Tarea online PROG10

Título de la tarea: Tarea online PROG10

Unidad: PROG10

Ciclo formativo y módulo: Desarrollo de Aplicaciones Web. Programación

Curso académico: 2024/2025

¿Qué contenidos o resultados de aprendizaje trabajaremos?

Contenidos

PROG 10.- Gestión de base de datos: relacionales y orientadas a objetos. Persistencia de objetos.

- BD relacionales.
  - El desfase objeto-relacional.
  - JDBC.
  - Conectores o Drivers.
  - Instalación de la base de datos.
  - Creación de las tablas en una base de datos.
    - Lenguaje SQL (I).
      - Lenguaje SQL (II).
  - Establecimiento de conexiones.
    - Configurar un conector JDBC en un proyecto NetBeans.
    - Registrar el controlador JDBC.
  - Ejecución de consultas sobre la base de datos.
    - Adición de información.
    - Recuperación de información (I).
      - Recuperación de información (II).
    - Actualización de información.
    - Borrado de información.
    - Cierre de conexiones.
    - Excepciones en JDBC.
- Persistencia de objetos.
  - ¿Qué son las bases de datos orientadas a objetos?
  - ¿Qué es el mapeado de objetos?
  - ¿Cómo empiezo a usar JPA?.
  - Unidades de persistencia (I).
    - Unidades de persistencia (II).
  - Modelo de datos en JPA.
    - Entidades.
    - Atributos.
    - Relaciones entre entidades.

Relaciones uno a uno.
Relaciones uno a muchos.
Operaciones CRUD con JPA.
Persistir y leer.
Actualizar y borrar instancias.
JPQL (I).
JPQL (II).

Resultados de aprendizaje

- **RA8**. Utiliza bases de datos orientadas a objetos, analizando sus características y aplicando técnicas para mantener la persistencia de la información.
- **RA9**. Gestiona información almacenada en bases de datos relacionales manteniendo la integridad y consistencia de los datos.

1.- Descripción de la tarea

Caso práctico

A **Juan** se le ocurrió mandarle a **Ana** hacer una aplicación para la gestión de un sistema de matrículas para un centro educativo. **Juan**, después de revisar el código, decide indicarle a **Ana** algunas de las mejoras que podría aplicar a dicha aplicación y así que aproveche para introducir algunos nuevos requisitos que acaba de incorporar el cliente y que ahora es el momento de aplicarlos. Eso es lo que va a intentar hacer **Ana**.

Para ello **Juan** le dice a **Ana**, que partiendo de su última versión de la aplicación para gestionar el sistema de matriculación, utilice un SGBD para almacenar los datos de dicha aplicación. **Ana**, ni corta ni perezosa, se pone manos a la obra.

