UD12.2.-

CAPA DE ACCESO A DATOS

Objeto de Acceso a Datos (DAO)

Programación – **1er DAW/DAM**

GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

CPFPBATOI

C/ Societat Unió Musical, 8 - 03802 Alcoi
Tel. 966 52 76 60 - Fax 966 52 76 61
03012165.secret@gva.es
www.cipfpbatoi.es

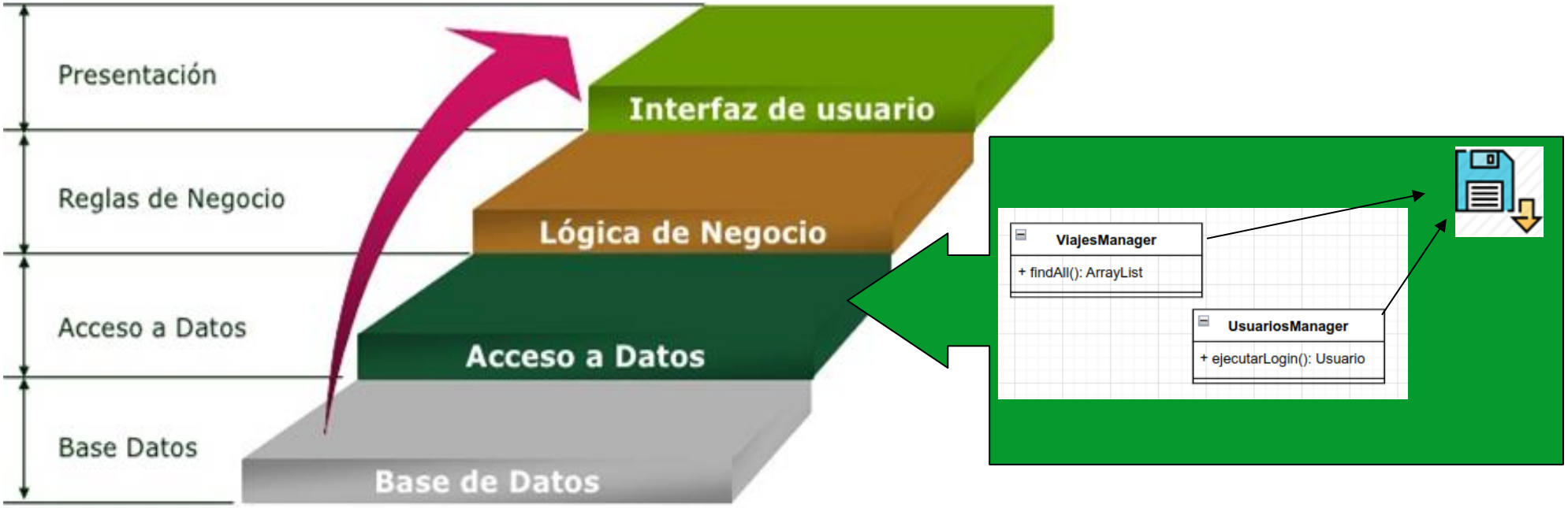
GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

UNIÓ EUROPEA

0. CONTENIDOS

- Introducción
- Capa de acceso a datos
  - Objeto de Acceso a Datos (DAO)
  - Objeto de Transferencia de Datos (DTO)
  - Ejemplos
- Actividades Previas

1. Introducción



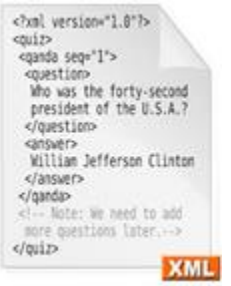
1. Introducción

Capa de acceso a datos

- Se trata de una capa que se va a encargar de las **tareas relacionadas con el acceso a la información** que tenga nuestra aplicación, ya sea para su **consulta** o para su **manipulación**.

