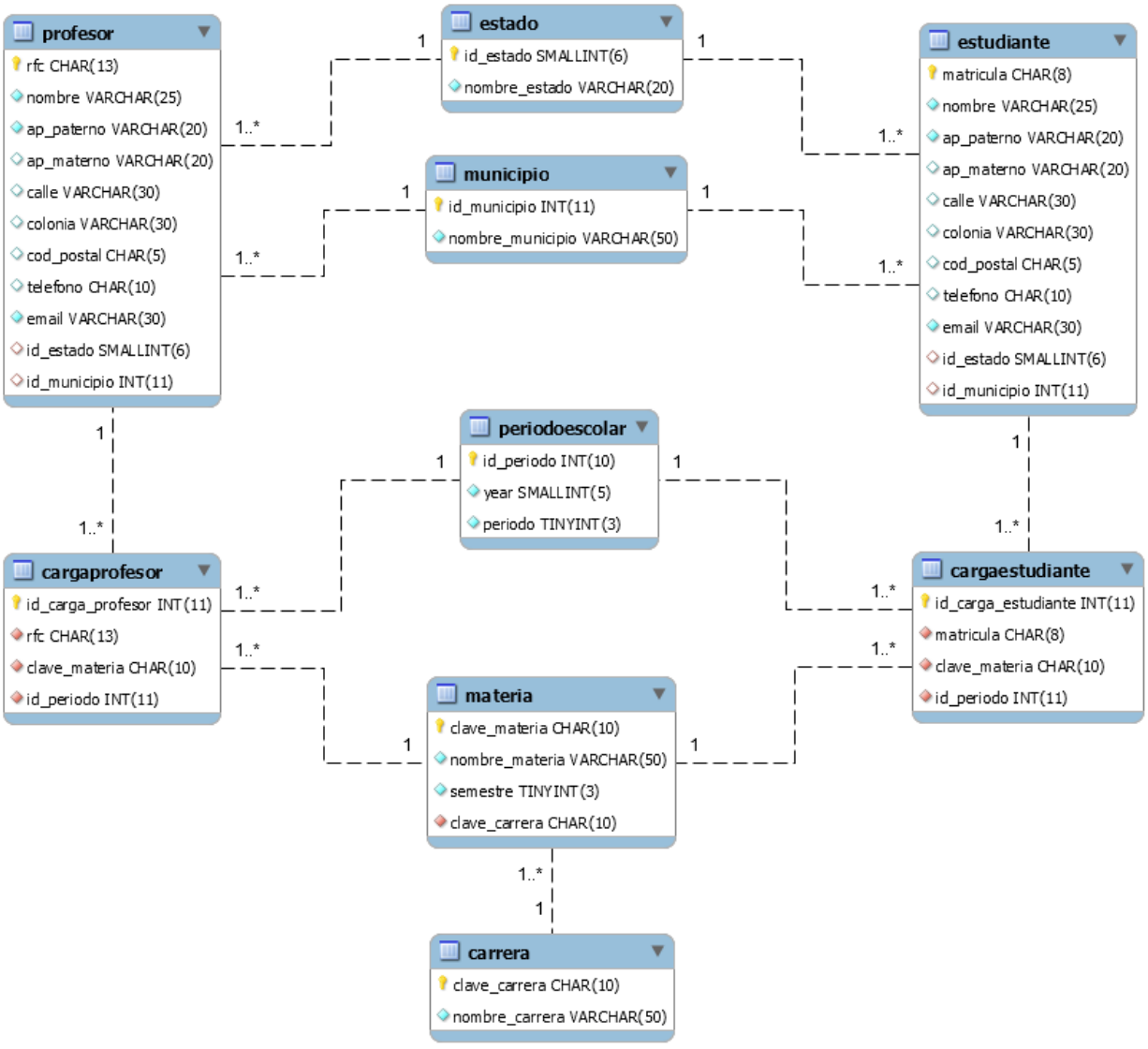
PROGRAMACION ORIENTADA A OBJETOS II

PROGRAMA 1

El primer programa consiste en desarrollar el conjunto de clases de entidad, implementar los DAOs genéricos para cada clase de entidad e implementar un DAO general para acceder a la siguiente base de datos que representa la información para llevar el control de concursos de programación tipo ICPC. Los DAOs en este programa deberán ser implementados usando JDBC para acceder a bases de datos en MySQL de manera similar a lo que se estará realizando en la práctica 8 con la guía del instructor, donde se desarrollará el DAO para las tablas municipio y estudiante. El diagrama de la base de datos a utilizar se muestra a continuación, que es similar a la usada en las prácticas 5 a 7:



En la figura los campos que tienen una llavecita amarilla a la izquierda son la llave primaria de la tabla, los campos con un rombo azul o rojo son obligatorios (los rojos indican de hecho que son llaves foráneas, es decir, que el valor contenido en estos campos son el valor de una llave primaria en la tabla relacionada, y esta relación se indica con las líneas punteadas que unen diferentes tablas).

Para crear la base de datos (cuyo nombre es `controlescolar`), y colocar datos iniciales, se proporcionan los scripts creaescolar.sql y llenaescolar.sql

Con el propósito de probar sus DAOs, pueden escribir clientes de texto básicos donde creen objetos de alguna clase de entidad, por ejemplo, Materia, y a través del DAO general se llame a los métodos correspondientes para agregar, actualizar, obtener o modificar una Carrera. Tales clientes serán para su propio uso y no serán evaluados como parte de la calificación.