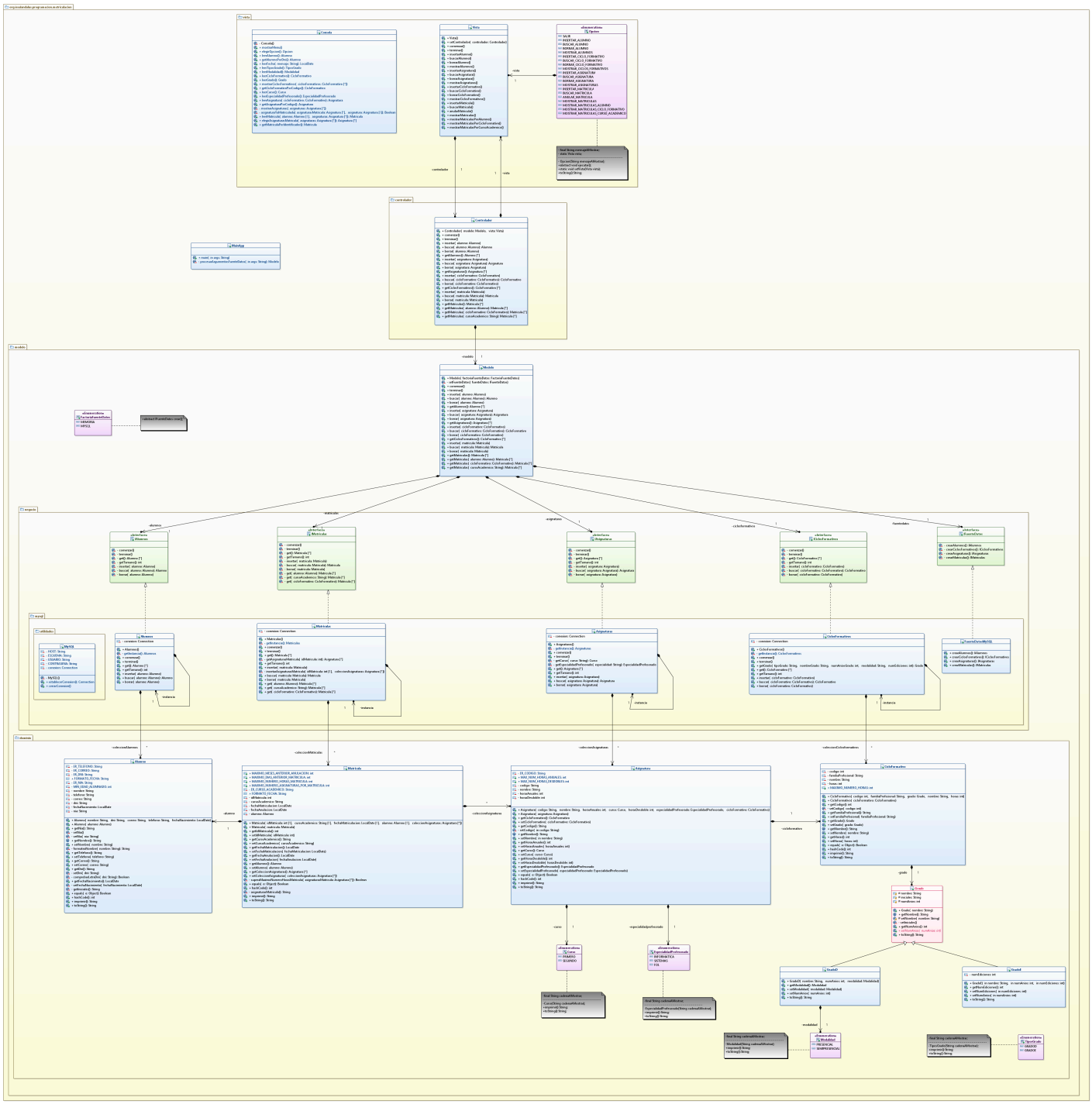
¿Qué te pedimos que hagas?

Nuestro cliente está muy satisfecho de como está quedando la aplicación. Sin embargo, nos acaba de comentar que por favor añadamos persistencia a los datos introducidos, ya que tal y como está ahora la aplicación no es **funcional**. Por tanto, en este **quinto spring** añadiremos persistencia a los datos utilizando para ello bases de datos relacionales. Por tal motivo, añadiremos al modelo que teníamos de memoria, un modelo de base de datos. Además, el cliente ha añadido un nuevo requerimiento: quiere que la aplicación no permita borrar un alumno, una asignatura o un ciclo formativo si existe alguna matrícula para dicho alumno, ciclo formativo o asignatura.

En definitiva, lo que nos pide el cliente es lo siguiente:

- Que la aplicación almacene los datos en una base de datos creada para tal efecto.
- La base de datos debe estar implementada en **MySQL**.
- Acomodar el proyecto para que gradle gestione la dependencia con el driver para java de MySQL en su última versión.
- Modificar el proyecto para que se puedan ejecutar las dos versiones: memoria con vista textual y BD con vista textual.
- Gestionar los alumnos para que su persistencia se lleve a cabo por medio de dicha BD.
- Gestionar los ciclos formativos para que su persistencia se lleve a cabo por medio de dicha BD.
- Gestionar las asignaturas para que su persistencia se lleve a cabo por medio de dicha BD.
- Impedir el borrado de un alumno si éste tiene ya una matrícula.
- Impedir el borrado de una asignatura si ésta tiene ya una matrícula.
- Impedir el borrado de un ciclo formativo si éste tiene ya una matrícula.

Para todo esto, te muestro un diagrama de clases y poco a poco te iré explicando los diferentes pasos a realizar:



Primeros Pasos

Lo primero que debes hacer es realizar un **clone** de tu repositorio a un nuevo repositorio para que contenga la versión 4 de la aplicación.
Clona tu repositorio remoto recién copiado en GitHub a un repositorio local que será donde irás realizando lo que a continuación se te pide.
Modifica el fichero gradle para que incluya entre sus dependencias la versión del driver de mysql que se usará en el proyecto:

```
// Driver MySQL
implementation 'mysql:mysql-connector-java:8.0.28'
```

Realiza tu **primer commit**.