- Podrá permitir el acceso a la información almacenada en **diferentes fuentes de datos :**

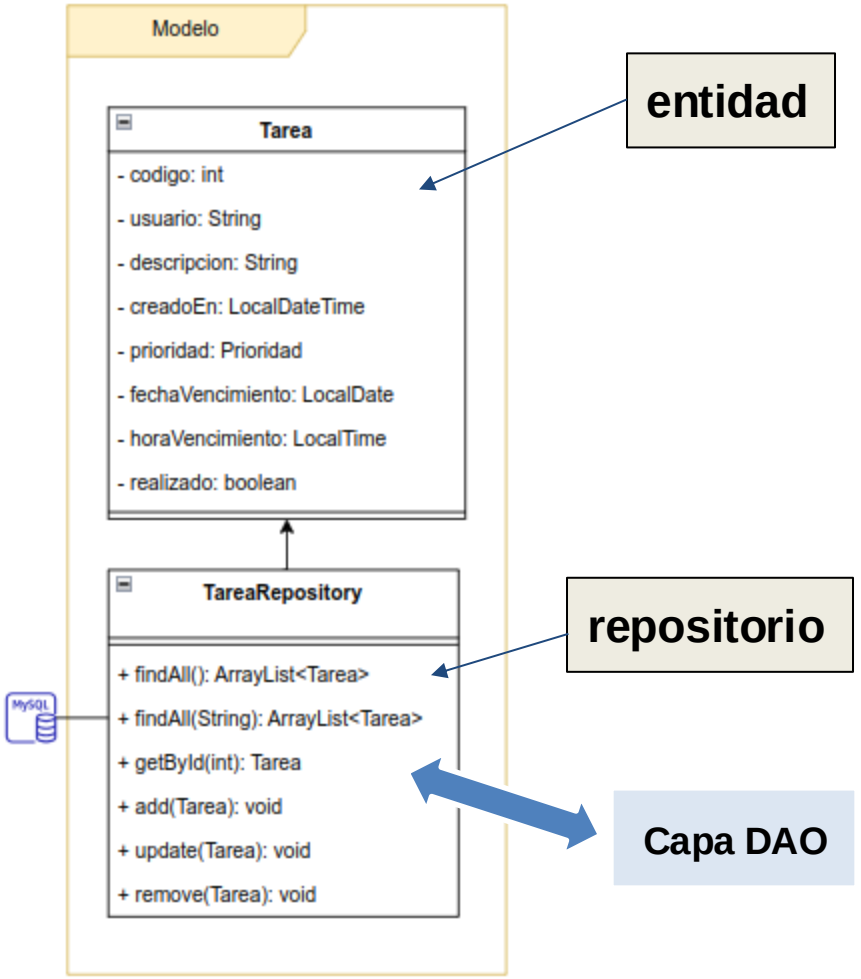
  - **Bases de datos** (Mysql/Mariadb, Postgresql, ...)
  - **Ficheros de texto o binarios**
  - **Documentos de intercambio de información** (xml, json, ...)



2. Capa de Acceso a Datos

La capa de acceso a datos (capa DAO) forma parte **del modelo** de nuestra aplicación.

- Recordemos que el modelo está compuesto por el conjunto de **entidades** y de **repositorios** de datos. Ahora añadimos también esta capa (experta en la fuente de datos elegida).
- En el momento en que la información necesita persistir o ser recuperada: el **repositorio** usará las entidades para encapsular la información (en **objetos de transferencia de datos**) para pasárselas a la capa **DAO** y viceversa.



2.1 Objeto de Transferencia de Datos (DTO)

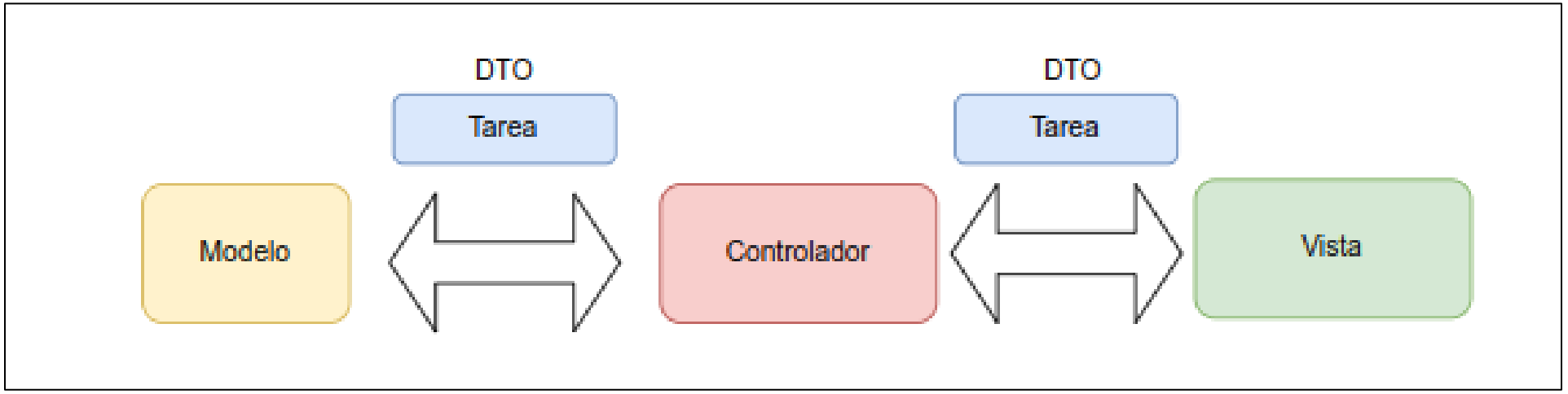
- Del inglés **Data Transfer Object** (DTO).
- Son los objetos que nacen de nuestras **entidades**.
- Un DTO es una clase simple que representa los datos con los que trabaja la aplicación. Como su nombre indica, se usará para **transferir datos** entre diferentes capas.

```
public class Tarea {

    private int codigo;

    private String descripcion;

    ....

    private boolean realizada;

    private int idCategoria;

    public Tarea(int id, String descripcion) {
        this.id = id;
        this.descripcion = descripcion;
....
        this.finalizada = false;
    }
     ...
}
```