Las clases de entidad que son usadas para representar cada una de las tablas, se encuentran en el paquete `poo2.progs.entidades`, y las implementaciones de los DAOs deberán colocare en el paquete `mx.edu.uaz.ingsoftware.poo2.basedatos` . Los métodos que deberán implementar los DAOs para cada tabla estarán indicados por la interface `Dao<T>` y el DAO general por la interface `DAOEscolares`. Algunos DAO podrían comunicar errores a través de la excepción `DaoException` (aunque para este programa optaremos por no comunicar estos errores) estas 2 interfaces y la clase de excepción estarán en el paquete `poo2.progs.interfaces`

INSTRUCCIONES PARA EL PROYECTO DEL PROGRAMA 1 (Y PRÁCTICA 8)

Paso 1: Deberá ejecutar los scripts creaescolar.sql y llenaescolar.sql que se encuentran en la carpeta raíz del proyecto, de la siguiente manera desde la terminal de comandos, habiéndose colocado primero en la carpeta raíz del proyecto**:**

```
mysql -u root -p  --default-character-set=utf8 < creaescolar.sql

mysql -u root -p  --default-character-set=utf8 < llenaescolar.sql
```

Para cada uno de estos se le pedirá la clave de root, tecléela y presione Enter. El segundo comando puede tomar algunos segundos pues coloca varios registros en las tablas.

Paso 2. Clone el repositorio usando la técnica que le convenga (ya sea usando Intellij IDEA o desde la terminal de comandos usando **git clone**). Los detalles están en el archivo README.md

Paso 3. Con la guía del instructor, se estará demostrando lo que hay que realizar para el Programa 1, completando específicamente lo relacionado a las tablas municipio y estudiante (la calificación de la Práctica 8 es en base a de estas dos tablas).

Paso 4. Una vez completado lo relacionado a la Práctica 8, podrá continuar de manera independiente lo restante. El mismo proyecto se seguirá usando para completar el código necesario para manejar el resto de las tablas.