Creación de base de datos MySQL en la nube

Crea en AWS una nueva base de datos MySQL. La base de datos debe llamarse **dbsistemamatriculacion**.
Crea cinco tablas en dicha base de datos llamadas respectivamente **alumno, cicloFormativo, asignatura, matricula y asignaturasMatricula**. Para ello haz uso del script sql (sql - 1,81 KB) proporcionado.
Crea un nuevo usuario llamado **admin** y con password **sistemamatriculacion-2025**. Este usuario deberá tener **permisos de lectura y escritura** sobre la base de datos.

Paquete negocio

Extrae las interfaces correspondientes a las clases **Alumnos**, **Asignaturas**, **CiclosFormativos** y **Matriculas**, llamándolas **IAlumnos**, **IAsignaturas**, **ICiclosFormativos** e **IMatriculas**, respectivamente.
Añade los métodos **comenzar** y **terminar** a cada una de las interfaces creadas.
Haz que las clases **Alumnos**, **Asignaturas**, **CiclosFormativos** y **Matriculas** implementen la interfaz que le corresponda. Esto provocará que tengas que implementar los métodos **comenzar** y **terminar** en las clases ya existentes que implementan dicha interfaz. Los cuerpos de ambos métodos en las cuatro clases deberán llamar respectivamente a los métodos **establecerConexion** y **cerrarConexion** de la clase **MySQL**.
Crea en el paquete **negocio** una Interface **IFuenteDatos** que tendrá los métodos **crearAlumnos**, **crearCiclosFormativos**, **crearAsignaturas**, **crearMatriculas**.
Crea un paquete llamado **memoria**.
Mueve las clases **Alumnos**, **Asignaturas**, **CiclosFormativos** y **Matriculas** al paquete **memoria**.
Crea en el paquete **memoria** una nueva clase llamada **FuenteDatosMemoria** que implementa la interfaz **IFuenteDatos**. Esta clase será la encargada de implementar el **patrón fábrica**, devolviendo en cada caso el resultado de crear la colección a la que hace referencia su nombre:

- Método **crearAlumnos** devolverá una nueva colección de tipo **Alumnos** del paquete **memoria**.
- Método **crearCiclosFormativos** devolverá una nueva colección de tipo **CiclosFormativos** del paquete **memoria**.
- Método **crearAsignaturas** devolverá una nueva colección de tipo **Asignaturas** del paquete **memoria**.
- Método **crearMatriculas** devolverá una nueva colección de tipo **Matriculas** del paquete **memoria**.

Realiza el **commit** correspondiente.

Paquete mysql

Crea el paquete **utilidades** que contendrá una clase **MySQL** que constará de las siguientes constantes y métodos:

- **HOST** cuyo valor será el **nombre de la máquina hospedada en la nube y que aloja la base de datos creada**.
- **ESQUEMA** que tendrá como valor el **nombre de la base de datos creada**.
- **USUARIO** cuyo valor será el **nombre del usuario** con el que te conectas a la base de datos.
- **CONTRASEN**A cuyo valor será la **password del usuario** con el que te conectas a la base de datos.
- Crea el método **establecerConexion** que se encargará de **realizar la conexión de la aplicación a la base de datos** alojada en la nube.
- Crea el método **cerrarConexion** que se encargará de **cerrar la conexión de la aplicación con la base de datos** alojada en la nube.

Añade la clase **Alumnos** tal y como aparece en el diagrama de clases teniendo en cuenta que:

- Implementa el patrón **singlenton** a través del atributo **instancia** y del método **getInstancia** que si el atributo **instancia** es nulo devolverá una instancia de la clase **Alumnos**, y si no es nulo devolverá el valor del atributo **instancia**.
- El constructor de la clase se encargará de llamar al método **comenzar**.
- El método **comenzar** deberá crear una conexión con la base de datos.
- El método **terminar** deberá cerrar la conexión con la base de datos.

El método **get** deberá devolver una lista formada por todos los alumnos existentes en la base de datos ordenados por **dni**.

El método **getTamano** deberá devolver el número de alumnos existentes en la base de datos.

El método **insertar** deberá insertar un nuevo alumno en la base de datos.

El método **buscar** deberá devolver el resultado de encontrar en la base de datos al alumno pasado como parámetro.

El método **borrar** deberá eliminar de la base de datos al alumno pasado como parámetro.

