml() hace decidir al usuario que formato mostrar, xml o json.

    - XML - salidaXML() da TODAS las operaciones CRUD y salidas en formato xml. Tras ello creamos los métodos que a su vez tienen los métodos del controller para realizar TODAS las consultas. Además de las operaciones del ejercicio 5
    - JSON - salidaJSON() da TODAS las operaciones CRUD y salidas en formato JSON. Tras ello creamos los métodos

que a su vez tienen los métodos del controller para realizar TODAS las consultas. Además de las operaciones del ejercicio 5.

- Service - Todas las entidades cuentan con métodos que hacen de conexión entre repositorio y mapper. En ellas encontramos prácticamente los mismos métodos para todas las entidades, a diferencia de los que hemos creado para mandar a mapper información más específica. Encontramos los métodos getAll(), getAllById(), post(), update(), delete()... pero creados de manera personalizada para cada entidad en su clase.

- Repository - Los métodos son los mismos mencionados en service ya que repositorio se encarga de estar en contacto directo con la base de datos y traer o poner dicha información en juego para ser manipulada. Los métodos internamente realizan la consulta y toman,guardan,borran o actualizan los datos.

- Mapper - Cuenta con métodos similares para cada clase, (siempre en base a BaseMapper). El primero es fromDTO() y el segundo es toDTO(). Como su nombre indica, uno coge información del DTO y otro manda información a los DTO. Así en un futuro podremos manipular estos métodos para convertir la información a un formato XML o JSON

- DTO- Al igual que mapper, cuenta con dos metodos principales. El primero siendo fromJSON() y el segundo toJSON(). Pues utilizamos estos métodos nuevamente para transformar la información “desdeJSON” hacia otro lugar o “haciaJSON” desde otro lugar como su traducción indica.

- DataBaseController - Los métodos son puramente de control sobre la base de datos. Unos se encargan de cargar la configuración. Otros de cargar los datos y el resto de realizar operaciones como el borrado, la actualización o la inserción de parámetros.

- Controller - En cada clase hacemos referencia a los datos de una tabla. Esta tabla puede ser leída en XML o JSON. Los métodos creados se encargan básicamente de eso. Coger todos, guardar, actualizar, borrar, coger por id… Implementados de manera doble para salir en xml o json a gusto del usuario. Son los que finalmente se le pasarán al usuario a través de la fachada.

- **Patrones**
  - Singleton - Hemos utilizado más de una vez este patrón para asegurarnos de que una clase tenga una única instancia a la vez a pesar de que esta proporcione un punto de acceso global.

  - Facade (fachada) - El propósito principal del uso de este patrón es organizar la información de cara a uso de la misma en el Main (app). Desde esta hacemos una especie de interfaz o biblioteca con todos los métodos finales del acceso a los datos para mostrarlos al usuario.

Implementación del sistema y relación Mapeo Objeto Relacional usando JPA con Hibenate + implementación operaciones CRUD.

Explicación basada en el apartado anterior y haciendo hincapié en las diferencias que supone el uso de JPA con Hibernate.

- **Clases** (diferencias principales con la manera manual)
    - DAO- se transforman las antiguas clases “modelo” en nuevas clases en las que utilizamos anotaciones para crear tablas @table y posteriormente columnas @column definiendo para cada una de ellas sus atributos. Utilizando @NamedQuery podemos hacer una consulta en la propia clase.

    - Manager - En la clase manager utilizamos singleton. Posteriormente creamos dos métodos de control. Uno para abrir el cupo de consultas en la base de datos y otro para cerrar el flujo de consultas.

    - Repositorio - En las clases repositorio utilizamos métodos para la obtención de datos. Primero nos apoyamos en el manager para acceder a la clase pertinente “DAO” una vez allí seleccionar su consulta ya creada gracias a las etiquetas. A la hora de realizar operaciones en la base de datos (es decir, no de obtención de datos) hacemos uso de los métodos merge(),commit()...

- **Persistence.xml**

- Siempre creado en META-INF de resources.
- Le hemos pasado a este fichero las propiedades de la base de datos a la que se conecta, desde la url, pasando por el usuario y contraseña para acabar en su driver correspondiente. Además le pasamos las clases DAO. Tenemos la posibilidad de ajustar la manera en la que vemos las consultas y trabajamos con el esquema.