NOTA IMPORTANTE: Por el momento las únicas pruebas completas son las que tienen que ver con lo que se va a completar para la práctica 8 (están dentro de la sección test en el paquete poo2.prac08.main). Para lo que completarán para el Programa 1, en los siguientes días se les estarán proporcionando los archivos de pruebas. Cada vez que haya nuevas clases de prueba deberás:

a) Agregarlas a la sección test en el paquete poo2.progs.main
b) Incluirlas haciendo un commit

c) Antes de que hagan un push (al completar la implementación de algún método y después de hacer el commit respectivo), **deben** hacer un pull antes de hacer el push (por si el repositorio remoto tiene incluidas nuevas pruebas), de otra manera les marcará error al hacer el push

CON RESPECTO A LAS CLASES DE ENTIDAD

A las clases de entidad que se generan de manera automática por parte de IntelliJ IDEA es necesario agregarles lo siguiente:

- Implementar la interface Serializable
- Tres constructores, 1 constructor vacío, 1 constructor que reciba la llave primaria de la tabla correspondiente y 1 constructor que reciba los datos obligatorios de la tabla correspondiente (siguiendo el orden que especifica la estructura de la tabla). Las clases que comienzan con **Detalle** tienen como campos obligatorios los de la clase de la que heredan (la clase de la que heredan está indicado por el resto del nombre de la clase), y los constructores de estas clases extendidas deben llamar al constructor correspondiente de la clase padre.
- El método equals y el método hashCode usando solamente el campo que es llave primaria
- El método toString que devolverá lo siguiente:
    - El valor de nombreCarrera en la clase Carrera
    - El valor de nombreMateria en la clase Materia
    - El valor de nombreMunicipio en la clase Municipio
    - El valor de nombreEstado en la clase Estado
    - El valor de nombre seguido de un espacio, del valor de apPaterno, de un espacio y el valor de apMaterno en la clase Estudiante (el espacio y valor de apMaterno solo se incluye en lo que se regresa si apMaterno no es null y tiene por lo menos un carácter
    - El valor de nombre seguido de un espacio, del valor de apPaterno, de un espacio y el valor de apMaterno en la clase Profesor (el espacio y valor de apMaterno solo se incluye en lo que se regresa si apMaterno no es null y tiene por lo menos un carácter
    - El valor de year seguido de un espacio y el un string que expresa el periodo de acuerdo a las siguientes reglas, si el valor de periodo es 1, el string es Agosto-Diciembre, si es 2 es Enero-Junio y si es 3 es Verano. Así, si por ejemplo un objeto de clase Periodoescolar tiene como atributos year=2020 y periodo=2, entonces el método toString debería devolver el string **`2020 Enero-Junio`**
    - El valor de year, string asociado con el periodo y el nombreMateria separados por n tabulador y asegurándose que el valor de year ocupa 4 espacios y que el valor del string asociado con el período ocupe 17 espacios para la clase DetalleCargaprofesor, por ejemplo: **`2020 Agosto-Diciembre ALGEBRA LINEAL`**
    - El valor de year, string asociado con el periodo y el nombreMateria separados por n tabulador y asegurándose que el valor de year ocupa 4 espacios y que el valor del string asociado con el período ocupe 17 espacios para la clase DetalleCargaestudiante, por ejemplo: **`2020 Enero-Junio ALGEBRA LINEAL`**
    - Las clases Cargaprofesor y Cargaestudiante no necesitan tener el método toString
    - Recuerde que la clase DetalleCargaprofesor hereda de la clase Cargaprofesor agregándole los atributos periodo y nombreMateria (el primero long y el segundo String) con sus respectivos getters y setters (los campos obligatorios serían los de la tabla cargaprofesor) y que la clase DetalleCargaestudiante hereda de la clase Cargaestudiante agregándole los atributos periodo y nombreMateria (el primero long y el segundo String) con sus respectivos getters y setters (los campos obligatorios serían los de la tabla cargaestudiante)

NOTA: En el proyecto que clonan ya todas las clases de entidad, pero se demostrará en las sesiones siguientes como se puede crear a partir de la base de datos de manera automática por parte de Intellij IDEA. Si no está usando Intellij, entonces se tendrían que crear la clase de manera manual