Añade la clase **CiclosFormativos** tal y como aparece en el diagrama de clases teniendo en cuenta que:

Implementa el patrón **singlenton** a través del atributo **instancia** y del método **getInstancia** que si el atributo **instancia** es nulo devolverá una instancia de la clase **CiclosFormativos**, y si no es nulo devolverá el valor del atributo **instancia**.

El constructor de la clase se encargará de llamar al método **comenzar**.

El método **comenzar** deberá crear una conexión con la base de datos.

El método **terminar** deberá cerrar la conexión con la base de datos.

El método **getGrado** que, dependiendo del parámetro de tipo **String tipoGrado**, deberá devolver un objeto de tipo **GradoD** o **GradoE**. que serán creados a partir del resto de parámetros pasados al método.

El método **get** deberá devolver una lista formada por todos los ciclos formativos existentes en la base de datos ordenados por **nombre**.

El método **getTamano** deberá devolver el número de ciclos formativos existentes en la base de datos.

El método **insertar** deberá insertar un nuevo ciclo formativo en la base de datos.

El método **buscar** deberá devolver el resultado de encontrar en la base de datos el ciclo formativo pasado como parámetro.

El método **borrar** deberá eliminar de la base de datos al ciclo formativo pasado como parámetro.

Añade la clase **Asignaturas** tal y como aparece en el diagrama de clases teniendo en cuenta que:

Implementa el patrón **singlenton** a través del atributo **instancia** y del método **getInstancia** que si el atributo instancia es nulo devolverá una instancia de la clase **Asignaturas**, y si no es nulo devolverá el valor del atributo **instancia**.

El constructor de la clase se encargará de llamar al método **comenzar**.

El método **comenzar** deberá crear una conexión con la base de datos.

El método **terminar** deberá cerrar la conexión con la base de datos.

El método **getCurso** que, dependiendo del parámetro de tipo **String curso**, deberá devolver un objeto de tipo **Curso**.

El método **getEspecialidadProfesorado** que, dependiendo del parámetro de tipo **String especialidad**, deberá devolver un objeto de tipo **EspecialidadProfesorado**.

El método **get** deberá devolver una lista formada por todos las asignaturas existentes en la base de datos ordenadas por **nombre**.

El método **getTamano** deberá devolver el número de asignaturas existentes en la base de datos.

El método **insertar** deberá insertar una nueva asignatura en la base de datos.

El método **buscar** deberá devolver el resultado de encontrar en la base de datos la asignatura pasada como parámetro.

El método **borrar** deberá eliminar de la base de datos la asignatura pasada como parámetro.

Añade la clase **Matriculas** tal y como aparece en el diagrama de clases teniendo en cuenta que:

Implementa el patrón **singlenton** a través del atributo **instancia** y del método **getInstancia** que si el atributo **instancia** es nulo devolverá una instancia de la clase **Matriculas**, y si no es nulo devolverá el valor del atributo **instancia**.

El constructor de la clase se encargará de llamar al método **comenzar**.

El método **comenzar** deberá crear una conexión con la base de datos.

El método **terminar** deberá cerrar la conexión con la base de datos.

El método **getAsignaturasMatricula** que a partir de un identificador de una matrícula, deberá devolver una lista de todas las asignaturas pertenecientes a dicha matrícula.

El método `get` deberá devolver una lista formada por todas las matrículas existentes en la base de datos ordenadas por fecha de matriculación en orden descendente (es decir, las matrículas más recientes primero) y en caso de que existan varias matrículas con la misma fecha de matriculación, deberá considerarse como segundo criterio de ordenación el nombre del alumno correspondiente a la matrícula.
El método `getTamano` deberá devolver el número de matrículas existentes en la base de datos.
El método `insertarAsignaturasMatricula` que a partir de un identificador de matrícula y una lista de asignaturas pertenecientes a la matrícula, deberá realizar las inserciones correspondientes en la tabla `asignaturasMatricula` de la base de datos.
El método `insertar` deberá insertar una nueva matrícula en la base de datos.
El método `buscar` deberá devolver el resultado de encontrar en la base de datos la matrícula pasada como parámetro.
El método `borrar` deberá eliminar de la base de datos la matrícula pasada como parámetro.
El método `get` de un alumno deberá devolver una lista de todas las matrículas del alumno pasado como parámetro.
El método `get` de un ciclo formativo deberá devolver una lista de todas las matrículas del ciclo formativo pasado como parámetro.
El método `get` de un curso académico deberá devolver una lista de todas las matrículas del curso académico pasado como parámetro.