Nota.- Para una mejor comprensión, vamos a **obviar** el campo `categoria` de las tareas según la tabla `tareas` de nuestra base de datos `tareas_db`.

El DTO no contendrá esta información de momento (lo trataremos en la próxima y última presentación)

2.1 Objeto de Transferencia de Datos (DTO)



Permite el intercambio de estos datos entre las diferentes capas.

2.2 Objeto de Acceso a Datos (DAO)

- Del inglés **Data Access Object** (DAO)
- Con la capa DAO y sus **Objetos de Acceso a Datos (DAO)** tenemos un nivel adicional de abstracción -> encargado de 'pelearse' directamente con los datos de la fuente de datos elegida.
- **Los DAO están ligados al sistema de almacenamiento** que se utilice (ficheros, bases de datos relacionales, …)
  - Cada uno de los objetos DAO tendrá una "**relación directa (1 <--> 1)**" con un fichero o **tabla de la base de datos**.
  - Permiten **persistir** la información que reciban de un DTO así como **recuperarla** de la fuente de datos y devolverla también con un DTO.
  - A los DAO únicamente se tendrá **acceso a través del Repositorio**.

2.2 Objeto de Acceso a Datos (DAO)

SQLTareaDAO

+ findAll(): ArrayList<Tarea>
+ findAll(String): ArrayList<Tarea>
+ getById(id): Tarea
+ add(Tarea): void
+ udpate(Tarea): void
+ remove(id: int): void

La información se obtiene/escribe de/en una base de datos

La información se obtiene/escribe de/en un fichero

InMemoryTareaDAO

+ findAll(): ArrayList<Tarea>
+ findAll(String): ArrayList<Tarea>
+ getById(id): Tarea
+ add(Tarea): void
+ udpate(Tarea): void
+ remove(id: int): void

La información se obtiene/escribe de/en memoria (un ArrayList, por ejemplo). Tal y como lo estábamos gestionando hasta ahora.

FileTareaDAO

+ findAll(): ArrayList<Tarea>
+ findAll(String): ArrayList<Tarea>
+ getById(id): Tarea
+ add(Tarea): void
+ udpate(Tarea): void
+ remove(id: int): void

2.3 DAO sobre base de datos : Tabla *tareas*

Recuerda que, de momento, vamos a **obviar este dato**.

2.3 DAO sobre base de datos

```
@Repository
public class SQLTareaDAO implements TareaDAO {

   private static final String DATABASE_TABLE = "tareas";

   @Autowired
   private final MySqlConnection mySqlConnection;

   // Obtiene todas las tareas de la base de datos
   @Override
   public ArrayList<Tarea> findAll() {...}

   // Obtiene todas las que comiencen por el texto dado
   @Override
   public ArrayList<Tarea> findAll(String texto) {...}

   // Obtiene la tarea de id dado
   @Override
   public Tarea getById(int id) throws NotFoundException {...}

   // Inserta la tarea
   @Override
   public void add(Tarea tarea) {...}
}
```

Este **DAO** nos pemite trabajar **exclusivamente** con la base de datos que almacena las tareas

Se necesita la conexión a la BD, por tanto, inyectamos la dependencia de dicha conexión para poderla usar después.