CON RESPECTO A LAS CLASES QUE IMPLEMENTAN LA INTERFACE Dao<T>

Se hace notar que las validaciones que serán realizadas para los DAO de Municipio y Estudiante en la práctica 8, deben realizarse en los restantes DAOs, y son básicamente las siguientes:

- Al agregar o actualizar un registro deben asegurarse de que:
    - los campos que son String no excedan en longitud del tamaño correspondiente a la columna de la tabla
    - que los campos que hagan referencia a valores que son llave primaria en otras tablas existan en tales tablas. Específicamente:
        - el valor de idEstado en Profesor y Estudiante debe existir en Estado
        - el valor de idMunicipio en Profesor y Estudiante debe existir en Municipio
        - el valor de claveCarrera en Materia debe existir en Carrera
        - el valor de idCargaProfesor en Cargaprofesor debe existir como valor en la columna asociada al atributo idPeriodo de PeriodoEscolar y el rfc debe existir en la columna asociada al rfc en Profesor
        - el valor de idCargaEstudiante en Cargaestudiante debe existir como valor en la columna asociada al atributo idPeriodo de PeriodoEscolar y la matricula debe existir en la columna asociada a la matricula en Estudiante
        - el valor claveMateria en Cargaestudiante y Cargaprofesor debe existir como valor en la columna asociada al atributo claveMateria en Materia
    - que los campos obligatorios no tengan un valor null en el objeto de la clase de entidad recibida y que en caso de ser String no estén vacíos (una excepción a esto es para los campos autoincrementables **al agregar** pues si el valor correspondiente al campo autoincrementable es null o cero,en el atributo de la clase de entidad recibida como argumento, esto indicaría que se pusieran un NULL para tal columna en la sentencia SQL lo cual a su vez indicaría a MySQL que le asigne el valor de manera automática. **En tal caso debería de actualizarse el objeto de la clase entidad recibido con el valor que se asignó (ejemplos de cómo determinar el valor que se le asignó al campo autoincrementable lo pueden encontrar en el código de las prácticas 6 y 7)**.
    - En el caso de DaoPeriodoescolar, se debe verificar que periodo solo tenga un valor entre 1 y 3 y que el valor de year esté entre 2000 y el año actual de acuerdo a la configuración de la computadora (se asume que solo se tienen registros desde el año 2000)
    - En el caso de DaoEstudiante y DaoProfesor, el valor de teléfono, si es que se proporciona (es decir si no es null y si no está vacío) debe ser de exactamente 10 dígitos
    - En el caso de DaoEstudiante y DaoProfesor, el valor de codpostal, si es que se proporciona (es decir si no es null y si no está vacío) debe ser de exactamente 5 dígitos
    - En el caso de DaoDatosEstudiante, el valor de matricula, debe estar compuesto de exactamente 8 dígitos
    - En el caso de DaoProfesor, el rfc debe ser de 10 o 13 caracteres validando que los primeros 10 tienen la estructura de un RFC (4 letras en mayúscula seguidas de 6 dígitos en formato AAMMDD, que representan la fecha de nacimiento, AA son los dos últimos dígitos del año, MM son los dos del mes y DD son los dos del día. (La fecha debe ser válida para aceptar el RFC). Después de estos primeros días viene cualquier combinación de 3 dígitos y/o letras mayúsculas (lo que se llama homoclave)
    - Para el programa (mas no para la practica 8), en el caso de DaoEstudiante y DaoProfesor el valor de email debe contener un email valido en cuanto a su estructura, donde se asumirá que el usuario está formado solo por dígitos, letras minúsculas, puntos, guiones y subrayados, y el nombre de dominio tiene al menos dos partes separados por punto, donde cada parte solo esta compuesta de letras minúsculas, guiones, subrayados y dígitos
    - En DaoMateria, el valor de semestre tiene que ser un entero entre 1 y 10.
    - Si algún valor no es válido el método save o update debería regresar false.