Crea la clase `FuenteDatosMySQL` que deberá implementar la interfaz `IFuenteDatos`, tal y como se indica en el diagrama. Esta clase será la encargada de implementar el **patrón fábrica**, devolviendo en cada caso el resultado de crear la colección a la que hace referencia su nombre:

Método `crearAlumnos` devolverá una nueva colección de tipo Alumnos del paquete `mysql`.
Método `crearCiclosFormativos` devolverá una nueva colección de tipo CiclosFormativos del paquete `mysql`.
Método `crearAsignaturas` devolverá una nueva colección de tipo Asignaturas del paquete `mysql`.
Método `crearMatriculas` devolverá una nueva colección de tipo Matriculas del paquete `mysql`.

Realiza el **commit** correspondiente.

Paquete modelo

Crea el enumerado `FactoriaFuenteDatos` tal y como se muestra en el diagrama de clases. Este enumerado implementa el **patrón método de fabricación** para las dos fuentes de datos que se van a tener: **MEMORIA y MYSQL**.
Añade a la clase `Modelo`, el atributo `fuenteDatos` tal y como aparece en el diagrama de clases.
Implementa el método `setter` correspondiente a la fuente de datos.
Modifica el constructor de la clase para que reciba y **establezca la fuente de datos de la aplicación** .
Modifica el método `comenzar` para que cree las colecciones en función de la fuente de datos a utilizar.
Modifica el método `terminar` para que llame al método `terminar` de cada una de las colecciones.
Realiza el **commit** correspondiente.

MainApp

Implementa el método `procesarArgumentosFuenteDatos` que creará un modelo cuya fuente de datos será la que se indique a través de los parámetros de la aplicación. Si el parámetro es `-fdmemoria`, se creará un modelo cuya fuente de datos será de tipo **MEMORIA**. En cambio, si el parámetro es `-fdmysql`, se creará un modelo cuya fuente de datos será de tipo **MYSQL**.
Modifica el método `main` para que se cree el modelo indicado a través de los parámetros de la aplicación.
Realiza el **commit** correspondiente.
Finalmente, realiza el **push** hacia tu repositorio remoto en GitHub.

Se valorará:

- La indentación debe ser correcta en cada uno de los apartados.
- El nombre de las variables debe ser adecuado.
- Se debe utilizar la clase **Entrada** para realizar la entrada por teclado.
- El programa debe pasar todas las pruebas que van en el esqueleto del proyecto y toda entrada del programa será validada, para evitar que el programa termine abruptamente debido a una excepción. Además, que ni decir tiene, el programa no debe contener ningún error léxico, sintáctico, de dependencias, etc.
- La corrección ortográfica tanto en los comentarios como en los mensajes que se muestren al usuario.

2.- Información de interés

Recursos necesarios y recomendaciones

- Un IDE de desarrollo (IntelliJ, NetBeans o Eclipse). En dicho IDE deberás haber instalado el plugins que permita trabajar con proyectos gradle.

Indicaciones de entrega

Deberás crear un archivo de texto que incluya las explicaciones sobre los puntos que las merezcan según tu criterio. Es imprescindible que en este documento incluyas la URL al repositorio GitHub que has debido crear para realizar esta tarea.

Empaquetarás en un archivo comprimido el documento de texto anterior y el proyecto creado con el IDE que hayas utilizado.

El envío del archivo comprimido anterior lo realizarás a través de la plataforma. El archivo se nombrará siguiendo las siguientes pautas:

Apellido1_Apellido2_Nombre_PROG_Tarea10

3.- Evaluación de la tarea

Criterios de evaluación implicados

Criterios de Evaluación RA8

c.- Se han instalado sistemas gestores de bases de datos orientados a objetos.
e.- Se han creado bases de datos y las estructuras necesarias para el almacenamiento de objetos.
f.- Se han programado aplicaciones que almacenen objetos en las bases de datos creadas.
g.- Se han realizado programas para recuperar, actualizar y eliminar objetos de las bases de datos.
h.- Se han realizado programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.

Criterios de Evaluación RA9

b.- Se han programado conexiones con bases de datos.
e.- Se han efectuado borrados y modificaciones sobre la información almacenada.

¿Cómo valoramos y puntuamos tu tarea?

Rúbrica de la tarea		
Criterios de Evaluación	**Calificación**	**Retroalimentación**
8c		
8e		
8f		
8g		
8h		
9b		
9e		