2.4 Ejemplo. Obtención de una Tarea

```
@Override
public Tarea getById(int id) throws NotFoundException {
    String sql = String.format("SELECT * FROM %s WHERE codigo = ? ", DATABASE_TABLE);

    Connection connection = mySQLConnection.getConnection();

    try (PreparedStatement ps = connection.prepareStatement(sql))
    {
        ps.setInt(1, id);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            return mapToTarea(rs);
        }

        throw new NotFoundException("La tarea no existe");

    } catch (SQLException e) {
        throw new DatabaseErrorException(sql + e.getMessage());
    }
}
```

Creamos un objeto de `Tarea` a partir de los datos obtenidos en el `ResultSet` (ver detalle en siguiente diapositiva)

2.4 Ejemplo. Obtención de una Tarea

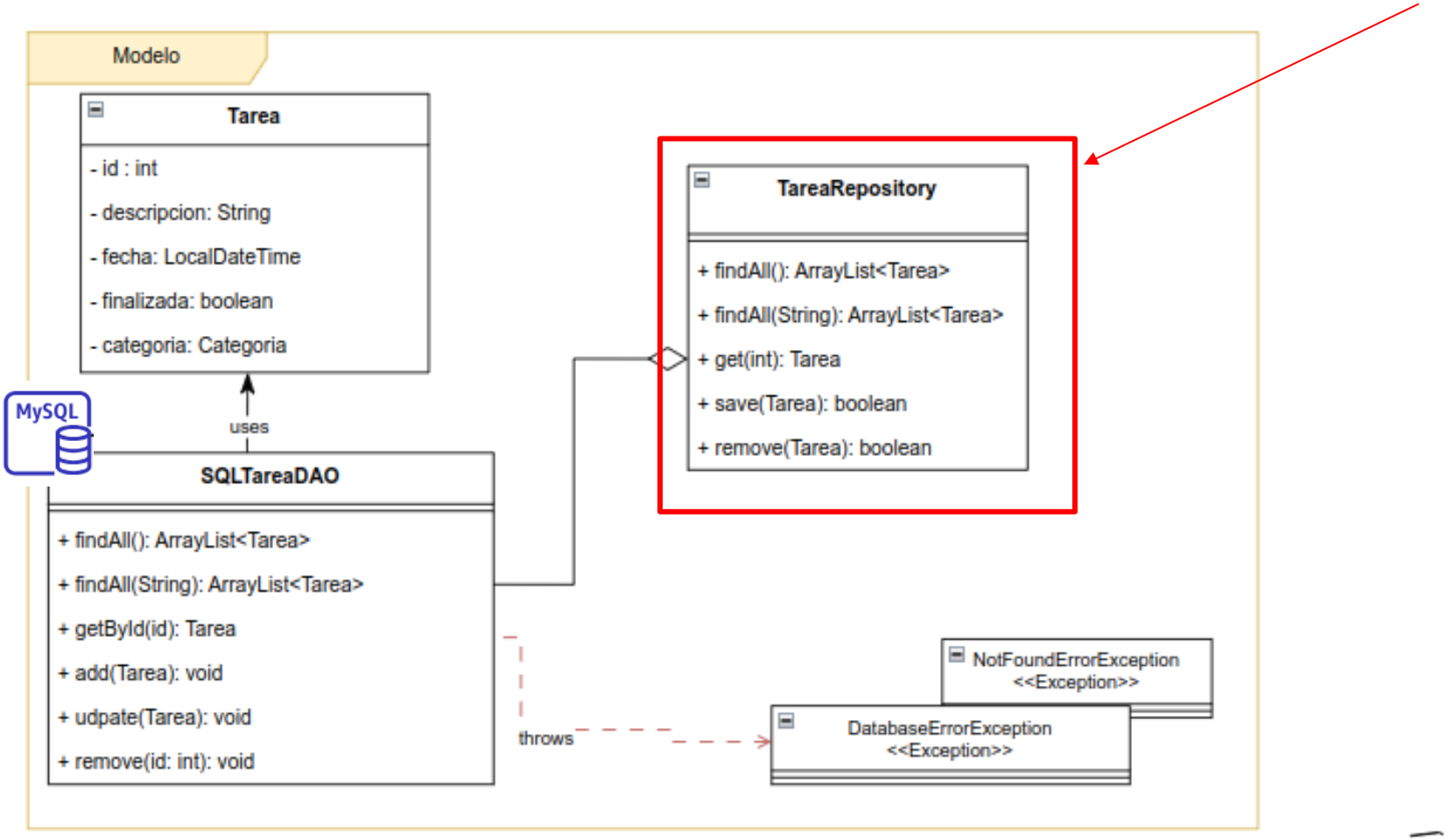
Mapeado de una tarea

Mapear = Leer campo a campo el *resultset* y asignar al atributo correspondiente de Tarea

```
private Tarea mapToTarea(ResultSet resultset) throws SQLException {
    int cod = resultset.getInt("codigo");
    String usuario = resultset.getString("usuario");
    String descripcion = resultset.getString("descripcion");
    LocalDateTime creadoEn = resultset.getTimestamp("fechaCreacion").toLocalDateTime();