- Al eliminar es necesario validar que el valor recibido como argumento no sea null, que sea del tipo adecuado a la llave primaria y que el valor de la llave primaria en el registro a eliminar no se encuentre como valor en alguna otra tabla relacionada:
    - un estudiante no se puede eliminar si ya tiene registros asociados en Cargaestudiante
    - Un estudiante no se puede eliminar si ya tiene registros asociados en Cargaestudiante
    - Un profesor no se puede eliminar si ya tiene registros asociados en Cargaprofesor
    - Una carrera no se puede eliminar si ya tiene registros asociados en Materia
    - Un periodoescolar no se puede eliminar si ya tiene registros asociados ya sea en Cargaestudiante o en Cargaprofesor
    - Una materia no se puede eliminar si ya tiene registros asociados ya sea en Cargaestudiante o en Cargaprofesor

Habiendo dejado claro esto, para cada tabla se debe implementar el DAO correspondiente (implementando la interface Dao<T>) y tales clases deberán llamarse DaoCarrera, DaoMateria, DaoEstado, DaoMunicipio, DaoProfesor, DaoCargaprofesor, DaoEstudiante, DaoCargaestudiante y DaoPeriodoescolar. Todas estas clases deberán quedar en el paquete `poo2.progs.basedatos`.

CASOS ESPECIALES DE Dao DONDE NO SE REQUIERE IMPLEMENTAR TODOS LOS METODOS:

- En las clases DaoEstado y DaoMunicipio (que implementan la interface Dao<T> para las tablas estado y municipio respectivamente) deberán simplemente regresar false en los métodos save, update y delete, dado que tales tablas se consideran catálogos, es decir, tablas cuyo contenido no cambia y por tanto no debería ser posible agregar, actualizar o eliminar registros en tales tablas.
- En las clases DaoCargaprofesor y DaoCargaestudiante, el método update debe simplemente devolver false dado que no necesitamos la funcionalidad de actualizar, el método getAll debe devolver una List vacía de objetos Cargaprofesor (o Cargaestudiante según sea el caso) ya que no necesitamos la funcionalidad de obtener todas las cargas.

Se reitera que, si bien es posible hacer que los métodos que implementan la interface comuniquen los errores que se pudieran presentar al hacer alguna operación en la base de datos usando la excepción DaoException, en este proyecto se ha decidido no hacerlo, y **solo los constructores si avisaran a través de la excepción DaoException si no se puede construir el objeto**. Simplemente regresaremos false en caso de que alguna operación no pueda realizarse. Durante la depuración, en el catch de los bloques try donde se ejecutan sentencias SQL podrían llamar a printStackTrace() sobre el objeto de la excepción, pero **asegúrense de quitar tales llamadas una vez que suban el programa final**.

CON RESPECTO A LA CLASE DaoEscolaresImpl

También deberá completarse el código de la clase DaoEscolaresImpl que implementa la interface DaoEscolares (la cual durante la práctica 8 se completará en lo relacionado a las tablas municipio y estudiante). La clase DaoEscolaresImpl se ayudará de los DAOs mencionados en el párrafo anterior, por lo cual en el método inicializaDaos deberá inicializar cada DAO pasándole el objeto Connection que se creó en el constructor de la clase DaoEscolaresImpl. Esta clase también deberá quedar en el paquete `poo2.progs.basedatos`.

La interface DaoEscolaresImpl, de hecho, tiene los siguientes métodos a implementar, muchos de ellos simplemente llamando al método save, get, getAll, delete o update de la clase correspondiente a la tabla consultada, a excepción de los marcados en azul cuyo código debe implementarse en esta clase dado que los Daos individuales no proporcionan la funcionalidad requerida al necesitarse la fusión (join) de información de varias tablas:

- **public List<Carrera> obtenCarreras()** Este método devolverá una lista de carreras contenidas en la tabla carrera de la base de datos

- **public boolean agregaCarrera(Carrera c)** Este método agrega la carrera representada por el objeto c a la tabla carrera (note que puede ser que no se pueda agregar el objeto por lo que regresa true si se pudo o false si no)
- **public boolean eliminaCarrera(String clave)** Este método elimina la carrera cuyo valor en el campo llave es igual al dado como argumento (clave) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no). NOTA: Este método debe devolver false en caso de que haya registros en la tabla materia que estén relacionados con la carrera que se desea borrar.
- **public boolean modificaCarrera(Carrera c)** Este método modifica el nombre de la carrera representada por el objeto c en la tabla carrera (note que puede ser que no se pueda modificar el objeto por lo que regresa true si se pudo o false si no)

- **public List<Materia> obtenMaterias()** Este método devolverá una lista de materias contenidas en la tabla materia de la base de datos
- **public boolean agregaMateria(Materia m)** Este método agrega la materia representada por el objeto m a la tabla materia (note que puede ser que no se pueda agregar el objeto por lo que regresa true si se pudo o false si no)
- **public boolean eliminaMateria(String clave)** Este método elimina la materia cuyo valor en el campo llave es igual al dado como argumento (clave) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no). NOTA: Este método debe devolver false en caso de que haya registros en la tabla cargaestudiante y/o cargaprofesor que tienen que ver con la materia a eliminar.
- **public boolean modificaMateria(Materia m)** Este método modifica el nombre, semestre y carrera de la materia representada por el objeto m en la tabla materia (note que puede ser que no se pueda modificar el objeto por lo que regresa true si se pudo o false si no)

- **public List<Periodoescolar> obtenPeriodos()** Este método devolverá una lista de periodos escolares contenidos en la tabla periodoescolar de la base de datos
- **public boolean agregaPeriodo(Periodoescolar p)** Este método agrega el periodo escolar representado por el objeto p a la tabla periodoescolar (note que puede ser que no se pueda agregar el objeto por lo que regresa true si se pudo o false si no). Este método debe obtener el id_periodo que se le haya asignado al registro agregado y colocarlo en el atributo correspondiente del objeto p recibido, ya que ese campo es autoincrementable
- **public boolean eliminaPeriodo(long id)** Este método elimina el periodo escolar cuyo valor en el campo llave es igual al dado como argumento (id) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no). NOTA: Este método debe devolver false en caso de que haya registros en la tabla cargaestudiante y/o cargaprofesor que tienen que ver con el periodo escolar a eliminar.
- **public boolean modificaPeriodo(Periodoescolar p)** Este método modifica el año y semestre del periodo escolar representado por el objeto p en la tabla periodoescolar (note que puede ser que no se pueda modificar el objeto por lo que regresa true si se pudo o false si no)

- **public List<Estado> obtenEstados()** Este método devolverá una lista de los estados contenidos en la tabla estado de la base de datos

- **public List<Municipio> obtenMunicipios(long idEstado)** Este método devolverá una lista de los municipios contenidos en la tabla municipio de la base de datos que son del estado recibido como argumento.

- **public List<Profesor> obtenProfesores()** Este método devolverá una lista de profesores contenidos en la tabla profesor de la base de datos
- **public boolean agregaProfesor(Profesor p)** Este método agrega el profesor representado por el objeto p a la tabla profesor (note que puede ser que no se pueda agregar el objeto por lo que regresa true si se pudo o false si no)
- **public boolean eliminaProfesor(String rfc)** Este método elimina el profesor cuyo valor en el campo llave es igual al dado como argumento (rfc) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no). NOTA: Este método debe devolver false en caso de que haya registros en la tabla en cargaprofesor asociados al profesor a eliminar.
- **public boolean modificaProfesor(Profesor p)** Este método modifica todos los datos de un profesor a excepción del RFC en la tabla profesor, los cuales están contenidos en el objeto p recibido como argumento (note que puede ser que no se pueda modificar el objeto por lo que regresa true si se pudo o false si no)