    Prioridad priority = Prioridad.fromText(resultset.getString("prioridad"));
    LocalDateTime vencimiento = resultset.getTimestamp("vencimiento").toLocalDateTime();
    boolean realizada = resultset.getBoolean("realizada");

    return new Tarea(cod, usuario, descripcion, creadoEn, priority, vencimiento.toLocalDate(),
            vencimiento.toLocalTime(), realizada);
}
```

2.4 Ejemplo. Accediendo a los datos

Una vez **implementados los métodos del DAO** podemos acceder a la información **desde el repositorio**.



2.4 Ejemplo. Accediendo a los datos

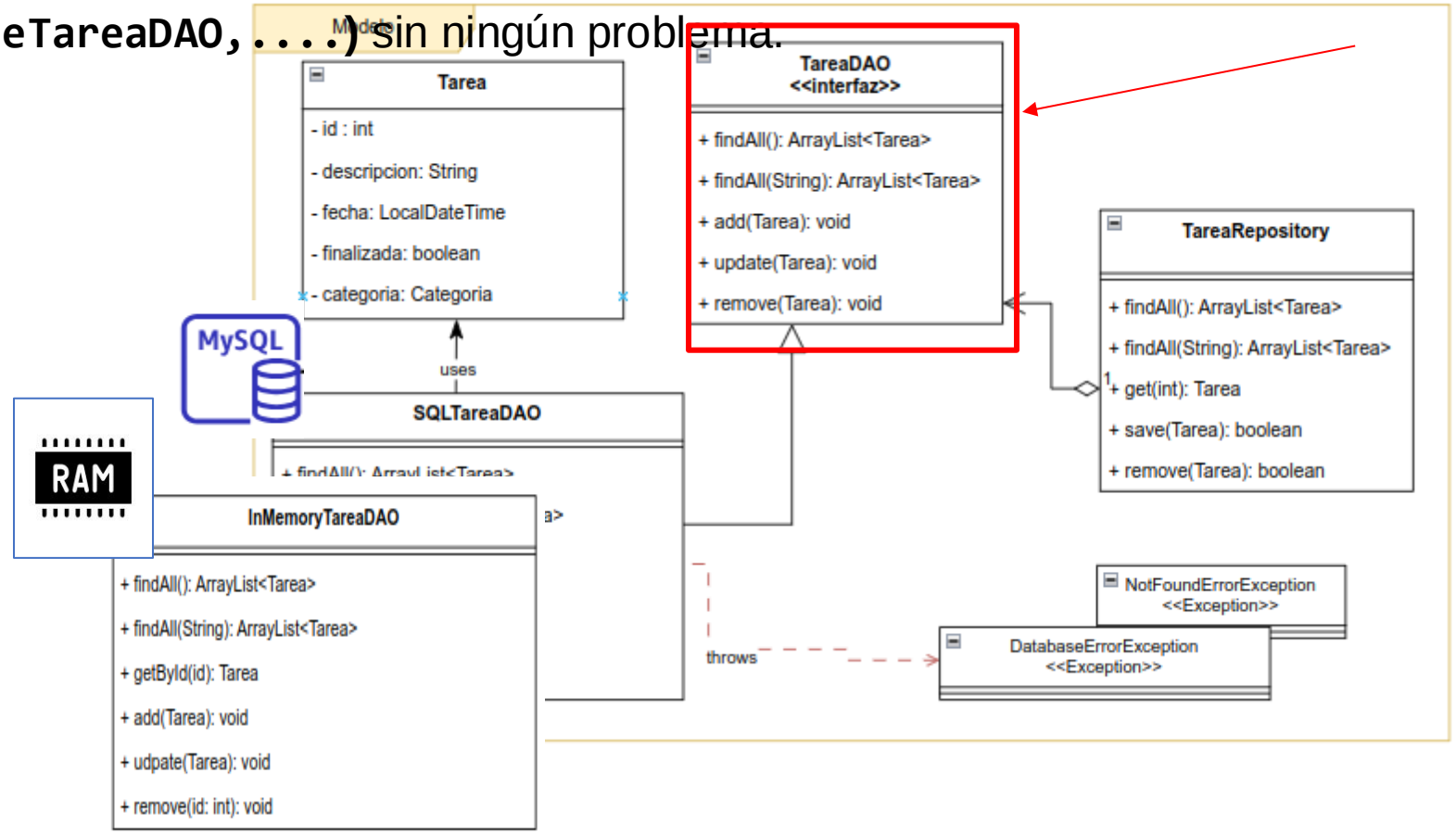
Una vez **implementados los métodos del DAO** podemos acceder a la información **desde el repositorio**.

```
@Repository
public class TareaRepository {

    private SQLTareaDAO sqlTareaDAO;

    @Autowired
    public TareaRepository(SQLTareaDAO sqlTareaDAO) {
        this.sqlTareaDAO = sqlTareaDAO;
    }

    /**
     * Obtiene la Tarea con codigo @codTarea. En caso de que no la encuentre
devolverá una excepción @NotFoundException
     * @param codigo
     */
    public Tarea get(int codigo) throws NotFoundException {
        return sqlTareaDAO.getById(codigo);
    }
...
```