- **public List<DetalleCargaProfesor> obtenCargaProfesor(String rfc)** Este método devolverá la carga del profesor cuyo rfc es el dado como argumento, en forma de un objeto de clase DetalleCargaprofesor, la carga es consultada de las tablas cargaprofesor, materia y periodoescolar.
- **public boolean agregaCargaProfesor(Cargaprofesor c)** Este método agrega la carga del profesor representada por el objeto c a la tabla cargaprofesor (note que puede ser que no se pueda agregar el objeto por lo que regresa true si se pudo o false si no). El valor de la columna asociada al atributo idCargaProfesor es autoincrementable y por tanto, debe obtenerse y almacenarse en el objeto c recibido una vez que se agrega el registro a la tabla.
- **public boolean eliminaCargaProfesor(long idCarga)** Este método elimina la carga del profesor identificada por el valor recibido (idCarga) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no)

- **public List<Estudiante> obtenEstudiantes()** Este método devolverá una lista de estudiantes contenidos en la tabla estudiante de la base de datos
- **public boolean agregaEstudiante(Estudiante e)** Este método agrega el estudiante representado por el objeto e a la tabla estudiante (note que puede ser que no se pueda agregar el objeto por lo que regresa true si se pudo o false si no)
- **public boolean eliminaEstudiante(String matricula)** Este método elimina el estudiante cuyo valor en el campo llave es igual al dado como argumento (matricula) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no). NOTA: Este método debe devolver false en caso de que haya registros en la tabla cargaestudiante asociados al estudiante a eliminar.
- **public boolean modificaEstudiante(Estudiante e)** Este método modifica todos los datos de un estudiante a excepción de la matrícula en la tabla estudiante, los cuales están contenidos en el objeto e recibido como argumento (note que puede ser que no se pueda modificar el objeto por lo que regresa true si se pudo o false si no)

- **public List<DetalleCargaestudiante> obtenCargaEstudiante(String matricula)** Este método devolverá la carga del estudiante en forma de un objeto de clase DetalleCargaestudiante cuya matrícula es la dada como argumento, la carga es consultada de las tablas cargaestudiante, materia y periodoescolar.
- **public boolean agregaCargaEstudiante(Cargaestudiante c)** Este método agrega la carga del estudiante representada por el objeto c a la tabla cargaestudiante (note que puede ser que no se pueda agregar el objeto por lo que regresa true si se pudo o false si no). El valor de la columna asociada al atributo idCargaEstudiante es autoincrementable y por tanto, debe obtenerse y almacenarse en el objeto c recibido una vez que se agrega el registro a la tabla.
- **public boolean eliminaCargaEstudiante(long idCarga)** Este método elimina la carga del estudiante identificada por el valor recibido (idCarga) (note que puede ser que no se pueda eliminar el objeto por lo que regresa true si se pudo o false si no)

PORCENTAJE DE CALIFICACION DE CADA ELEMENTO PARA PRACTICA 8:

CLASES DE ENTIDAD (6%):

- Agregado de la implementación de Serializable, los 3 constructores, el método equals, el método hashCode y el método toString a las clases de entidad Municipio (2.5%)

- Agregado de la implementación de Serializable, los 3 constructores, el método equals, el método hashCode y el método toString a las clases de entidad Estudiante: (3.5%)

CLASES DAO Individuales (64%):

- Implementación de Dao<Municipio> en clase DaoMunicipio: 14%

- Implementación de Dao<Estudiante> en clase DaoEstudiante: 50%

CLASE DAOEscolaresImpl: 30%

- Métodos que tienen que ver con Estudiante: 10%

- Método obtenMunicipios: 20%

PORCENTAJE DE CALIFICACION DE CADA ELEMENTO PARA PROGRAMA 1:

CLASES DE ENTIDAD (6%):

- Agregado de la implementación de Serializable, los 3 constructores, el método equals, el método hashCode y el método toString a las clases de entidad Estado, Municipio, Carrera y Materia: (0.5% cada clase que cumpla con todo lo indicado)

- Agregado de la implementación de Serializable, los 3 constructores, el método equals, el método hashCode y el método toString a las clases de entidad PeriodoEscolar, Profesor y Estudiante: (0.6% cada clase que cumpla con todo lo indicado)

- Agregado de la implementación de Serializable, los 3 constructores, el método equals y el método hashCode a las clases de entidad Cargaprofesor y Cargaestudiante: (0.3% cada clase que cumpla con todo lo indicado)

- Agregado de la implementación de Serializable, los 3 constructores (que reciben lo mismo que los constructores de la clases de la que heredan), los getters y setters para los campos extra a los heredados y el método toString a las clases de entidad DetalleCargaprofesor y DetalleCargaestudiante: (0.8% cada clase que cumpla con todo lo indicado)

CLASES DAO Individuales (64%):

- Implementación de Dao<Estado> en clase DaoEstado: 3%
- Implementación de Dao<Municipio> en clase DaoMunicipio: 2%
- Implementación de Dao<Cargaestudiante> en clase DaoCargaestudiante: 5%
- Implementación de Dao<Cargaprofesor> en clase DaoCargaprofesor: 5%
- Implementación de Dao<Carrera> en clase DaoCarrera: 10%
- Implementación de Dao<Materia> en clase DaoMateria: 10%
- Implementación de Dao<Periodoescolar> en clase DaoPeriodoescolar: 10%
- Implementación de Dao<Estudiante> en clase DaoEstudiante: 7%
- Implementación de Dao<Profesor> en clase DaoProfesor: 12%

CLASE DAOEscolaresImpl: 30%

- Métodos que tienen que ver con Estado: 0.8%
- Métodos que tienen que ver con Carrera: 3.2%
- Métodos que tienen que ver con Materia: 3.2%
- Métodos que tienen que ver con Periodoescolar: 3.2%
- Métodos que tienen que ver con Profesor: 3.2%
- Métodos que tienen que ver con Estudiante: 0.6%
- Métodos que tienen que ver con Cargaprofesor: 1.6%
- Métodos que tienen que ver con Cargaestudiante: 1.6%
- Método obtenMunicipios: 0.6%
- Método obtenCargaProfesor: 6%
- Método obtenCargaEstudiante: 6%

PUNTOS EXTRA: 10%

- Si durante el desarrollo del programa realizan commits atómicos (es decir, que representan el cambio de una unidad de código, en este caso un método) que describan de manera adecuada el cambio realizado obtendrán 10 puntos extra

LA CALIFICACION QUE SE OBTENDRÁ EN CADA CASO AL MOMENTO DE SUBIRLO A GITHUB ES LA QUE SE GENERA POR EL CODIGO DE LAS PRUEBAS, NO SON LOS PUNTOS QUE GITHUB PONE

RECUERDEN QUE LOS TRABAJOS DE PROGRAMACION NO SON REEMPLAZABLES LO CUAL SIGNIFICA QUE SI NO SON ENTREGADOS NO HABRA FORMA DE RECUPERAR EL PORCENTAJE DE LA CALIFICACION FINAL QUE LES CORRESPONDE.

La fecha y hora límite para hacer el push final es a las 23:59 hrs del 21 de marzo del 2021. NO SE ACEPTARÁN PROGRAMAS ENVIADOS POR CORREO ELECTRÓNICO. El nombre que utilice para las clases, atributos y métodos deben seguir las convenciones mencionadas en clase, si usa identificadores tales como xxx, xya, o similares el programa se le asignará una calificación reprobatoria. Finalmente, si se reciben programas copiados, todas las personas que así lo entreguen recibirán calificación reprobatoria **EN LA MATERIA, NO SOLAMENTE EN EL PROGRAMA**.