Instanciará automáticamente el objeto `sqlTareaDAO`.

2.4 Ejemplo. Accediendo a los datos

Como último paso, es importante y totalmente recomendable declarar una **interfaz** que defina todos los **métodos** que debe tener un **DAO de Tarea**. De esta forma, podremos **intercambiar fácilmente el origen de los datos (`SQLTareaDAO`, `InMemoryTareaDAO`, `FileTareaDAO`,...)** sin ningún problema.



2.4 Ejemplo. Accediendo a los datos

Al **utilizar el inyector** de dependencias @Autowired **sólo deberemos** tener en cuenta el tipo de **DAO aquí.**
Si quisieramos que nuestro origen de datos fuese una lista en memoria, sólo tendríamos que utilizar la otra implementación:
public TareaRepository(InMemoryTareaDAO tareaDao)

```
@Repository
public class TareaRepository {

    private TareaDao tareaDao;

    @Autowired
    public TareaRepository(SQLTareaDAO tareaDao) {
        this.tareaDao = tareaDao;
    }

    /**
     * Obtiene la Tarea con codigo @codTarea. En caso de que no la encuentre
devolverá una excepción @NotFoundException
     * @param codigo
     */
    public Tarea get(int codigo) throws NotFoundException {
        return tareaDao.getById(codigo);
    }
```

Solo nos **tenemos que preocupar de acceder** a la **información sin importar** si el **DAO** está implementado en **Ficheros**, en **Memoria** o en una **BD relacional**

2.4 Ejemplo II. Uso del DAO

Podríamos **implementar un DAO** utilizando una base de datos en memoria (ArrayList) y, al usarlo, nuestro repositorio y resto de la aplicación **no se vería afectado**

```
@Repository
public class InMemoryTareaDAO implements TareaDao {
  ....
   @Override
   public Tarea getById(int id) {
       Tarea tarea = find(codigo);
       if (tarea == null) {
         throw new NotFoundException("La tarea con código " + codigo + " no
existe");
       }
       return tarea;
   }
}
```

```
@Repository
public class TareaRepository {

   private TareaDao tareaDao;

   @Autowired
   public TareaRepository(InMemoryTareaDAO tareaDao)
   {
       this.tareaDao = tareaDao;
   }
....
```

Actividad Previa

Actividad 7.- Crea un fork de la **siguiente plantilla** e implementa los métodos que nos permitan:

- Visualizar el **listado de tareas** (mostrando el id de las categorías asociadas) y llevar a cabo el filtrado a partir del nombre de usuario

Pasos a seguir:

- **Revisa el controlador** asociado al **caso de uso** que vayas a resolver para averiguar la vista y el método del repositorio implicado
- **Revisa la vista asociada** al controlador.
- **Revisa** el **método del repositorio** que se utiliza desde el controlador y haz los cambios necesarios **(si es el caso)**
- **Implementa** el método de la clase `SQLTareaDAO` necesario para que funcione el caso de uso que estás resolviendo
- Captura la excepción `SQLException` dentro del propio DAO, lanzando en el bloque catch una nueva excepción `DatabaseErrorException` con mensaje, el mensaje de la excepción inicial

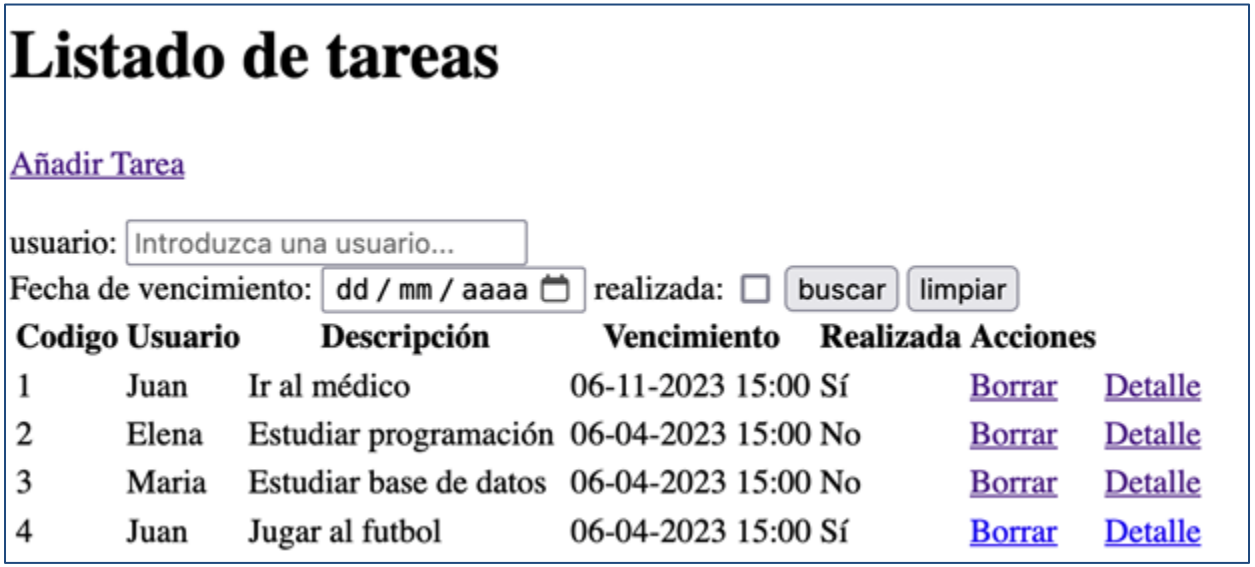
¿En qué capa se sitúan los cambios que has realizado?

Actividad Previa

Actividad 7 (continuación).- Implementa los métodos del DAO para poder llevar a cabo la inserción (*insertando el valor NULL en la categoría*) y el borrado de tareas a través de `SQLTareaDAO`. Prueba estos métodos a través de las vistas ya creadas.

3. Actividad Previa

Actividad 7 (continuación).- Implementa el método `findAllWithParams(Boolean isRealizada, LocalDate fecha, String usuario)` de la clase `SQLTareaDAO` y realiza los cambios necesarios para poder filtrar las tareas **por fecha** y si han sido **realizadas o no,** además de los que teníamos anteriormente. (Todos **los filtros** serán **opcionales;** los valores de los parámetros serán recibidos como `null` si no se envían datos**)**.



Listado de tareas

Añadir Tarea

usuario: Introduzca una usuario...

Fecha de vencimiento: dd / mm / aaaa realizada: buscar limpiar

Codigo	Usuario	Descripción	Vencimiento	Realizada	Acciones	
1	Juan	Ir al médico	06-11-2023 15:00	Sí	Borrar	Detalle
2	Elena	Estudiar programación	06-04-2023 15:00	No	Borrar	Detalle
3	Maria	Estudiar base de datos	06-04-2023 15:00	No	Borrar	Detalle
4	Juan	Jugar al futbol	06-04-2023 15:00	Sí	Borrar	Detalle

- Puedes implementar **los métodos** que consideres necesarios en el **repositorio** para llevar a cabo la actividad propuesta.

- Eso es todo... de momento :-)